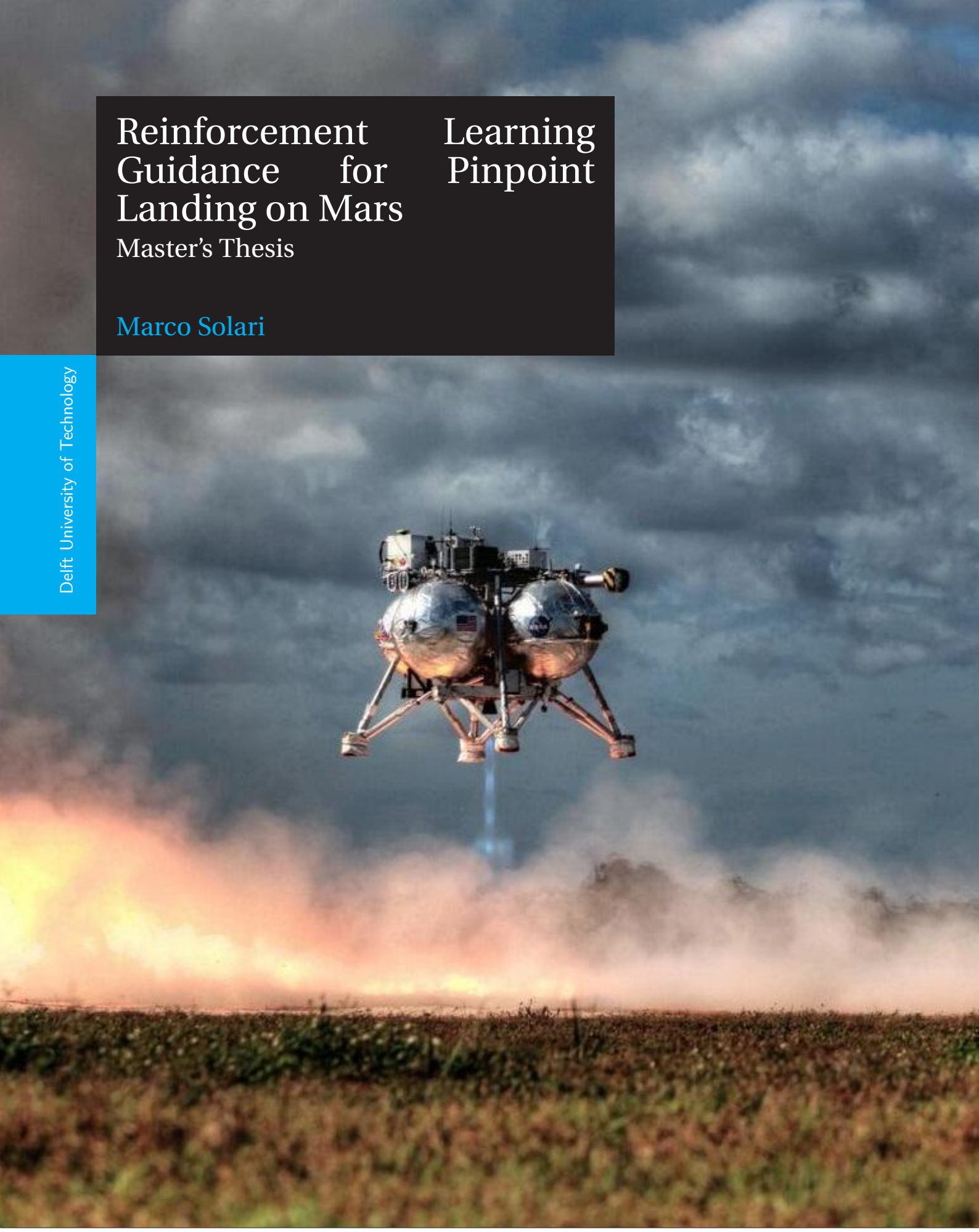
Reinforcement Learning Guidance for Pinpoint Landing on Mars

Master's Thesis

Marco Solari

Delft University of Technology



TU Delft Delft University of Technology

Challenge the future

Cover image credit: NASA (@MorpheusLander Twitter feed)
NASA's Morpheus Project – a prototype for vertical landing and takeoff for other planets – during a free flight test on Dec. 10, 2013.

Reinforcement Learning Guidance for Pinpoint Landing on Mars

Master's Thesis

by

Marco Solari

in partial fulfillment of the requirements for the degree of

Master of Science
in Aerospace Engineering

at the Delft University of Technology,
to be defended publicly on Wednesday, July 26, 2017 at 9:30 AM.

Student number:	4327896	
Project duration:	June 1, 2016 – July 26, 2017	
Supervisor:	Dr. ir. E. Mooij,	TU Delft, Astrodynamics and Space Missions
	Ir. S. Woicke,	TU Delft, Astrodynamics and Space Missions
Thesis committee:	Prof. Dr. ir. P. Visser,	TU Delft, Astrodynamics and Space Missions
	Dr. ir. G. de Croon,	TU Delft, Control and Simulation

This thesis is confidential and cannot be made public until July 25, 2018.

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft Delft University of Technology

SUMMARY

For future planetary landing missions, the capability of targeting a specific point on the planet's surface, and precisely and safely steer the lander to that point, is needed. This is known as pinpoint landing, and will enable surface rendezvous missions, as well as landing at non-inherently safe and unexplored Landing Sites (LSs), which are usually more scientifically interesting. New systems are currently under development for achieving these objectives, namely Terrain Relative Navigation (TRN) and Hazard Detection and Avoidance (HDA). However, these systems' performance depends on trajectory-related factors, such as LS visibility and the capability of handling divert maneuvers. Besides, the increasingly large mass of planetary landers makes fuel optimality an important aspect as well, as the mass capability of Entry, Descent and Landing (EDL) systems is currently at its limit for Mars landing missions. Future guidance systems should therefore be able to satisfy all these requirements.

This study focuses on the Powered Descent (PD) phase of a Mars landing, that is, the last portion of the EDL procedure, as starting from backshell separation until Touchdown (TD) on the surface. In particular, a suitable Powered Descent Guidance (PDG) is required, which can comply with the requirements mentioned above. From a literature review, a Reinforcement Learning (RL) method, with a Neural Network (NN) policy model, emerged as a promising solution to this problem. The selected method is based on the work of Gaudet and Furfaro [2014], who implemented a stochastic search algorithm to optimize the NN parameters by simulation in an uncertain environment. Contributions of this research to the existing body of knowledge are the extension of the method to a 3-dimensional environment and, most notably, the inclusion of pointing constraints, as needed by the TRN and HDA systems.

A RL method was chosen due to its alleged robustness against environmental disturbances, and because it provides a straightforward framework for the implementation of additional constraints. The results show that the optimized NN policy is indeed robust against perturbations, and generalizes very well to regions of the state space outside its training domain, which makes it suitable to handle divert maneuvers, as commanded by the HDA system. The optimization procedure is slow, and must therefore be carried out offline. However, once trained, the NN is extremely light, and could therefore run on-board in real-time. Besides, the stochastic search optimization method has been substantially improved in this work by means of a hybrid policy modeling approach, which makes use of both NN and Apollo guidance solutions. This implementation brought about improvements in accuracy, propellant consumption and optimization convergence speed.

However, while good results could be achieved for a baseline scenario, it has been very challenging to achieve acceptable performance in terms of propellant consumption and pointing accuracy for the full scenario. Besides, the issue of verifiability of the NN policy needs further attention, as the optimization produces NN outputs which are hard to inspect. Finally, another critical issue is that of large rotational rates, which are not easily constrained within this framework, and would result in unfeasible trajectories if rotational dynamics were accounted for. In spite of these issues, the results obtained in this work are promising, as they show the capability of this RL method to successfully include pointing constraints. However, further work is necessary to efficiently extend the method to a full scenario. Besides, as a future step, the guidance system should be designed with rotational dynamic constraints in mind, so as to yield actually feasible trajectories. In particular, a 6-Degrees of Freedom (DoF) guidance framework and the implementation of a value-function-based method for policy optimization are believed to be beneficial to this end.

PREFACE

This work is submitted in partial fullfilment of my Master's degree in Aerospace Engineering at the Delft University of Technology. It consists of the design, implementation and analysis of a guidance algorithm for powered descent on Mars, which is done by applying exciting and novel machine learning techniques. I was new to both these topics when I started this project. With hindsight, I am very grateful I had the chance to take this path.

For this, I am thankful to my supervisors, Dr. Erwin Mooij and Ir. Svenja Woicke, who first guided me in this direction. In particular, I thank Svenja, my daily supervisor, for her patience during this (long) period and for inviting me to attend the IPPW conference, which has been of great motivation to me. I thank Erwin for his precious inputs and prompt replies, and for allowing me the privilege of working two extra days on this report.

I would also like to thank Ir. Jacco Geul, for his useful feedback on machine learning and neural networks during our meeting, and Prof. Pieter N.A.M. Visser and Dr. Guido de Croon, for agreeing to be part of my thesis committee.

These years in Delft have also been full of new people and friends, too many to mention here. All I can say is, I am very thankful to you all for all the good times we have had together. A special, heartfelt thanks goes to the people I have been sharing a roof with during the last two years. I am not sure I would have made it without you. Finally, as I could have never undertaken this journey without their help and support, my parents should know they have all my gratitude and love.

Marco Solari
Delft, July 2017

CONTENTS

List of Acronyms

ACM	Apollo Command Module.
ALHAT	Autonomous precision Landing and Hazard Avoidance Technology.
ANN	Artificial Neural Network.
BL	Baseline.
CoM	Center of Mass.
CPU	Central Processing Unit.
DCM	Direction Cosine Matrix.
DIMES	Descent Image Motion Estimation Subsystem.
DoF	Degrees of Freedom.
EDL	Entry, Descent and Landing.
EDM	EDL Demonstrator Module.
EKF	Extended Kalman Filter.
EoM	Equations of Motion.
ESA	European Space Agency.
ESP	EDM Surface Platform.
FA	Function Approximator.
FOV	Field of View.
GMM-2B	Goddard Mars Model 2B.
GNC	Guidance, Navigation and Control.
GPM	Gauss Pseudospectral Method.
GPOPS	Gauss Pseudospectral Optimal Control Software.
HANNA	Hybrid Apollo guidance and NN Approach.
HDA	Hazard Detection and Avoidance.
HG	High Gate.
HiRISE	High Resolution Imaging Science Experiment.
IC	Initial Condition.
IMU	Inertial Measurement Unit.
LKF	Linear Kalman Filter.
LoS	Line of Sight.
LS	Landing Site.
MAV	Mars Ascent Vehicle.
MC	Monte Carlo.
MCD	Mars Climate Database.
MER	Mars Exploration Rover.
MESUR	Mars Environmental Survey.

MNN Multi-layer Neural Network.
MOLA Mars Orbiter Laser Altimeter.
MREP Mars Robotic Exploration Preparation.
MRO Mars Reconnaissance Orbiter.
MSL Mars Science Laboratory.
MSR Mars Sample Return.

NASA National Aeronautics and Space Administration.
NLP Non-Linear Programming.
NN Neural Network.
NPAL Navigation for Planetary Approach and Landing.

OCP Optimal Control Problem.
OCS Optimal Control Software.

PC Pointing Constraint.
PD Powered Descent.
PDG Powered Descent Guidance.

RAD Rocket Assisted Deceleration.
RBF Radial Basis Function.
RL Reinforcement Learning.
RMS Root Mean Square.
RT Retargeting.

S/C Spacecraft.
SNOPT Sparse Nonlinear OPTimizer.

TD Touchdown.
TGO Trace Gas Orbiter.
TIRS Transverse Impulse Rocket Subsystem.
TPBVP Two-Point Boundary-Value Problem.
TRL Technology Readiness Level.
TRN Terrain Relative Navigation.

V&V Verification and Validation.

List of Symbols

Reference Frames

Symbol	Description	Units
$\mathcal{AA}$	Airspeed-based aerodynamic frame	-
$\mathcal{AG}$	Groundspeed-based aerodynamic frame	-
$\mathcal{BA}$	Auxiliary aerodynamic frame	-
$\mathcal{BP}$	Nominal propulsion frame	-
$\mathcal{B}$	Body-centered frame	-
$\mathcal{G}$	Guidance frame	-
$\mathcal{I}$	Mars-centered inertial frame	-
$\mathcal{P}$	Propulsion frame	-
$\mathcal{R}$	Mars-centered rotating frame	-
$\mathcal{V}$	Vertical frame	-
$\mathcal{W}$	Wind-based frame	-

Greek Symbols

Symbol	Description	Units
α_A	Angle of attack	rad
$\alpha_{A,tot}$	Total angle of attack	rad
β_A	Angle of sideslip	rad
$\Delta\varepsilon_{LS}$	Total ε_{LS} span for a set of trajectories	rad
γ	Landing failure rate	-
λ	Traces decay factor	-
σ_0	Initial condition standard deviation	-
μ	Gravitational parameter	m^3/s^2

ω	Instantaneous rotational rate magnitude	rad/s
ω_{cb}	Central body spin rate	rad/s
ϕ	Roll angle	rad
ψ	Yaw angle	rad
ψ_{TD}	Azimuth of the attitude residual at TD	rad
ψ_T	Thrust azimuth angle	rad
ρ	Density	kg/m^3
θ	Pitch angle	rad
θ_{FOV}	FOV width	rad
$\tilde{\varepsilon}_{TD}$	Off-vertical attitude residual at TD	rad
ε_{FOV}	Camera boresight elevation angle in $\mathcal{B}$-frame	rad
ε_{LS}	LS elevation angle in $\mathcal{B}$-frame	rad
ε_{SC}	Spacecraft (S/C) elevation angle in $\mathcal{G}$-frame	rad
ε_T	Thrust elevation angle	rad

Roman Symbols

Symbol	**Description**	**Units**
$\bar{q}$	Dynamic pressure	$\frac{\text{kg}}{\text{ms}^2}$
$\mathbf{a}$	Acceleration vector	m/s^2
$\mathbf{C}$	Direction cosine matrix	-
$\mathbf{F}$	Force vector	N
$\mathbf{g}$	Gravitational acceleration vector	m/s^2
$\mathbf{r}$	Position vector	m
$\mathbf{v}$	Velocity vector	m/s
$\mathbf{x}$	State vector	-
$\mathbf{x}_0$	Initial condition vector	-
C_a	Axial aerodynamic coefficient	-
C_D	Drag coefficient	-

C_L	Lift coefficient	-
C_S	Side force coefficient	-
D	Drag	N
g_0	Reference gravity	s
h	Altitude	m
I_{sp}	Specific impulse	s
L	Lift	N
L_z	Vertical scale parameter	m
m_0	Wet mass	kg
m_E	Entry mass	kg
m_L	Landed mass	kg
m_P	Propellant consumption	kg
m_P^*	Optimal propellant consumption	kg
m_{prop}	Propellant mass	kg
R	Total reward	-
r	Reward	-
R_e	Equatorial radius	m
S	Side aerodynamic force	N
S_{ref}	Reference surface area	m^2
T	Thrust	N
t	Time	s
t_{go}	Time-to-go	s
V	Velocity magnitude	m/s
V_A	Airspeed	m/s

1

INTRODUCTION

The main requirement that past planetary landing missions imposed on the guidance system of the lander was that of soft landing. The state at the entry interface entirely determined the LS and the error ellipse. Only recently was the issue of a guided EDL addressed, as the Mars Science Laboratory (MSL) performed a guided entry, and so managed to greatly reduce the landing dispersion [Way et al., 2007]. However, to further increase the pinpoint landing capabilities of future landers, non-inertial navigation systems such as Terrain Relative Navigation (TRN) must be introduced. In fact, the main error source at TD is currently made up by the navigation system, which during the EDL sequence only relies on the dead reckoning provided by drift-corrupted Inertial Measurement Units (IMUs), and whose error thus increases with time. Additionally, the way current landers deal with hazards on the landing surface is hazard tolerance, rather than hazard avoidance. So, the implementation of Hazard Detection and Avoidance (HDA) functions is an important requirement for future planetary missions as well, as it would allow for landings at more hazardous LSs, which are often more scientifically interesting than inherently safe areas, and would thus relax the requirements regarding the hazard-tolerant design of the lander [Johnson et al., 2008]. Another objective enabled by means of these technologies is that of surface rendez-vous with existing assets on the planet, an endeavor which requires a substantial level of accuracy.

The performance of the TRN and HDA systems is a function of trajectory-dependent variables, such as range, path angle and LS visibility [Paschall et al., 2009]. For instance, the sensing instrument should be pointing at the LS when HDA is to be performed, whereas the elevation angle of the vehicle as seen from the LS should be close to a preferred value, which depends on the sensor employed. These requirements translate into constraints for the guidance system: a glide slope constraint and a Line of Sight (LoS) constraint, which limits the vehicle's attitude to a cone around a specified direction. A suitable guidance law must therefore be able to accommodate these constraints.

At the same time, a second issue of fundamental importance is that of fuel consumption. Again, none of the previously flown guidance laws was explicitly designed for fuel optimality, although schemes like the Apollo guidance provide a good approximation for close-to-nominal trajectories. However, future planetary landers will require substantial divert capabilities, as commanded by the hazard avoidance subsystem [Açıkmeşe et al., 2012]. For such off-nominal trajectories, fuel optimality will thus be an essential feature. Furthermore, fuel savings can either raise the maximum achievable LS altitude, allow for a more massive payload, or reduce the launch vehicle requirements, thus bringing down the mission cost.

While what has been said holds for every landing scenario, the challenges posed by an atmospheric body like Mars are certainly greater than those of an airless body landing case. In particular, in the case of Mars, the planetary environment forced the EDL technology capabilities to their limit

for MSL-class payloads, so that fuel savings and a robust guidance law are all the more critical [Braun and Manning, 2007]. The focus of this work will be on PDG for Mars landing scenarios. The powered descent phase is defined as beginning after parachute release and backshell separation, all the way to TD on the surface. For this problem, the matters of robustness to environmental disturbances, un-modeled dynamics, state knowledge uncertainties and Initial Conditions (ICs) dispersion need to be addressed.

From what has been said so far, a mission need, or problem, can be expressed as follows:

What kind of PDG system is needed to achieve soft landing with pinpoint accuracy (less than 100 m) on Mars?

As stemming from the literature review preceding this thesis work, a project goal can be written in terms of the method selected, namely a NN guidance algorithm, to be trained within a RL framework. As such, the main objective of this work can be defined as that contributing to the research on the fully constrained optimal landing guidance problem by implementing the proposed solution and testing its performance within a computer-based simulation environment.

From the literature review, NNs and RL did emerge as the selected approach to the design of a guidance system that can comply with the desired requirements. A RL learning framework has been chosen given its alleged robustness against environmental disturbances, a major element of a Mars landing scenario, and because it provides a straightforward framework for the implementation of pointing constraints. The design will mostly be based on the work done by Gaudet and Furfaro [2014], who implemented a stochastic search algorithm to solve the RL problem, while modeling the control policy with a NN. The contributions of this study to the current body of knowledge constists, most notably, of the extension of the method to a 3-dimensional environment and the enforcement of pointing constraints, which have not been implemented by Gaudet and Furfaro.

The main research question is therefore formulated as:

Can a NN-based PDG system for terminal descent on Mars, trained within a RL framework, allow for a safe and accurate TD, while fulfilling the requirements that enable the operation of the TRN and HDA system?

To structure the research, the problem has been broken down into several subquestions, which will be outlined in detail in Chapter 2, together with a brief overview of past and future Mars landing missions, the definition of the selected mission scenario, and of the guidance system requirements. Chapter 3 presents the reference frames used throughout this work, along with the transformations needed to perform coordinate conversions, and gives the necessary theoretical background on flight dynamics, needed for the implementation of the simulation environment. The latter will be introduced in Chapter 4, where the details of the implemented models and their verification will be given. Chapter 5 presents fundamental information about the tools and methods used for the development of the guidance system, namely NNs, RL and Optimal Control Software (OCS), before moving on to detail the actual implementation of the algorithm. Chapters 6 and 7 will present the results obtained and address the research subquestions, for a baseline scenario and a full scenario, respectively. Finally, Chapter 8 will close the loop by getting back to the main research question presented in this chapter. The insight gained throughout this project will be presented, conclusions about the suitability of the chosen method will be drawn, and recommendations for future work will be made.

2

MISSION HERITAGE, SCENARIO AND REQUIREMENTS

This chapter provides an overview of past and future Mars landing missions in Section 2.1. It then moves on to define the guidance requirements and present the corresponding mission scenario in Section 2.2, based on previous missions and the objectives of future ones. Finally, Section 2.3 breaks down the main research question in several subquestions and gives an outline of the structure followed to answer them.

2.1. MARS LANDING MISSIONS

This section's aim is that of giving an overview of the programs and missions that achieved (or, at least, attempted) a landing on Mars. The focus will be on the EDL systems, and especially on the aspects of such systems related to the terminal descent, namely: the LS selection and targeting, the system's robustness against hazards (i.e., either avoidance or tolerance) and the navigation techniques employed.

Over the last decades, several missions attempted to land on Mars. Figure 2.1 shows the landing sites of the most important ones, which will be presented in the following sections. The map was constructed from Mars Orbiter Laser Altimeter (MOLA) data complemented with the LS coordinates.

On the Red Planet, the landing takes place at the end of an EDL sequence which, due to the

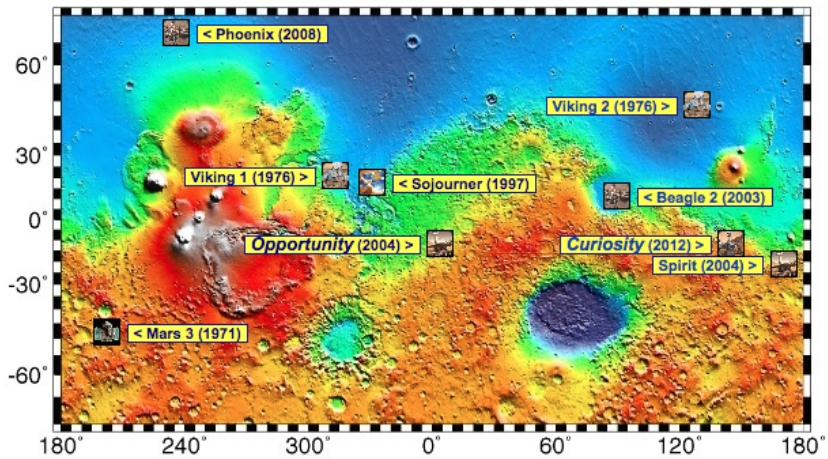


Figure 2.1: Map of the most important landing sites achieved on Mars. Source: NASA

presence of an atmosphere, is substantially different from, e.g., a lunar descent. Almost all of the surveyed entry vehicles made use of an aeroshell as main aerodynamic decelerator during the entry phase, and a parachute for further slowing down the descent and taking the lander to the initiation point of a powered descent stage. Since the early missions, the navigation system was provided by IMUs for attitude and state propagation, and radar altimeters for surface-relative altitude and velocity data during the terminal descent phase. What changes substantially from early to late Mars landing missions is ultimately the difference in altitude at the moment of parachute deployment, which is a consequence of the capsule's ballistic coefficient and whether a lifting or ballistic entry is flown.

A ballistic entry occurs when the angle of attack is zero and the total aerodynamic force acting on the vehicle is aligned with the velocity vector. On the other hand, a lifting entry is characterized by the presence of a lift force, which can be either steered (guided lifting entry) or kept in the upward direction (unguided lift-up entry).

2.1.1. PAST MISSIONS

In this section, an overview will be given of past Mars landing missions.

MARS 2, 3 & 7

Mars 2 and Mars 3 were two twin spacecrafts that were part of the U.S.S.R. space program. Launched in May 1971, they reached the Red Planet in December of the same year. Both of them were provided with a descent module, which was equipped with a parachute, retrorockets and a laser altimeter for the purpose of achieving soft landing.

Mars 2 crashed on the surface due to a malfunctioning that caused the parachute not to deploy, while Mars 3 actually became the first spacecraft to achieve soft landing on the planet. However, communications stopped only a few seconds after touchdown.

Mars 7 (also part of the U.S.S.R. space program), launched in 1973, was also equipped with a landing module, but due to a malfunctioning it separated prematurely from the flyby bus, thus missing the planet [Williams, 2015].

VIKING 1 & VIKING 2

Viking 1 and Viking 2, launched in August and September, 1975, respectively, were two twin space probes that, together, made up the National Aeronautics and Space Administration (NASA) Viking project. The LS selection was done by means of combined orbiter images with Earth-based radar data. The entry capsules flew an unguided lift-up trajectory, where the lift vector was kept vertical by means of a 3-axis attitude control system. The lifting entry strategy was chosen to maximize the altitude at which the parachute would deploy and increase the safety of the landing. For the same purpose, the landing site was also chosen in a low-elevation region. In fact, the main objective was that of achieving soft-landing [Williams, 2015].

The 3σ landing ellipse was about 280×115 km. In spite of such a large uncertainty, it was believed, a priori, that the landing ellipse was an inherently safe area. However, local hazards were detected, after landing, in the immediate proximity of the lander. No on-board hazard avoidance sensors were used for the terminal descent [Masursky and Crabill, 1981], as the landers performed a gravity turn to achieve soft landing [Ingoldby, 1978].

MARS PATHFINDER

The Mars Pathfinder Project, formerly known as Mars Environmental Survey (MESUR), was part of the NASA Discovery program. It was launched in December 1996, and reached Mars 8 months later in July 1997. Its purpose was that of demonstrating the feasibility of low-cost planetary landing and performing in-situ scientific measurements by delivering a station and a rover (Sojourner) to the surface of the planet.

Because of cost constraints (i.e., bringing the ΔV down), the capsule entered the atmosphere directly from its hyperbolic approach trajectory, as opposed to the Viking probes, which were first put into orbit around Mars. The trajectory flown was an unguided ballistic one, achieved through spin-stabilization to remove the overall effect of any leftover lift. With the selected entry angle γ_E and V_∞, the maximum LS altitude with respect to the MOLA reference surface was 2 km [Cook and McNamee, 1993], but the candidate LSs were selected to be below the reference surface. Besides, low slopes and limited roughness were sought from orbital images and radar observations extending at least for an area of about 200×70 km (3σ predicted landing ellipse).

The main difference with regards to the previous missions was the choice of delivering the payload to the ground by means of an airbag protective shell, that is, by means of a highly hazard-tolerant system. As compared to a legged lander, such a system provided better safety against rocky and inclined terrains, at the cost of an increased sensibility to winds and a more complicated rover egress operation.

MARS POLAR LANDER

The Mars Polar Lander was another NASA mission, which reached Mars in December 1999. The entry probe made use of an aeroshell for slowing down, followed by the deployment of the braking parachute. At about $h = 1.4$ km the powered descent phase was initiated to safely bring the spacecraft to the surface. After the scheduled touchdown, however, no signals were received from the lander: it is therefore not known whether the designed landing procedure was successful [Williams, 2015].

BEAGLE 2

Beagle 2 was an European lander that had been thought lost since 2003 up until 2014. No signals were received from it after the scheduled landing on the planet, but it has now been found in imagery from the Mars Reconnaissance Orbiter (MRO)'s High Resolution Imaging Science Experiment (HiRISE) instrument. The lander seems to have achieved soft landing on the planetary surface, as the images show the lander in what seems to be a partially deployed solar array configuration, a few hundreds of meters from what could be the jettisoned parachute and backshell [Webster, 2015].

MARS EXPLORATION ROVERS

The Mars Exploration Rover (MER) mission was made up by the two rovers Spirit and Opportunity, which were delivered to the surface of Mars in January 2004 by their respective landers. Similarly to the Pathfinder mission, the airbag-shrouded rovers were lowered by means of a tether, dropped, and left bouncing on the surface until they came to a halt [Williams, 2015].

The MER vehicles flew an unguided, ballistic entry trajectory, and were spin-stabilized. However, one major change was introduced in the terminal descent phase. In fact, it was found that strong steady-state winds can be quite common in Mars' lower atmosphere, and that could cause the lander to tilt and fire the retro-rockets at an angle, thus inducing an horizontal velocity component. Due to airbag-related constraints (i.e., maximum horizontal velocity tolerance), these winds represented a concern for the mission designers. The Descent Image Motion Estimation Subsystem (DIMES) was therefore developed and implemented to compensate for them.

As explained by Cheng et al. [2004], DIMES was the first tool used for horizontal velocity estimation during planetary landing, and can therefore be regarded as a precursor of the current TRN systems. It was based on descent imagery, combined with radar altimeter data and an IMU. Subsequent to taking three images between 2 km and 1.4 km altitude, the processed data would be fused in an algorithm whose outcome would be the estimated horizontal velocity component. These data could then be used to fire the Transverse Impulse Rocket Subsystem (TIRS), so as to nullify such component.

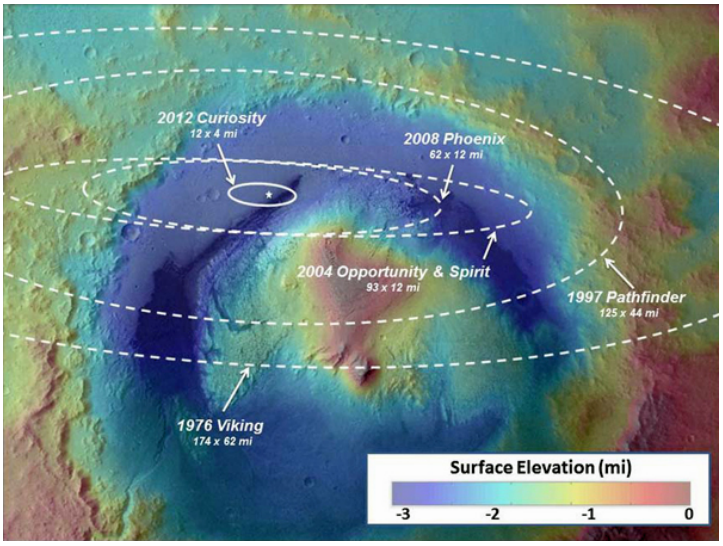


Figure 2.2: Mars landing ellipses size comparison. Source: NASA.

As a consequence, the cross-track landing dispersion was greatly reduced. Also due to improved atmospheric models, the landing ellipse shrank to approximately 150×20 km (3σ) and featured a more oblate shape when compared to those of previous missions (e.g. Viking and Pathfinder), as can be seen in Figure 2.2.

The Mars Pathfinder and the MER missions thus share the airbag-based landing system which is not suited for landing rovers of the class of the MSL. On the other hand, important differences exist between these two landers too, namely the entry mass, which was substantially smaller in the case of Pathfinder. In turn, the terminal velocity was up to 31% larger than Pathfinder's, which results in a larger Rocket Assisted Deceleration (RAD) subsystem.

PHOENIX

Phoenix landed on the Red Planet in May 2008. After slowing down, the lander started a powered descent at about 1 km altitude and soft-landed in an inherently safe area 3.5 km below the reference surface. Similarly to the MER and the Pathfinder missions, it flew a ballistic entry. In spite of this, it was 3-axis stabilized, as it was supposed to be guided in the first place. After lander separation, the S/C performed a gravity turn to reach an altitude of 50 m. As shown in Figure 2.2, the predicted 3σ landing error ellipse was about 100×20 km [Bonfiglio et al., 2011], although pre-entry prediction further brought the along-track uncertainty down to ~55 km [Desai et al., 2011].

As for the LS selection, it was performed using HiRISE imagery combined with the landing dispersion estimates. Rock size-frequency and terrain slopes estimates eventually yielded a probability of ~97 % for a safe TD within the predicted ellipse [Arvidson et al., 2008].

MARS SCIENCE LABORATORY

The MSL, also known as Curiosity, landed on Mars in August 2012. The mission's top-level requirements included landing a 850 kg payload within 10 km from the selected site, which was located 1 km above the reference surface, as opposed to those of all the previous missions, which were below the MOLA reference [Way et al., 2007]. The strict landing error requirement was due to the fact that the selected LS was situated within the Gale crater, whose dimensions and location imposed such constraint.

The achievement of such landing requirements is mainly dependent on the parachute deployment altitude and targeting accuracy on the horizontal plane (~12.5 km maximum). To achieve that, MSL performed a lifting and guided atmospheric entry, becoming the first vehicle to do so on Mars

(Viking, the only previous lifting-body capsule, had flown a full-lift-up, thus unguided, trajectory). The MSL capsule shape was derived from that of the Viking probes, while the entry guidance law was a modified version of the Apollo Command Module (ACM) re-entry guidance [Mendeck, 2011]. Three main phases can be distinguished:

- pre-bank phase;
- range control phase;
- heading alignment phase.

The pre-bank phase begins at the entry interface and ends when the drag acceleration reaches 0.1 Earth-g. The range control phase executes banking maneuvers in order to minimize the predicted down-range error at parachute deployment, while the crossrange is kept under control by executing bank reversals whenever it exceeds a specified dead-band threshold. When the velocity drops below ~1.1 km/s, the heading alignment phase is initiated that minimizes residual crossrange error but stops controlling the down-range. This phase lasts until parachute deployment.

After the parachute descent, large errors due to winds and atmospheric uncertainties have accumulated, such that robust radar measurements are needed before the powered descent phase can be initiated, which is nominally set to happen at $h = 1.5 - 2$ km and $v = 100$ m/s. The powered descent guidance logic is also divided in three stages:

- powered approach;
- constant velocity accordion;
- constant deceleration.

The powered approach phase goal is that of nullifying the horizontal velocity, so to achieve vertical attitude and have the radar pointed at the LS. The constant velocity accordion maintains the vertical velocity constant, while allowing for an altitude knowledge improvement of roughly 50 m. Finally, the constant deceleration phase brings the lander to the conditions required for initiating the Sky Crane operations ($h = 17$ m, $v = 0.75$ m/s) [Way et al., 2007].

The Sky Crane is another feature, and definitely the most innovative one, of the MSL mission's EDL system. As previously mentioned, the past missions to Mars employed either a legged lander (e.g., Viking, Phoenix) or an airbag system attached to a tether (e.g. Pathfinder, MER) for delivering their payload to the surface, whereas the Sky Crane system embedded the positive aspects of both these systems. In fact, it is both robust against inclined and rocky surface (like the airbag-based system) and against lateral winds (like the legged lander), on top of eliminating any rover egress difficulty [San Martin, 2013].

EXOMARS 2016

The ExoMars programme is an undertaking consisting of two missions developed cooperatively by European Space Agency (ESA) and Roscosmos, with contributions from NASA as well.

The first mission, or ExoMars 2016, launched on March 14, 2016, is made up of the Trace Gas Orbiter (TGO) and the EDL Demonstrator Module (EDM) Schiaparelli. As the name suggests, the orbiter is meant to study trace gases, and to conduct surface measurements. Besides, the orbiter will provide communication capabilities for the 2018 mission and beyond [Portigliotti et al., 2010]. On the other side, although it accommodates a descent camera and a small payload (DREAMS) for surface measurements during the dust storm season [Vago et al., 2015], the lander's main purpose is EDL technology demonstration for future missions [Martella et al., 2011], among which is the Doppler radar system used for ground relative altitude and velocity measurement [Bayle et al.,

Table 2.1: Terminal-descent-related data for the main Mars soft-landing missions.

	Viking 1&2	Phoenix	MSL
m_E [kg]	980	603	2700
m_L [kg]	612	364	850
m_{prop} [kg]	152	38	300
V_{HG} [m/s]	52	56	100
h_{HG} [m]	1600	940	1500-2000
$t_{go}@HG$ [s]	46	44	52
T_{TDE} [N]	2670 (3×)	293 (12×)	3000 (3×)
Sources	[Soffen and Thomas Young, 1972] [Cooley, 1977]	[Desai et al., 2011]	[Way et al., 2007]

2016]. The capsule, with an entry mass of about 600 kg, entered the atmosphere directly from its interplanetary trajectory, after separation from the TGO.

The EDM Surface Platform (ESP) features a very simple, light-weight and low volume hazard-tolerant landing system, designed to crush at TD and damp the impact with the surface. The propulsion system is made up of 9 pulsed liquid engines, with 400 N of thrust each, and a total propellant mass of 39 kg [Bayle et al., 2016]. The ESP begins the powered descent at approximately $h = 1400$ m and $v = 80$ m/s, performing a gravity turn for ~30 s [Martella et al., 2011]. The Guidance, Navigation and Control (GNC) system provides control of the ESP up until ~1.5 m above the surface, at which point the engines are cut off and the lander drops to impact the surface with ~4 m/s and 1 m/s of residual vertical and horizontal velocity, respectively Martella et al. [2011].

Unfortunately, the EDL sequence was not successful, as the lander impacted the surface with a large residual velocity. The high rotational rates under parachute led to IMU saturation, which lasted longer than expected. This, in turn, caused an incorrect propagation of the lander state and, thus, a failure in completing the descent and landing procedure.

SUMMARY

Table 2.1 summarizes the typical High Gate (HG) conditions and other powered-descent-related quantities for the some of the main Mars missions. The table only shows missions for which soft landing was not achieved by means of highly hazard tolerant designs. That is, Pathfinder and the MER missions are not included, as the airbag landing system made them very tolerant with respect to, e.g., large terminal velocities. As such, their terminal descent parameters are not representative of the kind of mission scenario which will be assumed for this work. The table rows refer to entry mass, landed mass, available propellant mass, velocity at HG, HG altitude, time-to-go at HG and maximum thrust of the terminal descent engine, respectively. The last row provides the corresponding sources. These data will be useful when defining guidance requirements and mission scenario in Sections 2.2.1 and 2.2.2, respectively.

Finally, Table 2.2 reports mission data from a more comprehensive EDL point of view. The most noteworthy quantities are TD velocity requirements, which will be referenced to in Section 2.2.2 as well, and rock height capability. Both these quantities show substantially different values for the Pathfinder and MER missions from all the other missions due to the different kind of hazard-coping design choice. Landed ellipse sizes also show the great improvement in landing accuracy that became possible with the introduction of DIMES with the MER missions.

2.1.2. FUTURE MISSIONS TO MARS

From what has been said so far, it is clear that, historically, LS selection has been based on a combination of orbital images for the generation of hazard maps and landing dispersion predictions. No capability of autonomously assessing the best LS was present on any of the missions presented in the previous sections. Additionally, inertial navigation techniques are not sufficient for the purpose

Table 2.2: EDL data for past Mars landing missions [Manning and Adler, 2005].

Landing Year:	1976	1976	1997	2004	2004	2008	2010
Mission:	**Viking 1**	**Viking 2**	**MPF**	**MER-A (Spirit)**	**MER-B (Opportunity)**	**Phoenix (planned)**	**MSL (planned)**
Entry From	Orbit	Orbit	Direct	Direct	Direct	Direct	Direct
Entry Velocity (km/s)	4.7	4.7	7.26	5.4	5.5	5.67	7.47
Orbital Direction	Posigrade	Posigrade	Retrograde	Posigrade	Posigrade	Posigrade	TBD
Entry Flight Path Angle (deg)	-17	-17	-14.06	-11.49	-11.47	-12.5	-14.6
Ballistic Coefficient (kg/m^2)	64	64	63	94	94	70	100
Entry Mass (kg)	992	992	584	827	832	600	2700
Entry Attitude Control	3-axis RCS	3-axis RCS	2 RPM passive	2 RPM passive	2 RPM passive	3-axis RCS	3-axis RCS
Entry Lift Control	C.M. offset	C.M. offset	no offset	no offset	no offset	C.M. offset	C.M. offset
Entry Guidance	Unguided	Unguided	Unguided	Unguided	Unguided	Unguided	Apollo guidance
Lift to Drag Ratio	0.18	0.18	0	0	0	0.06	0.22
Aeroshell (Heatshield) Diameter (m)	3.5	3.5	2.65	2.65	2.65	2.65	4.6
Heat Shield Geometry	70 deg cone	70 deg cone	70 deg cone	70 deg cone	70 deg cone	70 deg cone	70 deg cone
Heat Shield TPS	SLA-561	SLA-561	SLA-561	SLA-561	SLA-561	SLA-561	SLA-561
Peak Heating Rate (W/cm^2)	26	26	80-100	50	50	50	140?
Parachute Diameter (m)	16	16	12.5	14	14	12.4	19.7
Drag Coefficient (approx.)	0.6	0.6	0.4	0.4	0.4	0.6	0.6
Parachute Deploy Mach No.	1.1	1.1	1.57	1.77	1.77	1.6	2
Parachute Deploy Dyn.Pressure (Pa)	350	350	585	725	750	420	750
Parachute Deploy Altitude (km)	5.79	5.79	9.4	7.4	7.4	9	6.5
Altitude Sensing	RADAR Altimetry	RADAR Altimetry	RADAR Altimetry	RADAR Altimetry	RADAR Altimetry	RADAR Altimetry	RADAR Altimetry
Altitude Sensing Range (km)	137	137	1.6	2.4	2.4	1.6	6
Horizontal Velocity Sensing	Doppler RADAR	Doppler RADAR	none	Descent Imaging/IMU	Descent Imaging/IMU	Side-looking RADAR	Doppler RADAR
Terminal Descent Decelerator	Throttled Bi-prop	Throttled Bi-prop	Solid Rockets	Solid Rockets	Solid Rockets	Pulsed Bi-prop	Throttled Bi-prop
Horizontal Velocity Control	Throttled Bi-prop	Throttled Bi-prop	passive	lateral SRMs	lateral SRMs	Pulsed Bi-prop	Throttled Bi-prop
Touchdown Vertical Velocity (m/s)	2.4	2.4	12.5	8	5.5	2.4	<1
Touchdown Horizontal Velocity (m/s)	1	1	20 (design)	11.5	9	<1	<0.5
Touchdown Attenuator	3 legs	3 legs	4-pi Airbag	4-pi Airbag	4-pi Airbag	3 legs	6 wheels
Touchdown Rock Height Capab. (cm)	20	20	50	50	50	20?	100
Touchdown Sensor	Leg crush motion	Leg crush motion	none	none	none	Leg crush motion	Off Load
Landed Ellipse Major axis (km)	280	280	200	80	80	260	20
Landed Ellipse Minor axis (km)	100	100	100	12	12	30	20
Landing Site Elevation (km MOLA)	-3.5	-3.5	-2.5	-1.9	-1.4	-3.5	2

of achieving pinpoint landing, which is one of the main requirements for next-generation landers: novel methods like TRN and HDA will thus be necessary to integrate the measurements provided by the IMU. As these techniques have never been implemented on a real mission yet, this section will give an overview of the studies that have been done regarding their application to Mars landing missions, and the constraints that these systems pose on the S/C dynamics. Various projects are being carried out to develop and raise the Technology Readiness Level (TRL) of these technologies. Currently, the main ones are ESA's Navigation for Planetary Approach and Landing (NPAL) project (TRN only) and NASA's Autonomous precision Landing and Hazard Avoidance Technology (ALHAT) project (TRN and HDA). They are designed for landing scenarios on both airless and atmospheric bodies.

For the sake of completeness, also future missions that do not bring a direct contribution to the development of these systems will be briefly covered.

INSIGHT

The Interior Exploration using Seismic Investigations, Geodesy, and Heat Transport (InSight) mission is a NASA mission scheduled for launch in May 2018, and meant to study the interior of Mars. Its primary objective is that of putting a lander on the surface and, subsequently, the deployment of two instruments on board of it [Abilleira et al., 2014].

As for the EDL strategy adopted, the sequence is very similar to that of the Mars Phoenix lander. Lander separation from the backshell occurs at 1300 m altitude with 62 m/s residual velocity, followed by a gravity turn trajectory. No particular remarks are there to be made regarding advanced EDL technologies.

EXOMARS

The ExoMars programme is made up by two missions. The first one, already presented in Section 2.1.1, has reached Mars in October, 2016. The second mission, or ExoMars 2020, scheduled for launch in July, 2020, has the main objective of delivering a 300 kg-class rover and an instrument

platform to the surface, with the purpose of investigating whether life does or did exist on Mars. It will make use of a drill to collect soil samples down to a 2 m depth. As opposed to the 2016 mission's EDM, the 2020 lander will be equipped with throttled liquid engines, and legs will be used for TD [Vago et al., 2015].

MARS 2020

Mars 2020 is a flagship NASA missions scheduled for launch in July 2020. The EDL architecture will be based on that of MSL. The most important improvements with respect to its predecessor will be the addition of a TRN system to enable landing at less inherently safe surface areas, and a safe landing site selection system, which will identify a surface location with an hazard clearance compatible with the accuracy of the navigation estimate.

Hazard detection is still performed from orbital imagery, although an enhanced hazard detection system has been developed which makes use to machine learning to tune the parameters on which the detection algorithm is based. Field tests on the TRN algorithm show the capability of achieving 40 m 3σ position estimates, which allow the avoidance of 100 m-class hazards identifiable from orbit. During the actual descent and landing procedure, TRN will be performed in two steps (a coarse and a fine one) at altitudes between 3000 m and 2300 m, that is, before backshell separation, over a time span of 10 s [Johnson et al., 2015]. Both position, velocity and attitude will be estimated. Safe LS selection will take place during the free-fall section following backshell separation and prior to engine ignition. Finally, the guidance system, inherited from MSL, will perform the divert maneuver if necessary, for which ~ 35 kg of propellant were allocated.

MARS SAMPLE RETURN

The results obtained from the scientific payload of the ExoMars missions will be of fundamental importance for the future international endeavor of returning soil samples from Mars to Earth [Vago et al., 2015].

In fact, the long term objective of ESA's Mars Robotic Exploration Preparation (MREP) Programme is the Mars Sample Return (MSR) mission, which is a necessary step to take in preparation to future human exploration missions. The proposed mission is made up of three segments:

- a sample caching mission;
- the MSR Lander mission;
- the MSR Orbiter mission.

As part of the MSR Lander mission, the Mars Ascent Vehicle (MAV) would be placed on the surface of the planet. That would mean an entry mass of approximately 2200 kg, which would require a guided entry phase, as opposed to the spin-stabilized landers of the ExoMars programme. Besides, while a landing ellipse of approximately 100 × 15 km is envisioned for the 2018 ExoMars lander, a landing accuracy of 10 km is estimated for the MSR Lander [Buonocore et al., 2011].

2.2. GUIDANCE REQUIREMENTS AND MISSION SCENARIO

On Mars, the presence of an atmosphere introduces additional errors and challenges as compared to a descent on an airless body, as atmospheric density profile, winds and spacecraft modeling are accurate only to a certain degree. These errors add up to the persisting navigation errors, mainly driven by map-tie errors when modern navigation systems (e.g. TRN) are concerned [Knocke et al., 2004]. In particular, given the tenuous nature of Mars' atmosphere, sensitivity to density variations is increased as compared to an entry in a thicker atmosphere [Uhlig et al., 2015]. As an example, the contribution of the different error sources to the MER landed accuracy is shown in Figure 2.3,

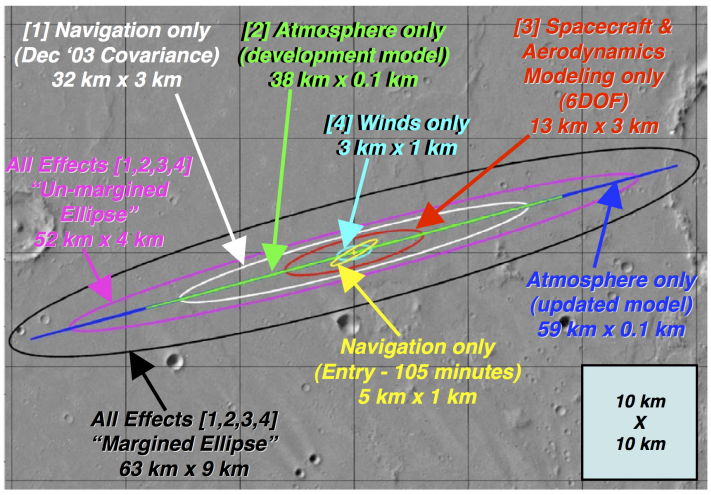


Figure 2.3: Error sources contributions to the landing ellipse [Knocke et al., 2004].

from which one can compare the dispersion for a landing on an airless body (navigation only) and on Mars.

Intermediate and large scale variations (i.e., weather and season) play a major role in the EDL system design, whereas small scale turbulences affect the vehicle stability, the landing dispersion and the guidance and control design requirements [Justus and Braun, 2007]. However, while that is mainly the case in the entry phase (high altitudes), during the descent and landing phases the lander is mostly affected by winds, especially under parachute [Justus and Braun, 2007]. Errors in wind modeling can cause significant variations in parachute deployment altitude [Way et al., 2007], induce an horizontal drift during the parachute phase, and create stability problems [Justus and Braun, 2007].

2.2.1. GUIDANCE REQUIREMENTS

These factors impose most strict requirements on the guidance system if the precise landing requirement is to be met. Starek et al. [2016] identify the most important requirements as:

- robustness;
- real-time implementability;
- verifiability.

As round-trip communication time to Mars takes several minutes, real-time implementation is a must. This entails that the algorithms that perform the guidance function should be able to run on on-board computers (if not at the moment, at least by the time such missions will be flown). Besides, verifiability implies the definition of methods to assess the performance of the system, and metrics to express it [Starek et al., 2016].

Also, the robustness requirement is particularly important on Mars as compared to an airless body or a body with a thicker atmosphere, for the reasons given above. Robustness is the characteristic that a system must have to properly operate under off-nominal conditions. As such, a guidance law for terminal descent on Mars should be robust particularly against horizontal winds and dispersions at HG. Another consequence of Mars' tenuous atmosphere is the very low altitude at which the supersonic parachute can be deployed: that makes the terminal descent phase timeline much shorter than it would otherwise be, which means the PDG should be able to maximize the divert capability to achieve pinpoint landing, while remaining close to fuel optimality.

Besides, the overall goal of future planetary landing missions to Mars, as far as the EDL system is concerned, can be summed up as that of landing heavier payloads at higher LS elevations, in non-inherently safe areas, while possibly relaxing the requirements on the robustness of the landing platform. Also, rendezvous with existing assets on the surface might be necessary for certain missions, such as MSR.

These objectives directly entail the need for pinpoint landing (for safe landing area maximization + surface rendezvous) and soft landing conditions (less fault-tolerant design required), which are of the highest priority for the GNC system. In fact, the PDG phase of the EDL sequence is designed for delivering the payload safely to the ground within the residual velocity envelope of the TD system [Manning and Adler, 2005]. Also, pinpoint landing is defined as the capability of landing within ~100 m (3σ) from the prescribed LS. Therefore, *constraints on the velocity and position errors* at TD are required. To accomplish this, and to maximize the payload mass and/or LS altitude, the following aspects will need to be addressed when designing the next-generation GNC system for planetary landing:

- target-relative location knowledge;
- on-board hazard detection;
- propellant efficiency,

First, given the very flimsy mass capability margins that were reached with the MSL mission, *fuel-optimality* constitutes one critical requirement, especially when needed for terminal descent, due to the large multiplier [Sostaric, 2007]. This will also be required to cope with the *changing target locations* that derive from making use of the HDA system, without overly penalizing the propellant mass. Also, the algorithm should be able to run *autonomously* on-board in *real-time*.

So far, hazard detection has only been performed remotely, and terrain-relative state estimation has been based on radars for altitude and vertical velocity (Mars Pathfinder), Doppler radars for horizontal velocity (Surveyor, Apollo, Phoenix, Viking), and passive imaging with subsequent processing for horizontal velocity too (DIMES on MER) [Adler et al., 2012], which might be regarded as the mere precursor of the current TRN systems. As such, current state-of-the-art PDGs have no explicit enforcement of the state constraints that are needed to have these systems work properly [Açıkmeşe et al., 2012].

As discussed during the introduction of mission need and main research question in Chapter 1, this study focuses on the guidance system for the PD phase of the EDL procedure, which is defined as starting after parachute release and backshell separation, until TD on the surface. With this in mind, and based on the previous discussion, the following guidance system top-level requirements can be identified:

GR-01 The guidance system shall enable landing of the S/C with a vertical velocity error at TD lower than 0.2 m/s.

GR-02 The guidance system shall enable landing of the S/C with a horizontal velocity error at TD lower than 0.1 m/s.

GR-03 The guidance system shall enable landing of the S/C with a position error at TD lower than 0.5 m.

GR-04 The propellant consumption m_P shall not exceed the optimal value m_P^* by more than 10 %.

GR-05 The LS shall remain within the optical sensor Field of View (FOV) between 1500 m and 1000 m altitude.

GR-06 The guidance system shall enable one LS redesignations up to 800 m at 1000 m altitude.

GR-07 Instantaneous rotational rates shall be lower than 3^o/s.

GR-08 The off-vertical attitude residual $\tilde{\varepsilon}_{TD}$ shall be lower than 8^o.

GR-09 The guidance system shall be able to operate autonomously.

GR-10 The guidance system shall be able to operate on-board in real-time.

GR-11 The guidance system shall be verifiable.

From Table 2.2 one can see that MSL landed with less than 1 m/s and 0.5 m/s of vertical and horizontal residual velocity, respectively. These errors do include state knowledge uncertainty as well, which is not to be accounted for as long as the guidance requirements are concerned. However, Way et al. [2013] report 0.85 m/s and 0.1 m/s as the design requirements for the MSL mission for vertical and horizontal velocities at TD, respectively. On the other hand, preliminary inputs on the capabilities of the RL algorithm to be developed can be obtained from Gaudet and Furfaro [2014], which shows that the 3σ velocity error at TD is in the order of several centimeters per second. Also, the mean position error at TD is in the order of a couple of tens of centimeters. It is therefore believed that the values determined in requirements GUID-01 to GUID-03 be appropriate to the end of enabling pinpoint landing and allowing for a safe TD, while at the same time not being unrealistically strict. It is to be noted that the target value for the vertical velocity is 0.4 m/s. The target point of the guidance algorithm, as well as the corresponding design requirements are summed up in Table 2.3.

Table 2.3: Target and required values for TD conditions.

	Target values	Design requirement
Position (distance from LS) [m]	0	<0.5
Vertical velocity [m/s]	0.4	<0.6
Horizontal velocity [m/s]	0	<0.1

Similarly, Gaudet and Furfaro [2014] show that the best case propellant consumption they get is $\sim 1.12 m_P^*$. As such, defining GUID-04 as to require a propellant consumption of less than $1.1 m_P^*$ is an ambitious, yet not unattainable, objective. As for the maximum retargeting range, assuming a sensor FOV of 30°, and altitude and downrange positions of ~1500 m, it follows that the footprint at the LS will have a diameter of approximately 1500 m. This leads to a required retargeting capability of up to 800 m. Additionally, the LS is required to remain within the camera FOV from the beginning of the PDG at 1500 m until the retargeting altitude (i.e., 1000 m, see Section 2.2.2), so as to allow for HDA operations.

A requirement has been set for rotational rates as well. In fact, even though this study focuses on Center of Mass (CoM) dynamics, and implements a 3-DoF guidance, it is nonetheless important that attitude rates can be ensured to be within reasonable bounds, so that the corresponding trajectories can be considered realistic from a attitude control system point of view. The required value has been retrieved from Topcu et al. [2007], which report 3^o/s as the maximum rotational rate during the PDG phase.

Finally, Mueller et al. [2010] report desired roll and pitch values of 3^o at TD, while the acceptable values are set at 6^o. Considering a worst case scenario in which both pitch and roll take up their maximum allowed values, the total off-vertical angle can be computed by means of Eqs. (3.16) and (6.5), and is found to be $\tilde{\varepsilon}_{TD} = 8.48^o$. Considering that the above-mentioned study refers to a lunar landing, the requirement for a Mars landing should be slightly more strict, given the larger gravitational acceleration, which might lead to excessive pressure on the lander leg that first makes

contact with the surface. However, Mueller et al. [2010] also allow for a larger terminal vertical velocity (i.e., 8 ft/s ~ 2.5 m/s) than it is required here (see Table 2.3). These effects offset each other, so that assuming a requirement of 8° for the total attitude residual at TD is deemed reasonable.

2.2.2. MISSION SCENARIO

The values assumed here for the calculation of the hazard detection area are derived from the selected scenario. Such scenario is based on the one used by Gaudet and Furfaro [2014], but some modifications were made to make it more realistic, on the basis of a comparison with past Mars missions and similar studies on the topic. Table 2.4 shows the initial conditions and thrust values used for this work, as compared to the original values assumed by Gaudet and Furfaro [2014].

As can be seen, in both cases the initial altitude is fixed with no uncertainty. That is because it is assumed that it is the altitude that triggers the beginning of the PDG phase. Besides, a comparison with the values of Table 2.1 shows that an initial altitude of 1500 m is within the envelope of the MSL mission, and close to that of Viking as well. No values for the y-components are given for the reference study, as a 2-dimensional analysis is carried out by Gaudet and Furfaro [2014]. Therefore, the first difference between the original scenario and the one assumed here will be the addition of the third dimension. A cross-track position dispersion of about half the along-track one was chosen based on typical landing ellipse major- to minor-axis ratios.

As for the initial velocity, on the other hand, more changes were necessary to make the scenario more realistic. In particular, the new choice was driven by real values of flight-path angle and velocity vector at backshell separation from past missions, which turned to not be validated by the values provided by Gaudet and Furfaro [2014]. For instance, the simulations done by Desai et al. [2011] and Way et al. [2013] for Phoenix and MSL, respectively, show minimum (1%-tile) and maximum (99%-tile) flight-path angle values at lander separation of approximately -90 ° and -70 °, respectively. Additionally, Way et al. [2013] obtained a simulated velocity magnitude at HG between ~60 m/s and ~95 m/s, with an average value of ≈77 m/s for the MSL-mission. With the new vertical and downrange velocity values, the average flight-path angle is now close to that of past missions to Mars (i.e., ~ 80°). The average magnitude of the velocity vector, as computed with the components indicated in Table 2.4, is also close to the values mentioned above.

The table also shows values for wet mass m_0 and thrust bounds T_{min} and T_{max}. Initial mass has been kept constant, as it is assumed reasonable for a near future mission, given a slightly larger value than the MSL mission (m_0 ~ 1700 kg [Edquist et al., 2007]). On the other hand, thrust bounds have been increased as compared to the values used by Gaudet and Furfaro [2014], to give the lander the additional control authority it requires to handle the larger negative vertical velocities.

Finally, retargetings and pointing constraints will be included as well when simulating the full scenario. This is better described by means of Figure 2.4. As one can see, the retargeting is scheduled to take place at 1000 m altitude. This is the same altitude used in the reference study, and it is considered reasonable, as it is believed to be sufficiently high to enable a retargeting capability in compliance with requirement GR-06, while providing enough time for HDA operations. In fact, the lander reaches this altitude after approximately 7 s from the beginning of the PD. Assuming HDA operations start at the beginning of the PD, this amount of time would be sufficient for the system to scan the surface, detect possible hazards, and command a new LS if necessary (e.g., Johnson et al.

Table 2.4: Mission scenario initial conditions and thrust bounds, as compared to the reference study by Gaudet and Furfaro [2014].

	x[m]		y[m]		z[m]		v_x[m/s]		v_y[m/s]		v_z[m/s]		m_0[kg]	T[N]	
	$\square_0$	3σ	$\square_0$	3σ	$\square_0$	3σ	$\square_0$	3σ	$\square_0$	3σ	$\square_0$	3σ		min	max
[Gaudet and Furfaro, 2014]	-1760	210	-	-	1500	0	80	20	-	-	-57.5	17.5	1900	4972	13258
PRESENT WORK	-1760	105	0	50	1500	0	10	15	0	15	-75	22.5	1900	5000	16000

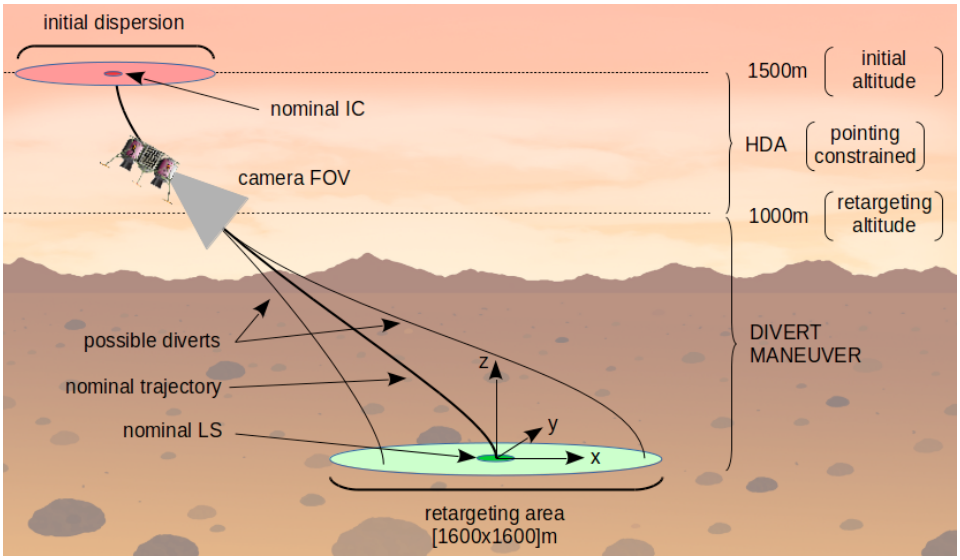


Figure 2.4: Reference mission scenario.

[2008] assume 3 s for hazard detection). As such, the LS will be enforced to remain within the camera FOV only between 1500 m and 1000 m. Given these elements, four different scenarios can be identified, which will be used throughout this work. These are:

- **baseline scenario (BL)**: neither retargetings not pointing constraints are included;
- **pointing constraint scenario (PC)**: pointing constraints are included, retargetings are not;
- **retargeting scenario (RT)**: retargetings are included, pointing constraints are not;
- **full scenario (RTPC)**: both pointing constraints and retargetings are included.

2.3. RESEARCH STRUCTURE

The main research question, presented in Chapter 1, has been formulated as:

Can a NN guidance system for terminal descent on Mars, trained within a RL framework, allow for a safe and accurate TD, while fulfilling the requirements that enable the operation of the TRN and HDA system?

The problem is broken down into several subquestions, reported below. The performance metrics investigated are, in general, position and velocity errors at TD (accuracy), propellant consumption and LS visibility.

RQ-1 Is the RL guidance suitable for efficiently, safely, and accurately steer the lander to TD under nominal conditions?

RQ-1.1 How do accuracy and propellant consumption compare to those of a traditional guidance scheme?

RQ-1.2 How does propellant consumption compare to the optimal value?

RQ-1.3 What is the envelope of initial conditions from which the lander can achieve safe landing at the desired LS?

RQ-1.4 How does the performance of the selected algorithm relative to the benchmark change when ideal navigation and/or no wind is assumed?

RQ-1.5 What are the computational requirements of the NN guidance? How do they compare to those of the benchmark algorithm?

RQ-2 Is the RL framework suitable for the inclusion of attitude constraints and for withstanding LS retargetings?

RQ-2.1 Is it possible to successfully include pointing constraints in the RL scheme? Is the RL framework a convenient one for the enforcement of such constraints?

RQ-2.2 Are rotational kinematics slow enough to provide good HDA and TRN performance?

RQ-2.3 How good are these quantities (in particular, attitude rate and LS visibility) as compared to that of the benchmark?

RQ-2.4 When only a LS redesignation (no pointing constraints) is introduced, what is the variation of:

RQ-2.4.1 the envelope of feasible initial conditions?

RQ-2.4.2 the performance metrics?

RQ-2.4.3 the reachable retargeting points?

RQ-2.5 When only pointing constraints are added (no retargeting), what is the variation of:

RQ-2.5.1 the envelope of feasible initial conditions?

RQ-2.5.2 the performance metrics?

RQ-2.5.3 the reachable retargeting points?

RQ-2.6 How do performance metrics vary when both a retargeting and the pointing constraints are included? And the envelope of initial conditions?

RQ-3 How robust is the algorithm?

RQ-3.1 How sensitive are the performance metrics to variations in environmental parameters?

RQ-3.1.1 varying wind speeds and directions

RQ-3.1.2 varying atmospheric density

RQ-3.1.3 varying gravitational acceleration

RQ-3.2 How sensitive are the performance metrics to variations in system parameters?

RQ-3.2.1 varying thrust perturbations

RQ-3.2.2 varying state knowledge uncertainties

RQ-3.2.3 varying sensor FOV

RQ-4 What is the effect of varying weights in the reward function on the performance metrics?

RQ-4.1 What is the relation between landing accuracy and fuel consumption?

RQ-4.2 What is the effect of the strictness of pointing constraints?

In order to answer the research questions above, the selected algorithm will be tested within a simulation environment, introduced in Chapter 4. The tests to be carried out can be divided into three main blocks.

First, the algorithm will be tested by means of simulations for a baseline scenario under nominal conditions, that is, without retargeting or pointing constraints. Furthermore, the first comparisons with the benchmark Apollo guidance algorithm will be carried out. In particular, the performance difference under varying disturbance magnitudes will be probed. The corresponding results mainly link back to RQ1, and will be presented in Section 6.2.

Second, after the algorithm has been extended to include the capability of handling attitude and attitude rate constraints, and LS retargetings, additional metrics can be analyzed, such as LS visibility and attitude rates. These can be used to derive the first conclusions about the suitability of the algorithm for the problem of pinpoint landing. These tests are aimed at answering RQ2, and their results will be reported in Chapter 7.

Finally, after the preliminary tests have been done, an in-depth sensitivity analysis will be carried out afterwards (RQ3). One simulation variable at a time will be varied, while keeping the other ones constant, and a simulation (or set of simulations, if the effect of a random sequence if being investigated) will be run. This is done for all relevant variables, and allows to investigate the robustness of the algorithm with regards to single parameters. Also, stress cases will be simulated to probe the performance of the algorithm under extreme initial conditions dispersions. The results will be shown in Section 7.3. The results of the tuning of the algorithm (RQ4) will be mainly presented in Section 6.1.

3

FLIGHT DYNAMICS

This chapter will give an overview of the reference frames (Section 3.1), attitude representations and frame transformations (Sections 3.2 and 3.3) and Equations of Motion (EoM) formulations (Section 3.4) used throughout this work for the development of the simulation environment (Chapter 4), and as a handy reference when needed.

3.1. REFERENCE FRAMES

As taken from Mooij [1994], the most important reference frames can be categorized into planet-centered frames and vehicle-centered frames.

The planetocentric reference frames are frames whose origin is located at the CoM of the central body. Among them, one finds:

- **Mars-centered inertial reference frame** ($\mathcal{I}$): the x- and y-axes lie in the equatorial plane, and the direction of the x-axis is given by the zero-longitude meridian at zero time. The z-axis is perpendicular to the x-y plane (pointing north), while the y-axis completes the right-handed frame.
- **Mars-centered rotating reference frame** ($\mathcal{R}$): the x- and y-axes lie in the equatorial plane, and the x-axis is in the direction of the zero-longitude meridian at any moment in time. The z-axis is perpendicular to the x-y plane (pointing north), while the y-axis completes the right-handed frame.

On the other hand, all the following reference frames originate in the CoM of the vehicle:

- **Body reference frame** ($\mathcal{B}$): the definition of this frame has been modified with respect to the one given by Mooij [1994]. Assuming a rotationally symmetric vehicle body, the z-axis is aligned with the axis of symmetry and is pointing downwards. The x-axis is perpendicular to it and is negative in the forward (or downrange) direction. The y-axis completes the right-handed frame.
- **Propulsion reference frame** ($\mathcal{P}$): the x-axis is the only relevant one. It is aligned with the thrust vector and positive in the thrust direction; its attitude with regards to the body frame is thus given by the elevation and azimuth angles of the thrust.
- **Nominal propulsion reference frame** ($\mathcal{BP}$): this is actually an auxiliary body frame, as it is fixed with respect to the vehicle. This coincides with the nominal $\mathcal{P}$-frame, that is, when no thrust misalignments are there, and will ease the modeling of these perturbations later on in Section 4.13;

- **Vertical reference frame** ($\mathcal{V}$): the z-axis is pointing to the CoM of the central body, while the x-axis lies in a meridian plane, is perpendicular to the x-axis and is positive towards the northern hemisphere. The y-axis completes the reference frame.

- **Aerodynamic reference frame**: groundspeed-based ($\mathcal{AG}$) or airspeed-based ($\mathcal{AA}$). The x-axis is aligned with the velocity vector relative to the ground or the atmosphere, respectively. The z-axis is aligned with the lift vector (based on groundspeed or airspeed variables, respectively) but has opposite direction. The y-axis completes the right-handed frame.

- **Auxiliary aerodynamic reference frame** ($\mathcal{BA}$): this is another auxiliary body frame, as it is fixed to the vehicle. It is defined in order to avoid modifications to the transformation from the $\mathcal{AA}$-frame, and, as such, the computation of the aerodynamic angles. The x- and z-axes correspond to the z- and x-axes of the $\mathcal{B}$-frame, respectively. The y-axis completes the right-handed frame.

- **Wind based** ($\mathcal{W}$): the x-axis is collinear with the wind-velocity vector, the z-axis is perpendicular and positive in the downward direction, and the y-axis completes the reference frame. In case the wind motion is not horizontal, the $\mathcal{W}$-frame can be obtained by two rotations of the $\mathcal{V}$-frame. However, vertical winds are disregarded for this work, as explained in Section 4.8, so one rotation will suffice for this transformation.

- **Guidance reference frame** ($\mathcal{G}$): surface-fixed reference frame with origin at the nominal LS. The z-axis is perpendicular to the surface and pointing upwards. The x- and y-axes are pointing south and east, respectively.

The definition of the $\mathcal{B}$- and $\mathcal{G}$-frames has been chosen for an easy interface with the $\mathcal{V}$-frame, which was left unchanged from the original formulation by Mooij [1994]. With the so-defined reference frames, a straight-forward computation of the attitude angles is available, as well as an intuitive representation of the state components, which are defined with respect to the $\mathcal{G}$-frame. In the following sections, the reason for this will be given.

By choosing the $\mathcal{G}$-frame as the reference frame in which the EoM are expressed, a flat-Mars assumption is being made. In fact, horizontal displacements for the problem at hand will be in the order of a couple of kilometers. From a quick computation, the altitude error this induces is $\sim 0.5\,\text{m}$. This has an effect on the computation of the gravitational acceleration of $\sim 10^{-6}\,\text{m/s}^2$, which, even when propagated for 100 s, has an effect of only 0.01 m on the position, and can therefore be neglected.

3.2. ROTATIONS AND ATTITUDE REPRESENTATIONS

The rotation from one reference frame to another can be expressed by means of a rotation matrix. Expressing the rotation between two frames is practically equivalent to representing the attitude of one frame with regards to a reference. Since we are interested in the attitude of the vehicle, and the $\mathcal{B}$-frame defined in Section 3.1 is fixed w.r.t. the vehicle, representing the attitude translates into representing the rotation from the chosen reference frame to the $\mathcal{B}$-frame.

Although this study focuses on 3-DoF dynamics, an analysis of the LS visibility, as coming from the HDA requirements, will be performed. Since the camera boresight is fixed with respect to the $\mathcal{B}$-frame, knowledge of this frame's attitude with respect to the LS-fixed $\mathcal{G}$-frame (that is, vehicle attitude) will enable this analysis. As such, a way of expressing such attitude is required.

Although there are several ways to represent this, which can be advantageous in terms of numbers of parameters or intuitive visualization, they are usually converted back into a 3×3 rotation matrix when an actual frame/vector transformation is to be carried out. Therefore, we will start this discussion by introducing the rotation matrix, or Direction Cosine Matrix (DCM).

DIRECTION COSINE MATRIX

A direction cosine matrix is a matrix whose entries represent the cosines of the angles between the unit vectors of the two reference frames of which it expresses the relative attitude. So, the DCM, labeled as C in the mathematical expressions, from frame $\mathcal{A}$ to frame $\mathcal{B}$ can be written as:

$$\mathbf{C}_{\mathcal{B}}^{\mathcal{A}} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \tag{3.1}$$

This result can be obtained by expressing the axes of frame $\mathcal{B}$ in $\mathcal{A}$-coordinates as

$$\mathbf{b}_i = (\mathbf{b}_i \cdot \mathbf{a}_1)\mathbf{a}_1 + (\mathbf{b}_i \cdot \mathbf{a}_2)\mathbf{a}_2 + (\mathbf{b}_i \cdot \mathbf{a}_3)\mathbf{a}_3, \quad i = 1,2,3 \tag{3.2}$$

So, expressing these three equations in matrix form, one gets:

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} = \begin{bmatrix} \mathbf{b}_1 \cdot \mathbf{a}_1 & \mathbf{b}_1 \cdot \mathbf{a}_2 & \mathbf{b}_1 \cdot \mathbf{a}_3 \\ \mathbf{b}_2 \cdot \mathbf{a}_1 & \mathbf{b}_2 \cdot \mathbf{a}_2 & \mathbf{b}_2 \cdot \mathbf{a}_3 \\ \mathbf{b}_3 \cdot \mathbf{a}_1 & \mathbf{b}_3 \cdot \mathbf{a}_2 & \mathbf{b}_3 \cdot \mathbf{a}_3 \end{bmatrix} \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{bmatrix} \tag{3.3}$$

where the 3×3 matrix is the DCM of Eq. (3.1). The matrix so defined can be used to convert the unit vectors making up the orthogonal right-handed set of frame $\mathcal{A}$ into those of frame $\mathcal{B}$, as well as any arbitrary vector expressed in $\mathcal{A}$-coordinates into the same vector expressed in $\mathcal{B}$-coordinates.

The DCM is an orthonormal matrix, that is:

$$\mathbf{C}\mathbf{C}^T = \mathbf{I} \text{ or } \mathbf{C}^{-1} = \mathbf{C}^T \tag{3.4}$$

and

$$\det \mathbf{C} = 1 \tag{3.5}$$

So, since $\mathbf{C}^{-1}$ is the inverse operator, if

$$\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} = \mathbf{C} \begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{bmatrix} \tag{3.6}$$

the following holds:

$$\begin{bmatrix} \mathbf{a}_1 \\ \mathbf{a}_2 \\ \mathbf{a}_3 \end{bmatrix} = \mathbf{C}^{-1} \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} = \mathbf{C}^T \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} \tag{3.7}$$

that is

$$\mathbf{C}_{\mathcal{B}}^{\mathcal{A}} = \mathbf{C}_{\mathcal{A}}^{\mathcal{B}^{-1}} = \mathbf{C}_{\mathcal{A}}^{\mathcal{B}^T} \tag{3.8}$$

The DCM is a therefore a 9-parameter attitude representation, and, since the minimum number of parameters to express 3-dimensional attitude is three, six of those parameters are redundant, which makes this parametrization very computationally burdensome. On the other hand, using the DCM for attitude propagation introduces no singularities [Choukroun, 2014].

EULER ANGLES

The most basic frame transformation can be obtained by rotating one reference frame around one of its own axes, by an arbitrary angle θ. When the rotation is carried out about the x-, y-, or z-axis,

the resulting rotation matrices are, respectively:

$$\mathbf{C}_1(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix} \tag{3.9}$$

$$\mathbf{C}_2(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \tag{3.10}$$

$$\mathbf{C}_3(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

Using a number of sequential rotations around the axes of the initial and intermediate frames, it is possible to express any rotation. When combining multiple rotation matrices, the resulting matrix is obtained by multiplying the matrices expressing the sequential rotations from right to left. For instance, one can write a (3,2,1) rotation from $\mathcal{A}$ to $\mathcal{B}$ as follows:

$$\mathbf{C}_{\mathcal{B}}^{\mathcal{A}} = \mathbf{C}_{\mathcal{B}}^{\mathcal{A}''}\mathbf{C}_{\mathcal{A}''}^{\mathcal{A}'}\mathbf{C}_{\mathcal{A}'}^{\mathcal{A}} \tag{3.12}$$

$$= \mathbf{C}_{1''}(\theta_1)\mathbf{C}_{2'}(\theta_2)\mathbf{C}_3(\theta_3) \tag{3.13}$$

where the first and second intermediate frames are called $\mathcal{A}'$ and $\mathcal{A}''$, respectively. That means the first rotation is executed about the z-axis of the $\mathcal{A}$ frame, the second rotation is performed about the y-axis of the $\mathcal{A}'$ frame, while the last one is a rotation about the x-axis of the $\mathcal{A}''$ frame. The matrix representing the inverse rotation is:

$$\mathbf{C}_{\mathcal{A}}^{\mathcal{B}} = \mathbf{C}_3(-\theta_3)\mathbf{C}_{2'}(-\theta_2)\mathbf{C}_{1''}(-\theta_1) \tag{3.14}$$

Sequential unit axis-rotations are used to express the frame transformations that will be presented in Section 3.3, as the angles of the unit axis-rotations are usually defined quantities which are used to intuitively parameterize the rotation. The same concept can be applied to attitude representation purposes. In that case, Euler angles are the most common set of angles used to do so.

Euler angles are defined by sequential rotations around the axes of the initial and intermediate reference frames. They are also known as the attitude angles: the roll angle φ, the pitch angle θ and the yaw angle ψ. These angles are defined by performing a (3,2,1) rotation from frame $\mathcal{A}$ (reference) to frame $\mathcal{B}$ (target)

$$\mathcal{A} \xrightarrow[\psi\,\theta\,\varphi]{3\,2\,1} \mathcal{B} \tag{3.15}$$

around the unit axes in the same fashion as shown in Eq. (3.13). Notice that the order of the rotations is important, and different sequences will yield different results. Also, although Euler angles is a generic term which could be associated to any rotation sequence, for the sake of convenience, the term Euler angles will be hereinafter used to refer to the attitude angles (yaw, pitch and roll), unless stated otherwise.

The Euler angles attitude representation is a minimal one, in that the number of parameters involved is the minimum one. They do, however, introduce a singularity in the kinematic equations, for a matrix inversion has to be performed, which requires the determinant to be different than 0. If it is not, a singularity occurs: this phenomenon is known as gimbal lock.

Rotational dynamics are disregarded for this work. However, the computation of the attitude angles is nonetheless important for aiding in the analysis of LS visibility. As such, they are inferred from the current state and commanded acceleration as explained in Section 4.6, the Euler angles represent the most convenient representation strategy, as their interpretation is straight-forward and intuitive.

EULER ANGLES AND DCM

For the rotation given in Eq. (3.15), the DCM is written as:

$$\mathbf{C}_{\mathcal{B}}^{\mathcal{A}} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi & \sin\varphi\sin\theta\sin\psi + \cos\varphi\cos\psi & \sin\varphi\cos\theta \\ \cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi & \cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi & \cos\varphi\cos\theta \end{bmatrix} \tag{3.16}$$

If, on the other hand, the DCM is known and the attitude angles are to be found, one can obtain them as

$$\begin{aligned} \theta &= -\arcsin(C_{13}) \\ \varphi &= \text{atan2}(C_{23}, C_{33}) \\ \psi &= \text{atan2}(C_{12}, C_{11}) \end{aligned} \tag{3.17}$$

3.3. FRAME TRANSFORMATIONS

In Table 3.1 reported the frame transformation matrices between the frames presented in Section 3.1, expressed in terms of unit axis-rotations, or as combinations of already defined DCMs.

Table 3.1: Frame transformations between the reference frames defined in Section 3.1.

Frames			**Expression**	**Variables involved**		
				Symbol	Unit	Description
$\mathcal{V}$	$\longrightarrow$	$\mathcal{R}$	$\mathbf{C}_{\mathcal{R}}^{\mathcal{V}} = \mathbf{C}_3(-\tau)\mathbf{C}_2(\frac{\pi}{2}+\delta)$	τ δ	[rad] [rad]	planetocentric longitude planetocentric latitude
$\mathcal{W}$	$\longrightarrow$	$\mathcal{V}$	$\mathbf{C}_{\mathcal{V}}^{\mathcal{W}} = \mathbf{C}_3(-\chi_W)\mathbf{C}_2(-\gamma_W)$	χ_W γ_W	[rad] [rad]	wind vector heading wind vector flight-path angle
$\mathcal{P}$	$\longrightarrow$	$\mathcal{BP}$	$\mathbf{C}_{\mathcal{BP}}^{\mathcal{P}} = \mathbf{C}_1(-\psi_T)\mathbf{C}_2(-\varepsilon_T)$	ψ_T ε_T	[rad] [rad]	thrust vector azimuth angle thrust vector elevation angle
$\mathcal{B}$	$\longrightarrow$	$\mathcal{R}$	$\mathbf{C}_{\mathcal{R}}^{\mathcal{B}} = \mathbf{C}_{\mathcal{R}}^{\mathcal{V}}\mathbf{C}_{\mathcal{V}}^{\mathcal{AG}}C_{\mathcal{AG}}^{\mathcal{B}}$	-	-	-
$\mathcal{BP}$	$\longrightarrow$	$\mathcal{B}$	$\mathbf{C}_{\mathcal{B}}^{\mathcal{BP}} = \mathbf{C}_2(-\pi/2)$	-	-	-
$\mathcal{P}$	$\longrightarrow$	$\mathcal{B}$	$\mathbf{C}_{\mathcal{B}}^{\mathcal{P}} = \mathbf{C}_{\mathcal{B}}^{\mathcal{BP}}\mathbf{C}_{\mathcal{BP}}^{\mathcal{P}}$	-	-	-
$\mathcal{V}$	$\longrightarrow$	$\mathcal{G}$	$\mathbf{C}_{\mathcal{G}}^{\mathcal{V}} = \mathbf{C}_2(\pi)$	-	-	-
$\mathcal{W}$	$\longrightarrow$	$\mathcal{G}$	$\mathbf{C}_{\mathcal{G}}^{\mathcal{W}} = \mathbf{C}_{\mathcal{G}}^{\mathcal{V}}\mathbf{C}_{\mathcal{V}}^{\mathcal{W}}$	-	-	-
$\mathcal{BA}$	$\longrightarrow$	$\mathcal{B}$	$\mathbf{C}_{\mathcal{B}}^{\mathcal{BA}} = \mathbf{C}_1(\pi)\mathbf{C}_2(-\pi/2)$	-	-	-
$\mathcal{V}$	$\longrightarrow$	$\mathcal{B}$	$\mathbf{C}_{\mathcal{B}}^{\mathcal{V}} = \mathbf{C}_3(-\psi)\mathbf{C}_2(-\theta)\mathbf{C}_1(-\varphi)$	-	-	-

3.4. TRANSLATIONAL DYNAMICS

In this section, only translational dynamics will be covered, as the simulator will be developed as a 3-DoF environment. In fact, the inclusion of rotational dynamics in the simulation environment would require either the implementation of a control system, which is assumed as perfect in this work, or the development of a 6-DoF guidance logic, as opposed to the CoM guidance used. The material presented in this section is taken from Mooij [1994].

3.4.1. STATE VARIABLES

Position and velocity are commonly expressed for a vehicle flying in a planetary atmosphere in either spherical or Cartesian coordinates. The latter are chosen as the state representation method for

this work. The two different formulations are introduced in the following sections, and the reasons for the choice of a Cartesian representation over a spherical one will be given.

CARTESIAN COORDINATES

In Cartesian coordinates, position is expressed by the x, y and z variables, and velocity by the $\dot{x}$, $\dot{y}$ and $\dot{z}$ variables. The state vector is then most commonly expressed as

$$\mathbf{x} = \begin{pmatrix} x & y & z & \dot{x} & \dot{y} & \dot{z} \end{pmatrix}^T$$

SPHERICAL COORDINATES

In spherical coordinates, position is represented by the distance from the origin r, longitude τ and latitude δ. Additionally, the groundspeed V_G, the flight path angle γ_G and heading angle χ_G define the velocity (in case $\mathcal{R}$ is the chosen reference). One could also express the velocity in a Cartesian form, where the components are the radial and two tangential projections of the velocity vector: v_r, v_τ and v_δ.

CONSIDERATIONS

Cartesian coordinates will be chosen over spherical ones for different reasons. First, surface-relative Cartesian coordinates make for an easy interpretation of quantities, such as altitude, downrange and crossrange (and the corresponding velocity and acceleration components), therefore easing the analysis process. Besides, the commands computed by the guidance algorithm, as well as the landing problem itself, are defined in terms of surface-relative $\mathcal{G}$-coordinates. Therefore, it would not make sense, from a computational point of view, to implement the EoM in spherical coordinates, as it would require additional coordinate transformations within the simulation environment. On the other hand, formulating the guidance in spherical coordinates for the problem at hand would be counter-intuitive and unhandy for the designer, as component variations would be small and harder to interpret. Finally, wind and gravitational data are obtained in Cartesian coordinates, providing a further reason to prefer this representation.

3.4.2. EXTERNAL FORCES

The external forces acting on a S/C in a planetary atmosphere are usually of three kinds:

- aerodynamic forces;
- gravitational forces;
- propulsion forces.

AERODYNAMIC FORCES

This kind of forces depends on the airspeed-based variables V_A (velocity), α_A (angle of attack) and β_A (angle of sideslip). They can be expressed as

$$\mathbf{F}_{A,\mathcal{A}\mathcal{A}} = \begin{pmatrix} -D \\ -S \\ -L \end{pmatrix} = \begin{pmatrix} -C_D \bar{q} S_{ref} \\ -C_S \bar{q} S_{ref} \\ -C_L \bar{q} S_{ref} \end{pmatrix} \tag{3.18}$$

The three components D, S and L are the aerodynamic drag, side and lift forces, respectively, while C_D, C_S and C_L are the corresponding coefficients, which are function of the Mach number M and of the aerodynamic angles α_A and β_A. Given the assumptions made on the rotationally symmetric shape of the lander, the side force component S will always be equal to zero. Also, it does not make sense to use the sideslip angle either, as the total angle of attack $\alpha_{A,tot}$ is sufficient to

unambiguously define the $\mathcal{AG}$- and $\mathcal{AA}$-frames. S_{ref} is the vehicle's aerodynamic reference surface area and q_{dyn} is the dynamic pressure, which is defined as

$$\bar{q} = \frac{1}{2}\rho V_A^2$$

where ρ is the atmospheric density and V_A is the airspeed.

The procedure to compute the aerodynamic force in the airspeed-based aerodynamic frame in presence of winds will be introduced in Section 3.4.4. Once these have been computed, for Cartesian EoM in surface-relative coordinates, Eq. (3.18) needs to be reformulated in the $\mathcal{G}$-frame. Thus:

$$\mathbf{F}_{A,\mathcal{G}} = \mathbf{C}_{\mathcal{G}}^{\mathcal{AA}}\mathbf{F}_{A,\mathcal{AA}} \tag{3.19}$$

GRAVITATIONAL FORCES

The gravitational force is expressed in the $\mathcal{R}$-frame as

$$\mathbf{F}_{G,\mathcal{R}} = m\mathbf{g}_{\mathcal{R}} \tag{3.20}$$

where m is the vehicle's mass and $\mathbf{g}$ is the gravitational acceleration, which is expressed in Cartesian coordinates as:

$$\mathbf{g}_{\mathcal{R}} = \begin{pmatrix} g_x \\ g_y \\ g_z \end{pmatrix}$$

PROPULSION FORCES

By definition, the total thrust force T is expressed in the $\mathcal{P}$-frame as follows:

$$\mathbf{F}_{T,\mathcal{P}} = \begin{pmatrix} T \\ 0 \\ 0 \end{pmatrix}$$

The following quantities are assumed to be known:

- $\mathbf{r}_{T,i}$: location of the i-th thruster;
- T_i: magnitude of the i-th engine thrust force;
- $\varepsilon_{T,i}$: elevation angle of the i-th thrust force (with regards to a *local* $\mathcal{BP}$-frame);
- $\psi_{T,i}$: azimuth angle of the i-th thrust force (with regards to a *local* $\mathcal{BP}$-frame).

It is thus possible to compute the total thrust force and the moment it produces on the vehicle. In fact, the thrust force will, in general, not be applied at the CoM, which induces a torque on the system. However, for this problem, the nominal thrust vector is assumed to be aligned with the $\mathcal{B}$-frame z-axis, and opposite in direction. Also, only one engine thruster is assumed. As such, the variables relevant to the definition of this force shrink down to T, ε_T and ψ_T.

The usual formulation of the DCM from the $\mathcal{P}$- to the $\mathcal{B}$-frame has been modified to account for the specific definition of the $\mathcal{B}$-frame for the chosen vehicle configuration. The conversion is then given by

$$\mathbf{C}_{\mathcal{B}}^{\mathcal{P}} = \mathbf{C}_2(-\pi/2)\mathbf{C}_1(-\psi_T)\mathbf{C}_2(-\varepsilon_T) = \begin{bmatrix} -\cos\psi_T\sin\varepsilon_T & \sin\psi_T & \cos\psi_T\cos\varepsilon_T \\ \sin\psi_T\sin\varepsilon_T & \cos\psi_T & -\sin\psi_T\cos\varepsilon_T \\ -\cos\varepsilon_T & 0 & -\sin\varepsilon_T \end{bmatrix}$$

So, the propulsion forces are expressed in the body frame $\mathcal{B}$ as

$$\mathbf{F}_{T,\mathcal{B}} = \mathbf{C}_{\mathcal{B}}^{\mathcal{P}}\mathbf{F}_{T,\mathcal{P}} = \begin{pmatrix} -T\cos\psi_T\sin\varepsilon_T \\ T\sin\psi_T\sin\varepsilon_T \\ -T\cos\varepsilon_T \end{pmatrix} \tag{3.21}$$

3.4.3. TRANSLATIONAL EOM IN A ROTATING FRAME

For the problem at hand, using a rotating reference frame brings about a number of advantages over using an inertial one. First, gravitational acceleration and wind vector are defined in planet-fixed coordinates. As such no transformation, or only a time-constant one, is required to express these quantities in a rotating frame. Besides, the thrust force is expressed in the $\mathcal{B}$-frame, whose attitude is expressed with respect to the $\mathcal{V}$-frame, which is also planet-fixed. As such, the conversion of this force into planet-fixed coordinates is faster. Finally, as the landing problem is defined in terms of variables such as downrange and altitude, which are defined with respect to the LS, a reference frame which is surface-fixed is highly desirable.

Treating the vehicle as a point mass, Newton's second law of motion in inertial space can be adjusted so as to be valid in the rotating frame $\mathcal{R}$:

$$\boldsymbol{F}_{\mathcal{R}} = m\frac{d^2\boldsymbol{r}_{cm}}{dt^2} + 2m\boldsymbol{\omega}_{\mathcal{R}} \times \frac{d\boldsymbol{r}_{cm}}{dt} + m\boldsymbol{\omega}_{\mathcal{R}} \times (\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{r}_{cm}) \tag{3.22}$$

where $\boldsymbol{\omega}_{\mathcal{R}} = \begin{bmatrix} 0 & 0 & \omega_{cb} \end{bmatrix}^T$ is the angular velocity vector of the rotating frame $\mathcal{R}$ and $\boldsymbol{F}_{\mathcal{R}}$ is the resultant external force applied to the vehicle (which includes the propulsive force), as expressed in the rotating frame. Inverting this equation, one can write:

$$m\frac{d^2\boldsymbol{r}_{cm}}{dt^2} = m\frac{d\boldsymbol{V}_{\mathcal{R}}}{dt} = \boldsymbol{F}_{\mathcal{R}} - 2m\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{V}_{\mathcal{R}} - m\boldsymbol{\omega}_{\mathcal{R}} \times (\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{r}_{cm}) \tag{3.23}$$

where:

- $2\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{V}_{\mathcal{R}}$ is the Coriolis acceleration to the vehicle's motion in the rotating frame;
- $\boldsymbol{\omega}_{\mathcal{R}} \times (\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{r}_{cm})$ is the apparent acceleration due to the spin rate of the rotating frame;
- the left-hand side represents the apparent acceleration of the vehicle in the rotating frame.

The kinematic equation

$$\frac{d\boldsymbol{r}_{cm}}{dt} = \boldsymbol{V}_{\mathcal{R}} \tag{3.24}$$

completes the general EoM formulation.

FORMULATION IN CARTESIAN COORDINATES

Let us express the state (i.e., position and velocity, in this case) in Cartesian coordinates in the $\mathcal{R}$-frame:

$$\boldsymbol{r}_{cm}^{\mathcal{R}} = \begin{pmatrix} x_{\mathcal{R}} & y_{\mathcal{R}} & z_{\mathcal{R}} \end{pmatrix}^T \tag{3.25}$$

$$\boldsymbol{V}_{\mathcal{R}} = \begin{pmatrix} u_{\mathcal{R}} & v_{\mathcal{R}} & w_{\mathcal{R}} \end{pmatrix}^T \tag{3.26}$$

$$\tag{3.27}$$

Then, the apparent accelerations can be written as:

$$\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{V}_{\mathcal{R}} = \begin{pmatrix} -\omega_{cb} v_{\mathcal{R}} \\ \omega_{cb} u_{\mathcal{R}} \\ 0 \end{pmatrix} \tag{3.28}$$

$$\boldsymbol{\omega}_{\mathcal{R}} \times (\boldsymbol{\omega}_{\mathcal{R}} \times \boldsymbol{r}_{cm}^{\mathcal{R}}) = \begin{pmatrix} -\omega_{cb}^2 x_{\mathcal{R}} \\ -\omega_{cb}^2 y_{\mathcal{R}} \\ 0 \end{pmatrix} \tag{3.29}$$

Substituting into Eqs. (3.23) and (3.24) yields

$$\frac{d\boldsymbol{V}_{\mathcal{R}}}{dt} = \begin{pmatrix} \dot{u}_{\mathcal{R}} \\ \dot{v}_{\mathcal{R}} \\ \dot{w}_{\mathcal{R}} \end{pmatrix} = \begin{pmatrix} \omega_{cb}^2 x_{\mathcal{R}} + 2\omega_{cb} v_{\mathcal{R}} \\ \omega_{cb}^2 y_{\mathcal{R}} - 2\omega_{cb} u_{\mathcal{R}} \\ 0 \end{pmatrix} + \frac{1}{m}(\boldsymbol{F}_{A,\mathcal{R}} + \boldsymbol{F}_{G,\mathcal{R}} + \boldsymbol{F}_{T,\mathcal{R}}) \tag{3.30}$$

$$\frac{d\boldsymbol{r}_{cm}^{\mathcal{R}}}{dt} = \begin{pmatrix} u_{\mathcal{R}} \\ v_{\mathcal{R}} \\ w_{\mathcal{R}} \end{pmatrix} \tag{3.31}$$

The EoM formulation in the rotating frame differs from that in the inertial frame for the presence of additional terms to account for the apparent forces. However, these forces can be shown to be small with respect with the second term of the right-hand side of the equation. For Mars, $\omega_{cb} \sim 5\,\mathrm{s}^{-1}$, the equatorial radius $R_e = 3397\,\mathrm{km}$, which can be assumed for $x_{\mathcal{R}}$ and $y_{\mathcal{R}}$, and worst case velocities of $\sim 100\,\mathrm{m/s}$ for the a terminal landing scenario. This yields values in the order of $\sim 10^{-2}\,\mathrm{m/s^2}$, which are approximately one order of magnitude smaller than typical aerodynamic accelerations and two orders of magnitude smaller than gravitational accelerations. For these reasons, these components will be disregarded. Finally, given the flat-Mars assumption presented in Section 3.1, the EoM can be written in $\mathcal{G}$-coordinates, which can be obtained via a time-constant transformation from the $\mathcal{R}$-frame, as:

$$\frac{d\boldsymbol{V}_{\mathcal{G}}}{dt} = \begin{pmatrix} \dot{u}_{\mathcal{G}} \\ \dot{v}_{\mathcal{G}} \\ \dot{w}_{\mathcal{G}} \end{pmatrix} = \frac{1}{m}(\boldsymbol{F}_{A,\mathcal{G}} + \boldsymbol{F}_{G,\mathcal{G}} + \boldsymbol{F}_{T,\mathcal{G}}) \tag{3.32}$$

$$\frac{d\boldsymbol{r}_{cm}^{\mathcal{G}}}{dt} = \begin{pmatrix} u_{\mathcal{G}} \\ v_{\mathcal{G}} \\ w_{\mathcal{G}} \end{pmatrix} \tag{3.33}$$

This formulation of the EoM will be the one implemented in the simulator. The following section will now move on to present the procedure used to obtain the aerodynamic force in presence of wind.

3.4.4. EFFECT OF WIND

The wind vector describes the motion of the atmosphere with respect to the rotating reference frame $\mathcal{R}$ or, in this specific setup, the guidance frame $\mathcal{G}$. Wind data, as obtained from the Mars Climate Database (MCD) (Section 4.8), is already expressed in surface-relative coordinates, so that their expression in $\mathcal{G}$-coordinates is quite straightforward.

WIND EQUATIONS IN CARTESIAN COORDINATES

The velocity obtained by integrating the EoM is called groundspeed velocity, that is, the velocity of the vehicle w.r.t. to the surface. However, aerodynamic variables like the angle of attack α depend on airspeed-based parameters. Therefore, the effect of wind must be included before such variables can be obtained.

The procedure followed for the inclusion of wind in the EoM has been modified with respect to the one provided by Mooij [1994], to take advantage of the rotationally symmeteric vehicle configuration in absence of proper aerodynamic data. First, Mach number data cannot be easily found for the landing vehicles used during PD. Besides, it is not very influencial at this stage of the EDL, as velocities are already subsonic at the beginning of the PD. Also aerodynamic coefficients cannot be easily found for landing vehicles, and drag and lift coefficient profiles have been obtained by means of a cosine-effect simplification on the axial coefficient of the Beagle capsule, as explained in Section 4.9. Additionally, a procedure to convert $\alpha_{A,tot}$ into α_A and β_A could not be retrieved. For all these

reasons, the modified procedure, which does not require knowledge of β_A and C_S, is used. This consists of computing the total angle of attack $\alpha_{A,tot}$, constructing the $\mathcal{AA}$-frame in $\mathcal{BA}$-coordinates, and using it to convert the aerodyanmic force back into $\mathcal{BA}$-coordinates. The main steps are:

- initial position and velocity (in Cartesian coordinates) are assumed to be known; so is the initial attitude;
- the wind vector is given in Cartesian coordinates;
- compute the airspeed-based velocity $\boldsymbol{V}_A$;
- compute the airspeed-based total angle of attack $\alpha_{A,tot}$;
- compute drag and lift coefficients C_D and C_L;
- compute aerodynamic forces based on these variables;
- construct $\mathcal{AA}$-frame and convert forces into groundspeed-based vectors;
- propagate state via EoM;
- continue to next time-step.

The velocity of the vehicle w.r.t. the ground is given by the vectorial sum of the velocity of the vehicle relative to the atmosphere and the velocity of the atmosphere relative to the ground. That is:

$$\boldsymbol{V}_{B/G,\mathcal{G}} = \boldsymbol{V}_{B/A,\mathcal{G}} + \boldsymbol{V}_{A/G,\mathcal{G}} \tag{3.34}$$

where

- $\boldsymbol{V}_{B/G,\mathcal{G}}$ is the velocity of the vehicle w.r.t. the $\mathcal{G}$-frame, expressed in $\mathcal{G}$-coordinates;
- $\boldsymbol{V}_{B/A,\mathcal{G}}$ is the velocity of the vehicle w.r.t. the atmosphere, expressed in $\mathcal{G}$-coordinates;
- $\boldsymbol{V}_{A/G,\mathcal{G}}$ is the velocity of the atmosphere w.r.t. the $\mathcal{G}$-frame, expressed in $\mathcal{R}$-coordinates.

Inverting Eq. (3.34), one can write

$$\boldsymbol{V}_{B/A,\mathcal{G}} = \begin{pmatrix} u_A \\ v_A \\ w_A \end{pmatrix} = \boldsymbol{V}_{B/G,\mathcal{G}} - \boldsymbol{V}_{A/G,\mathcal{G}} \tag{3.35}$$

The airspeed-based velocity vector magnitude is then

$$V_A = \sqrt{u_A^2 + v_A^2 + w_A^2} \tag{3.36}$$

The aerodynamics angles have now to be computed, which define the attitude of the vehicle w.r.t. the airspeed-based velocity. The aerodynamic angles relevant to the purpose of computing the necessary aerodynamic forces are the angle of attack and the angle of sideslip. To obtain them, one has first to express the airspeed-based velocity in the $\mathcal{BA}$-frame, the auxiliary body frame define to make interfacing with the $\mathcal{AA}$-frame easier. This is done as follows:

$$\boldsymbol{V}_{B/A,\mathcal{BA}} = \begin{pmatrix} u_B \\ v_B \\ w_B \end{pmatrix} = \mathbf{C}_{\mathcal{B}}^{\mathcal{G}} \boldsymbol{V}_{B/A,\mathcal{G}} \tag{3.37}$$

One can then compute the total angle of attack as

$$\alpha_{A,tot} = \arccos\left(\frac{\boldsymbol{V}_{B/A,\mathcal{BA}} \cdot \hat{\mathbf{i}}_{\mathcal{BA}}}{\|\boldsymbol{V}_{B/A,\mathcal{BA}}\|}\right) \tag{3.38}$$

and express the $\mathcal{AA}$-axes in $\mathcal{BA}$-coordinates as

$$\hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} = \frac{\boldsymbol{V}_{B/A,\mathcal{BA}}}{\|\boldsymbol{V}_{B/A,\mathcal{BA}}\|} \tag{3.39}$$

$$\hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} = \hat{\mathbf{i}}_{\mathcal{BA}} \times \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} \tag{3.40}$$

$$\hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} = \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} \times \hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} \tag{3.41}$$

By means of these unit vectors, one can then construct the DCM from $\mathcal{AA}$ to $\mathcal{BA}$ as

$$\mathbf{C}_{\mathcal{BA}}^{\mathcal{AA}} = \begin{bmatrix} \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} & \hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} & \hat{\mathbf{z}}_{\mathcal{AA},\mathcal{BA}} \end{bmatrix} \tag{3.42}$$

which has been verified to produce $\mathcal{AA}$-frames consistent with the sign of the aerodynamic coefficients, as explained in Section 4.9.

Knowing the aerodynamic coefficients as functions of the angle of attack, as presented in Section 4.9, the aerodyanamic forces can be obtained in the $\mathcal{AA}$-frame as in Eq. (3.18), where the dynamic pressure is $\bar{q} = \frac{1}{2}\rho V_A^2$. These can then be converted back into $\mathcal{BA}$-coordinates first, and subsequently $\mathcal{G}$-coordinates to be integrated.

4

SIMULATOR DEVELOPMENT AND VERIFICATION

A 3-DoF simulation environment has been developed in Simulink for testing the guidance algorithm to be implemented. The top-level architecture of the simulator is shown in Figure 4.1, which will be explained here by following the data flow, and presenting the various models and blocks as they are encountered.

The simulation kernel represents the core of the model, where the external forces are computed and the EoM integrated. The implementation and verification of the integrator block will be presented in Section 4.2. All the parameters needed for the computation of the forces are also input to the kernel from the corresponding modules, namely the vehicle module and the environmental module.

The output of the simulator kernel is the propagated state, which is input to the sensor model block (Section 4.11). This simulates the uncertainties in the measurements taken by the on-board instrumentation. Together with the actuator model (Section 4.13), the sensor model represents the hardware interface between the environment and the GNC system. Given the elements included in the problem, the sensor measurements consist of both inertial acceleration measurements, taken by the IMU, and non-inertial measurements, taken by the TRN system.

Despite the guidance system being the GNC subsystem of interest for this work, blending these measurements in a filter is believed to provide a more realistic set of inputs to the guidance system. As such, a simple Linear Kalman Filter (LKF) is included between the sensors and the guidance system block, to act as the navigation filter. Further considerations on this choice will be given in Section 4.12. The implementation and verification of the navigation system block is treated in Section 4.12. The so-obtained state is the input to the guidance logic block, which computes the action to take. As part of the simulator development, the Apollo guidance benchmark algorithm has been implemented and verified (Section 4.5). In fact, this has been done immediately after the development of the simulation kernel, as it allows to close the simulation loop, and verify the correct integration of subsequent models within the simulator core. The attitude computation (Section 4.6) and camera footprint (Section 4.7) blocks also work closely with the guidance system block, as their output is function of the trajectory, and directly depend on the actions commanded by the guidance system. The LS redesignation model (Section 4.10) is also shown as part of the same module as the guidance system in Figure 4.1, although more specifically part of the HDA system. However, a full HDA model is beyond the scope of this work, and a simplified, random retargeting model is implemented in its stead, as this is sufficient to test the capability of the guidance system to withstand this kind of maneuvers. This concludes the models which are part of the GNC system, as the control system is assumed to be perfect.

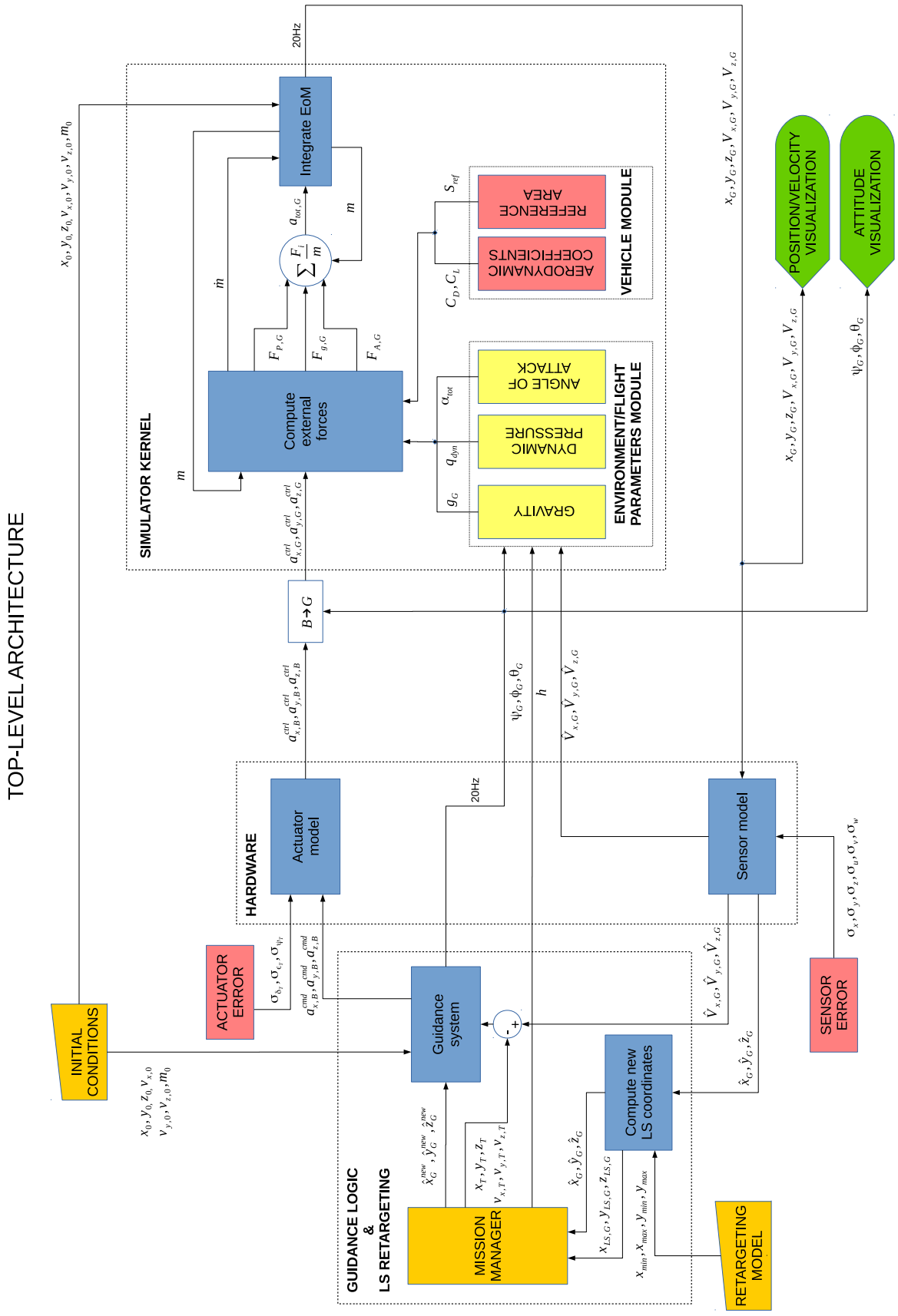


Figure 4.1: Simulator top-level architecture.

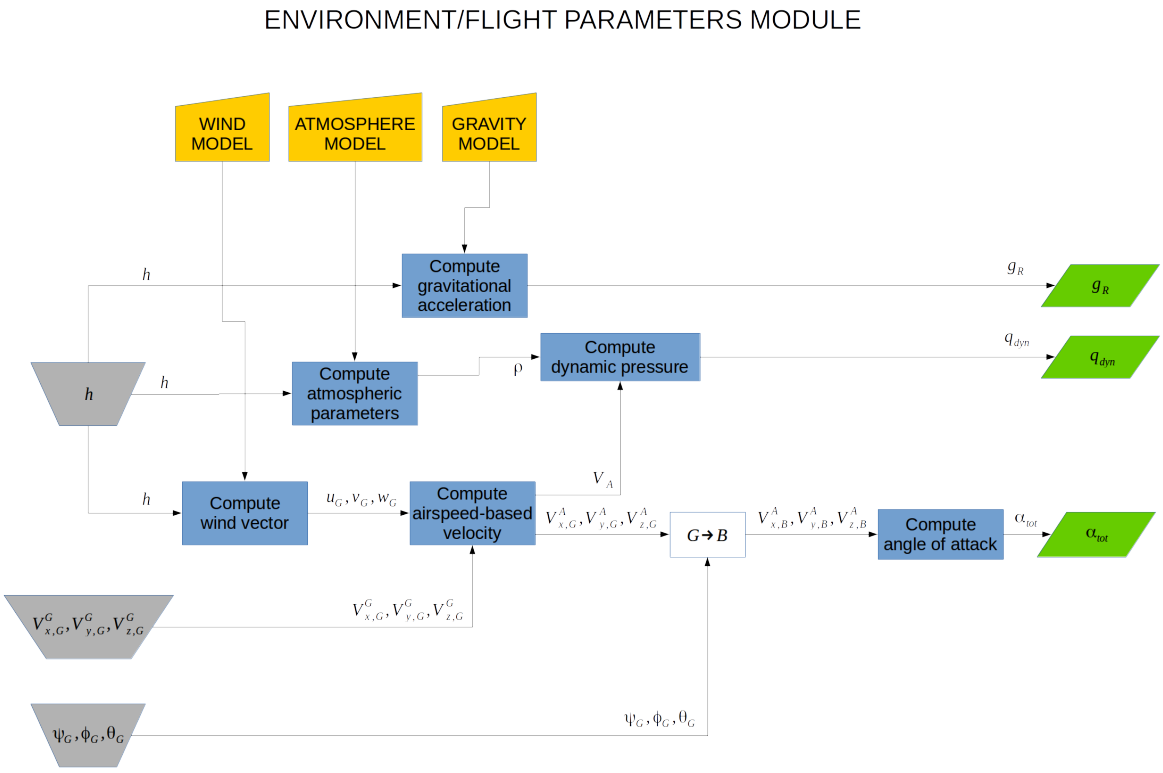


Figure 4.2: Architecture of the environment block.

The action commanded by the guidance system is then perturbed in the actuator model, whose output is used to recompute the external forces and the mass flow, and repeat the state propagation. Together with the above-mentioned models, there are a number of frame transformations blocks in the simulator. These will be presented and verified in Section 4.1.

The environment module, made up by the yellow blocks in Figure 4.1, is shown in greater detail in Figure 4.2. One can see that wind and atmosphere models (Section 4.8) are input to the block, together with the gravity model (Section 4.4). Current state and attitude are also input to this block. One can then use these to compute gravitational acceleration, dynamic pressure and aerodynamic angles.

The outputs of the environment block represent the input to the external forces block, shown in Figure 4.3. Vehicle parameters, commanded acceleration, current system mass and attitude are also input to this block. These allow to compute the propulsion force (Section 4.3) given the actuator model, the gravitational force and the aerodynamic force (Section 4.9). Once all these forces are known, the corresponding total acceleration can be obtained and integrated. Mass flow is also computed as part of the propulsion force block (Section 4.3), and integrated to propagate the mass.

In the following sections, the blocks and models mentioned above are presented and verified. Section 4.14, which concludes the chapter, discusses the verification of the integrated system.

4.1. FRAME TRANSFORMATIONS AND COORDINATE CONVERSIONS

These blocks will be verified by carrying out transformations whose outcome can be visually predicted given the input parameters. The result will then be compared against the expected one. Subsequently, the inverse transformation will be verified to reproduce the initial vector when applied to the result of the first transformation.

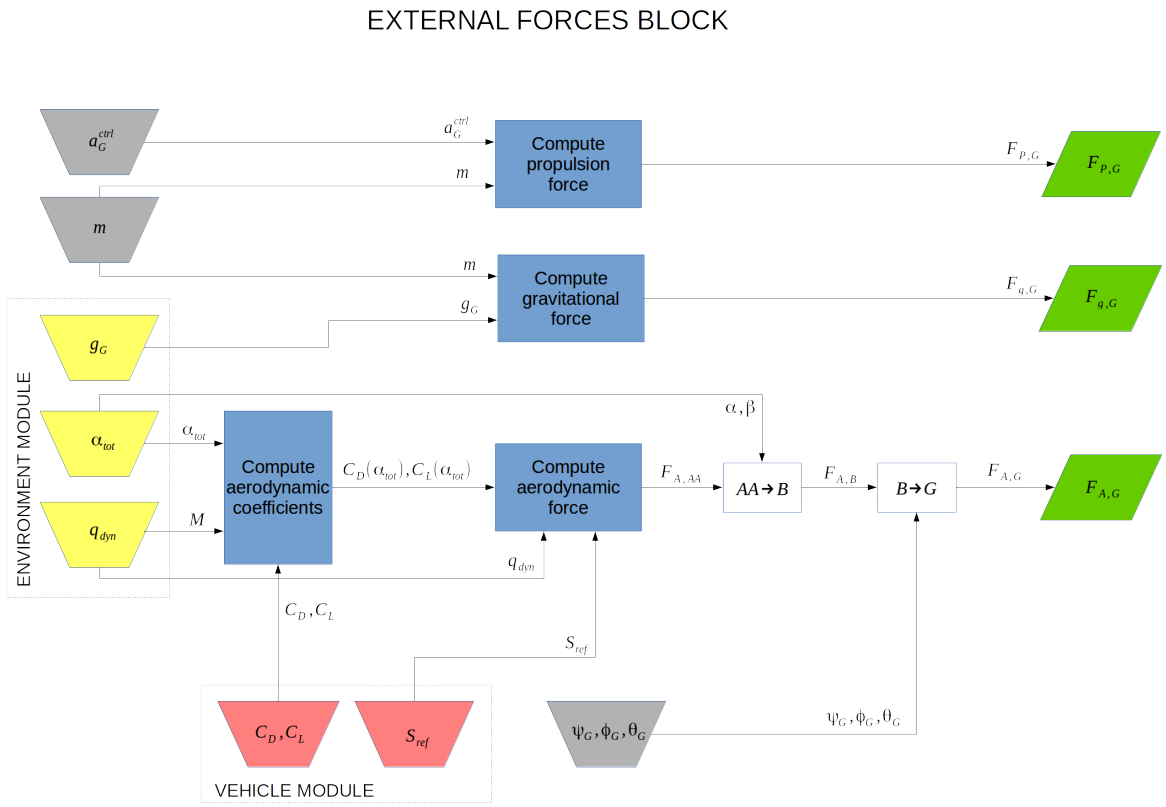


Figure 4.3: Architecture of the external forces block.

$\mathcal{G}$-FRAME TO $\mathcal{V}$-FRAME

The $\mathcal{G}$-frame (guidance frame), or landing site frame, is defined as originating at the landing site, and having the x-axis positive in the South direction, the z-axis positive upwards, and the y-axis completing the right-handed frame. Its orientation is thus fixed with respect to the vertical frame $\mathcal{V}$, which is defined with the x-axis pointing North and the z-axis pointing downwards. The y-axis follows. The (constant) transformation from $\mathcal{G}$ to $\mathcal{V}$ is then easily identified visually as a 180° rotation around the y-axis.

The rotation matrix from $\mathcal{G}$ to $\mathcal{V}$ is therefore written as

$$\mathbf{C}_{\mathcal{V}}^{\mathcal{G}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.1}$$

To verify this, the unit vector $\hat{\mathbf{a}}_{\mathcal{G}} = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}^T$ will be transformed into $\mathcal{V}$-components. The result is trivial and can be easily inferred visually: it should be $\hat{\mathbf{a}}_{\mathcal{V}} = \begin{bmatrix} -1/\sqrt{3} & 1/\sqrt{3} & -1/\sqrt{3} \end{bmatrix}^T$. After carrying out the computation, it is verified that the result matches the expected one. The transformation is therefore verified.

$\mathcal{V}$-FRAME TO $\mathcal{B}$-FRAME

The transformation from the vertical to the body frame is defined by the attitude angles (roll, pitch and yaw). The formulation of the DCM is available in Eq. (3.16) of Section **??**, which gives background information about Euler angles. It is nonetheless reported here for convenience.

$$\mathbf{C}_{\mathcal{B}}^{\mathcal{V}} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\varphi\sin\theta\cos\psi - \cos\varphi\sin\psi & \sin\varphi\sin\theta\sin\psi + \cos\varphi\cos\psi & \sin\varphi\cos\theta \\ \cos\varphi\sin\theta\cos\psi + \sin\varphi\sin\psi & \cos\varphi\sin\theta\sin\psi - \sin\varphi\cos\psi & \cos\varphi\cos\theta \end{bmatrix} \tag{4.2}$$

To verify it, a set of Euler angles whose outcome can be extrapolated visually is chosen, namely:

$$\psi = 90^{\circ} \tag{4.3}$$

$$\theta = 45^{\circ} \tag{4.4}$$

$$\varphi = 45^{\circ} \tag{4.5}$$

Then, the unit vector $\hat{\mathbf{a}}_{\mathcal{V}} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T$ is chosen, which, expressed in $\mathcal{B}$-components, should become $\hat{\mathbf{a}}_{\mathcal{B}} = \begin{bmatrix} \sqrt{2}/2 & 1/2 & 1/2 \end{bmatrix}^T$. Substituting the angle values into the DCM, and applying the rotation matrix to the vector expressed in $\mathcal{V}$-components, the correct vector in $\mathcal{B}$-components is obtained. This frame transformation is therefore verified.

$\mathcal{B}$-FRAME TO $\mathcal{AA}$-FRAME

The transformation from $\mathcal{B}$ to $\mathcal{AA}$ can be verified in a similar fashion. The rotation matrix results from two consecutive rotations (Table 3.1), and is built as follows:

$$\mathbf{C}_{\mathcal{AA}}^{\mathcal{B}} = \mathbf{C}_3(\beta_A)\mathbf{C}_2(-\alpha_A) \tag{4.6}$$

$$= \begin{bmatrix} \cos\beta_A & \sin\beta_A & 0 \\ -\sin\beta_A & \cos\beta_A & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\alpha_A & 0 & \sin\alpha_A \\ 0 & 1 & 0 \\ -\sin\alpha_A & 0 & \cos\alpha_A \end{bmatrix} \tag{4.7}$$

$$= \begin{bmatrix} \cos\beta_A\cos\alpha_A & \sin\beta_A & \cos\beta_A\sin\alpha_A \\ -\sin\beta_A\cos\alpha_A & \cos\beta_A & -\sin\beta_A\sin\alpha_A \\ -\sin\alpha_A & 0 & \cos\alpha_A \end{bmatrix} \tag{4.8}$$

Now, again, a set of aerodynamic angles which lead to an easily predictable outcome are chosen. For example, by choosing $\alpha_A = 30^{\circ}$ and $\beta_A = 45^{\circ}$, one can tell that the vector $\hat{\mathbf{a}}_{\mathcal{B}} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$ should result in $\hat{\mathbf{a}}_{\mathcal{AA}} = \begin{bmatrix} \sqrt{6}/4 & \sqrt{6}/4 & -1/2 \end{bmatrix}^T$. When applying the transformation matrix in Eq. (4.8) to the vector $\hat{\mathbf{a}}_{\mathcal{B}}$, the correct result is obtained. The transformation is thus verified.

$\mathcal{B}$-FRAME TO $\mathcal{P}$-FRAME

The propulsion frame $\mathcal{P}$ is defined as originating at the CoM, with the x-axis parallel to the $\mathcal{B}$-frame z-axis, but opposite in direction. The y-axis coincides with that of the body frame, while the y-axis completes the right-handed frame.

The rotation matrix from $\mathcal{G}$-frame to $\mathcal{V}$-frame can then be written by means of Eq. (3.3), which results in the following

$$\mathbf{C}_{\mathcal{P}}^{\mathcal{B}} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{4.9}$$

To verify this, the unit vector $\hat{\mathbf{a}}_{\mathcal{B}} = \begin{bmatrix} 1/\sqrt{3} & 1/\sqrt{3} & 1/\sqrt{3} \end{bmatrix}^T$ will be transformed into $\mathcal{P}$-components. The result is trivial and can be easily inferred visually: it should be $\hat{\mathbf{a}}_{\mathcal{P}} = \begin{bmatrix} -1/\sqrt{3} & 1/\sqrt{3} & -1/\sqrt{3} \end{bmatrix}^T$. After carrying out the computation, it is verified that the result matches the expected one. The transformation is therefore verified.

$\mathcal{B}$-FRAME TO $\mathcal{BA}$-FRAME

The additional frame $\mathcal{BA}$ was introduced to simplify the extrapolation of the angle of attack from the airspeed-based velocity, while maintaining unaltered the definition of the $\mathcal{AA}$-frame. The transformation from $\mathcal{B}$ to $\mathcal{BA}$ is given by

$$\mathbf{C}_{\mathcal{BA}}^{\mathcal{B}} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \tag{4.10}$$

The verification is done similarly to the previous transformations.

$\mathcal{B}$-FRAME TO $\mathcal{C}$-FRAME

The camera reference frame $\mathcal{C}$ is defined with y-axis corresponding to that of the $\mathcal{B}$-frame, z-axis in the LoS-direction, and x-axis completing the right-hand triplet. The rotation from $\mathcal{B}$ to $\mathcal{C}$ can then be written as

$$\mathbf{C}_{\mathcal{C}}^{\mathcal{B}} = \mathbf{C}_2(-\theta_{EP} - \theta_{FOV}/2) \tag{4.11}$$

$$= \begin{bmatrix} \cos(\theta_{EP}+\theta_{FOV}/2) & 0 & \sin(\theta_{EP}+\theta_{FOV}/2) \\ 0 & 1 & 0 \\ -\sin(\theta_{EP}+\theta_{FOV}/2) & 0 & \cos(\theta_{EP}+\theta_{FOV}/2) \end{bmatrix} \tag{4.12}$$

where θ_{EP} is the half-angle of cone within which the engine plumes are confined and θ_{FOV} is the angular diameter of the camera FOV. This is a negative rotation about the y-axis of magnitude $\theta_{EP} + \theta_{FOV}/2$, which is the minimum angle to make sure the FOV is clear from the engine plumes. The verification is done similarly to the previous transformations.

4.2. INTEGRATOR

The integrator block is made up of the integration of the EoM and of the S/C mass equation. The EoM to be integrated are given in Eqs. (3.30) and (3.31), and are reported here for convenience.

$$\frac{d\boldsymbol{v}}{dt} = \begin{pmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{pmatrix} = \frac{1}{m}(\boldsymbol{F}_A + \boldsymbol{F}_G + \boldsymbol{F}_T) \tag{4.13}$$

$$\frac{d\boldsymbol{r}}{dt} = \begin{pmatrix} u \\ v \\ w \end{pmatrix} \tag{4.14}$$

For the sake of verifying the integration of these equations, a scenario is set up on the xz-plane with only a constant force acting on the vehicle, which, assuming constant mass, results in a constant external acceleration $\mathbf{g}$. The resulting problem can be easily solved by hand. Analytical and numerical solutions can then be compared.

The initial conditions are chosen as follows

$$x_0 = 0\,\text{m} \tag{4.15}$$

$$z_0 = 0\,\text{m} \tag{4.16}$$

$$\dot{x}_0 = 70\,\text{m/s} \tag{4.17}$$

$$\dot{z}_0 = 70\,\text{m/s} \tag{4.18}$$

and the external acceleration acting on the vehicle is

$$g_x = 0\,\text{m} \tag{4.19}$$

$$g_z = -3.7114\,\text{m/s}^2 \tag{4.20}$$

which yields the position and velocity plots shown in Figure 4.4. As can be seen, the results correctly represent the solution of the selected scenario, and the analytical and numerical solutions match well.

Similarly, integrating the mass equation

$$\frac{dm}{dt} = \dot{m} \tag{4.21}$$

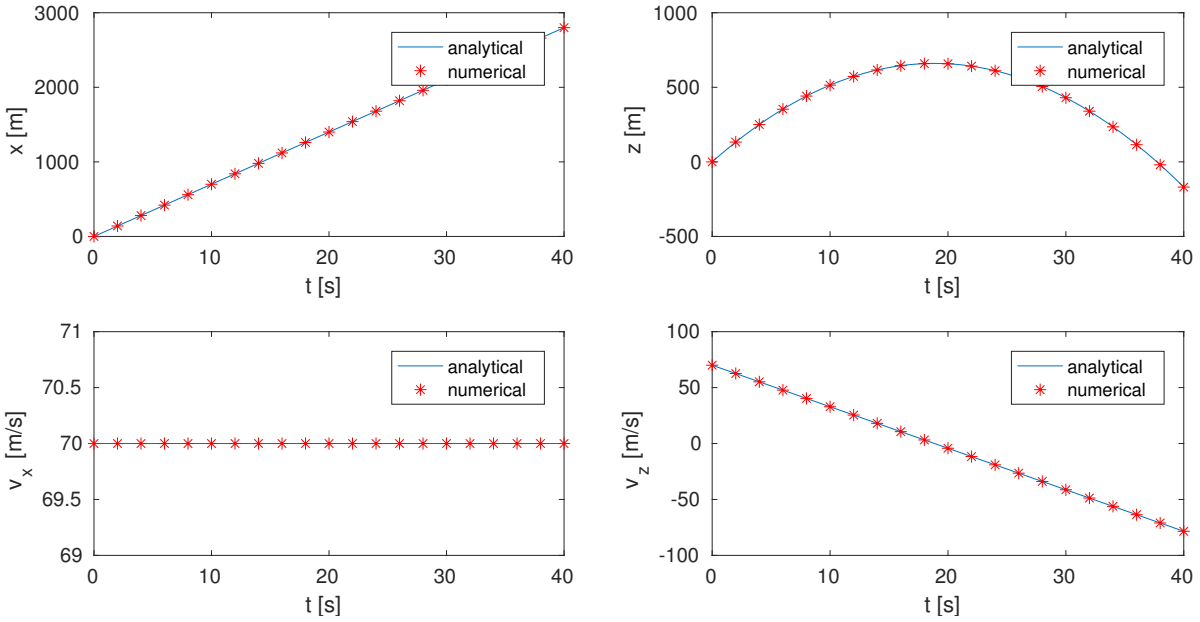


Figure 4.4: Integration of the EoM.

with $m_0 = 1900\,\text{kg}$ and $\dot{m} = -5\,\text{kg/s}$ for a time-span of 40 s. The result has been checked and, similarly to the case of position and velocity, one finds that, by comparing the numerical result to the analytical solution, the match between the two is sufficiently accurate.

The integrator used is a 4th order Runge-Kutta, with time step $\Delta t = 0.1\,\text{s}$. The integrator time-step will be subsequently reduced to 0.02 s, as a consequence of the implications brought about by the implementation of the LKF, as explained in Section 6.3. This value will be used for most of the simulations run. It also allows for a higher guidance frequency, which is run at the highest possible frequency given the step-size (i.e., 20 Hz).

4.3. PROPULSION FORCE AND MASS FLOW

By multiplying the commanded acceleration vector magnitude by the current system mass, one gets the propulsive force vector $\mathbf{F}_P$. One can compute the mass flow that corresponds to the given acceleration as

$$\dot{m} = \frac{\|\mathbf{F}_P\|}{g_0 I_{sp}} \tag{4.22}$$

where reference gravity $g_0 = 9.8\,\text{m/s}^2$ and specific impulse $I_{sp} = 225\,\text{s}$ are known and constant quantities. The force vector $\mathbf{F}_P$ and the mass flow $\dot{m}$ are then fed into the integration block for the propagation of system state and mass, respectively.

The verification of this block is done by choosing several different input values for the acceleration, all of which have the same magnitude. As a consequence, the resulting mass flow and the magnitude of the force vector should remain the same. It is therefore enough to check the values once analytically (by reproducing the calculation in Eq. (4.22)) and verify their consistency when varying the inputs as explained.

For instance, by choosing $a_{cmd} = \begin{bmatrix} 0 & 0 & 3 \end{bmatrix}^T \text{m/s}^2$ and $m = 1900\,\text{kg}$, it is verified that $\|\mathbf{F}_P\| = 5700\,\text{N}$ and $\dot{m} = 2.585\,\text{kg/s}$, as expected. By then choosing different components for the commanded acceleration such that its magnitude remains constant, it is checked that the outputs remain constant.

The so-obtained commanded propulsion force is expressed in the $\mathcal{P}$-frame. It will then be perturbed according to the actuator model presented in Section 4.13, before being converted back into

$\mathcal{B}$-, and ultimately $\mathcal{G}$-coordinates by means of the appropriate transformations.

4.4. GRAVITY MODEL

The `gravitysphericalharmonic` Matlab function will be used for the computation of the gravitational acceleration. This block implements the Goddard Mars Model 2B (GMM-2B), which is accurate enough for the nature of the problem at hand. In fact, given the altitude range at which the simulation takes place, and the duration of the process, a first-order Newtonian model could serve a similar purpose. However, the GMM-2B is a more conservative choice, as its level of accuracy is larger than that of the Newtonian model. Besides, being a built-in Matlab function, this relaxes the requirements on its verification, which will be carried out for this block by comparison to the Newtonian model.

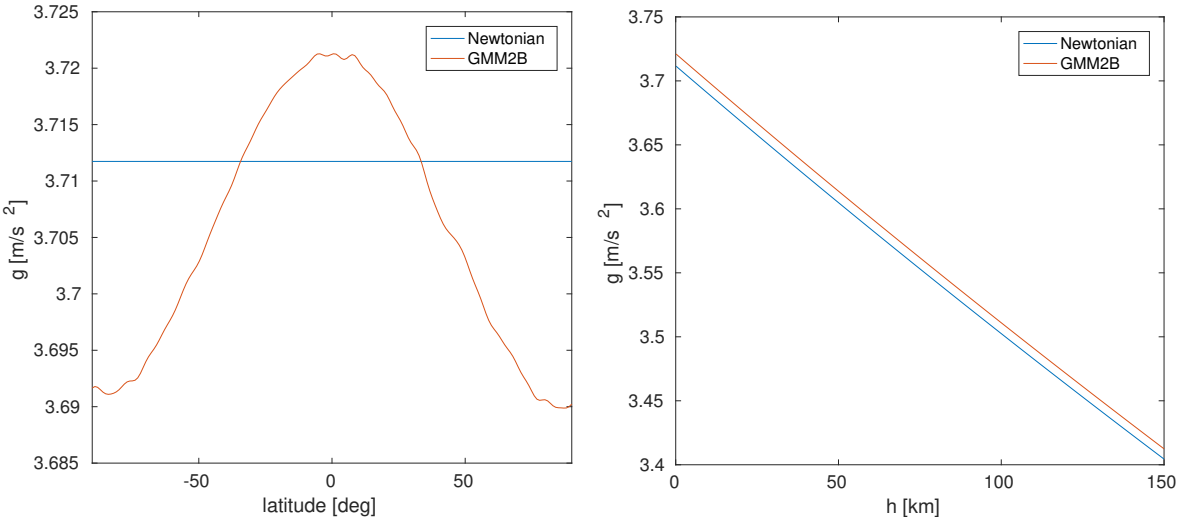
One can compute the Newtonian gravitational acceleration as

$$g = \frac{\mu}{(R_e + h)^2} \tag{4.23}$$

where $R_e = 3397\,\text{km}$ is Mars' equatorial radius and $\mu = 4.2832 \cdot 10^4\,\text{km}^3/\text{s}^2$ its gravitational parameter [Wertz et al., 2009]. The so-obtained Newtonian acceleration is compared to the norm of the GMM-2B model output, first by varying the latitude while keeping $h = 0\,\text{m}$ (altitude above equatorial radius) constant, and then by comparing the altitude profiles above a point located on the equator. The results are shown in Figure 4.5.

In the first figure one can see that the value obtained using only the Newtonian term is expectedly constant, while the values obtained through the GMM-2B are lower at the Equator and larger at the Poles, which is also in line with expectations. In fact, the latter is a spherical harmonics model that accounts for the oblateness of the planet ($J2$-term) and higher-order terms as well (up to order 60). By looking at Figure 4.5b, one can see that, except for an initial offset arising for the reason mentioned above, the vertical profile in the two cases is very similar.

These observations show that the `gravitysphericalharmonic` block was correctly implemented in the simulation loop.



(a) Gravitational acceleration comparison as a function of longitude.

(b) Gravitational acceleration comparison as a function of altitude.

Figure 4.5: Gravitational acceleration comparison between a Newtonian model and GMM-2B.

4.5. BENCHMARK GUIDANCE

The development of the benchmark guidance (i.e., Apollo guidance) is inserted here with the purpose of closing the simulation loop, and verifying its basic functionalities. The blocks to follow, although essential for the simulations to be carried out for this work, only add further functionalities, whose absence does not prevent the model from being run in closed loop.

The Apollo guidance, presented by Klumpp [1974], has been chosen as the benchmark algorithm given its ease of implementation, its proven flight history and readiness level, which makes it a natural reference for novel guidance algorithms, and its light computational requirements. It computes the control actions by solving a Two-Point Boundary-Value Problem (TPBVP), where the acceleration is modeled as a second-order polynomial, and velocity and position are obtained by integration.

$$\begin{aligned}\mathbf{a}(t) &= \mathbf{c}_0 + \mathbf{c}_1 t + \mathbf{c}_2 t^2 \\ \mathbf{v}(t) &= \mathbf{v}_0 + \mathbf{c}_0 t + \frac{1}{2}\mathbf{c}_1 t^2 + \frac{1}{3}\mathbf{c}_2 t^3 \\ \mathbf{r}(t) &= \mathbf{r}_0 + \mathbf{v}_0 t + \frac{1}{2}\mathbf{c}_0 t^2 + \frac{1}{6}\mathbf{c}_1 t^3 + \frac{1}{12}\mathbf{c}_2 t^4\end{aligned} \tag{4.24}$$

Given three final and three initial conditions, the solution to this linear system is uniquely determined as

$$\begin{pmatrix}\mathbf{c}_0 \\ \mathbf{c}_1 \\ \mathbf{c}_2\end{pmatrix} = \begin{bmatrix}\mathbf{I} & -\frac{6}{t_{go}}\mathbf{I} & \frac{12}{t_{go}^2}\mathbf{I} \\ -\frac{6}{t_{go}}\mathbf{I} & \frac{30}{t_{go}^2}\mathbf{I} & -\frac{48}{t_{go}^3}\mathbf{I} \\ \frac{6}{t_{go}^2}\mathbf{I} & -\frac{24}{t_{go}^3}\mathbf{I} & \frac{36}{t_{go}^4}\mathbf{I}\end{bmatrix}\begin{pmatrix}\mathbf{a}_t \\ \mathbf{v}_t - \mathbf{v}_0 \\ \mathbf{r}_t - \mathbf{r}_0 - \mathbf{v}_0 t_{go}\end{pmatrix} \tag{4.25}$$

where $\mathbf{c}_0$, $\mathbf{c}_1$ and $\mathbf{c}_2$ are 3×1 vectors, and $\mathbf{I}$ are 3×3 identity matrices. The arbitrary quantity t_{go} will be chosen according to the principle adopted for the actual Apollo missions, that is, such that the vertical component of the acceleration is a linear function of time [Klumpp, 1974]. This means setting the second derivative of the first of Eq. (4.24) equal to zero, i.e.:

$$c_{2_z} = 0$$

which, when the target vertical acceleration a_{t_z} is set to zero, yields a time-to-go equal to

$$t_{go} = 3\frac{r_{t_z} - r_{0_z}}{v_{0_z} + 2v_{t_z}} \tag{4.26}$$

The boundary conditions chosen for the verification of the benchmark guidance implementation are the nominal initial and final conditions chosen for the landing problem, and presented in Section 2.2.2.

$$\mathbf{r}_0 = \begin{pmatrix} -1760 & 0 & 1500 \end{pmatrix}^T \text{m} \tag{4.27}$$

$$\mathbf{r}_t = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \text{m} \tag{4.28}$$

$$\mathbf{v}_0 = \begin{pmatrix} 10 & 0 & -75 \end{pmatrix}^T \text{m/s} \tag{4.29}$$

$$\mathbf{v}_t = \begin{pmatrix} 0 & 0 & -0.4 \end{pmatrix}^T \text{m/s} \tag{4.30}$$

Given these quantities, it is possible to determine the time-to-go. According to Eq. (4.26), this is found to be $t_{go} = 59.3668\,\text{s}$, when rounded to the sixth significant digit. The value obtained numerically is verified to be consistent. The same can then be done with the constants $\mathbf{c}_0$, $\mathbf{c}_1$ and $\mathbf{c}_2$, which are also checked to be computed correctly in the Simulink implementation.

Once these parameters are computed, one can analytically compute the time-series of the acceleration, velocity, and position components, which are illustrated in Figure 4.6 (the y-components

are overlooked, as the simulation entirely takes place in the xz-plane). By looking at the plots, one can see that the initial and final conditions are complied with, and that the latter are reached after the correct time t_{go} for the analytical solution. The vertical acceleration plot in the bottom right shows the commanded acceleration. This is obtained by subtracting constant surface gravity ($g_{Mars} = 3.7114\,\mathrm{m/s^2}$) from the vertical component of the net acceleration, which is the actual solution to the TPBVP.

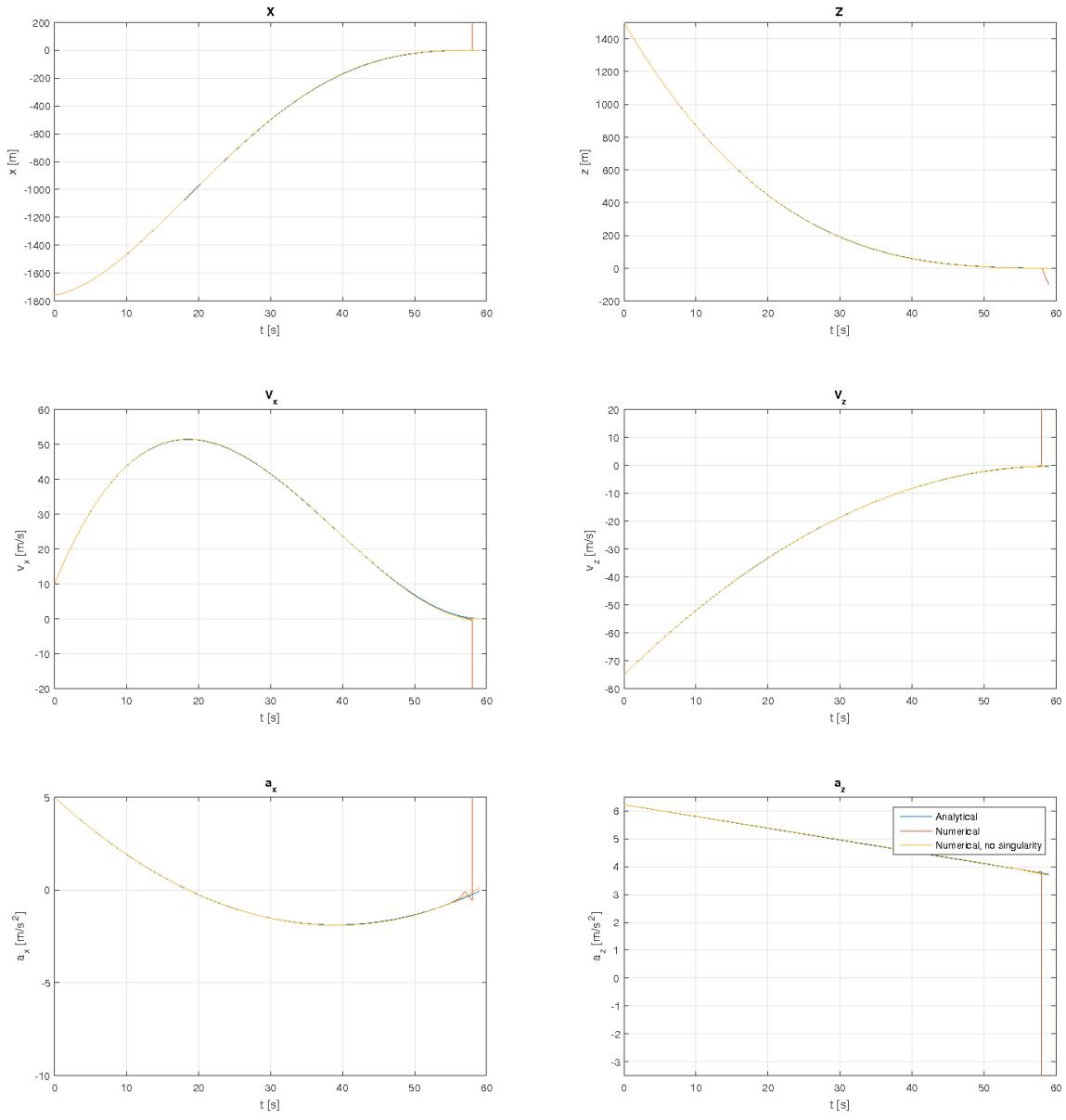


Figure 4.6: Time histories of position, velocity and acceleration components, generated analytically and numerically, with and without singularity avoidance.

When the same guidance is inserted in the simulation loop, comprising of the blocks developed so far (that is, gravity model and integrator), and run in a closed-loop fashion, a singularity occurs towards the end of the process, as can be seen in Figure 4.6. That happens because t_{go} is recomputed at every time step, and close to the final conditions it approaches 0. Therefore, the implemented Apollo guidance cannot be used for practical simulation purposes, and an adaptation is

necessary. A time threshold is introduced, after which the guidance switches to open-loop mode, thus avoiding the singularity. By looking at the figure, one can see that the effect of the singularity sets off approximately $4-5\,$s before TD. As such, the threshold is set to $t_{go}^{(thres)} = 5\,$s, which is also low enough to limit the errors resulting from the open-loop portion of the trajectory. In order to isolate the effect of this open-loop section, the gravitational acceleration is kept constant over altitude (and equal to surface gravity g_{Mars}) and the simulator step-size is reduced to 0.001 s. The residuals at TD for the nominal trajectory are reported in Table 4.1. One can see that they are very small, at about $10^{-12}\,$m, $10^{-8}\,$m/s and $10^{-10}\,$m/s^2 or smaller, for position, velocity and acceleration, respectively. The actual values have been interpolated for an altitude $h = 0\,$m. These effect of the open-loop trajectories is expected to be amplified as perturbations are added to the model, but to remain nonetheless small enough not to compromise landing safety.

Table 4.1: Numerical Apollo guidance residuals, with singularity avoidance switch, and RK4 integrator with $\Delta t = 0.001\,$s.

		x	z
$\Delta\mathbf{r}$	[m]	$-2.9394\cdot10^{-12}$	0
$\Delta\mathbf{v}$	[m/s]	$1.7115\cdot10^{-8}$	$-3.9194\cdot10^{-9}$
$\Delta\mathbf{a}_{net}$	[m/s^2]	$8.3863\cdot10^{-10}$	$-7.0166\cdot10^{-14}$

4.6. ATTITUDE COMPUTATION

Although rotational dynamics are not implemented within the simulator, it is still important to compute attitude information for the purpose of checking LS visibility *a-posteriori*.

The output of the guidance system, that is, the commanded acceleration vector in $\mathcal{G}$-coordinates is only enough to fix two of the three rotational DoFs. As an attitude profiling function is beyond the scope of this work, some assumptions will be made about the attitude of the vehicle in order to determine all the Euler angles. First, it is assumed that the landing site shall always lie on the plane created by the x- and z-axes of the $\mathcal{B}$-frame. In fact, that is the plane on which the central vector of the camera FOV lies as well. This allows to remove the rotational lability of the vehicle while maximizing LS visibility. Given the acceleration vector and the position relative to the LS in $\mathcal{G}$-coordinates ($\mathbf{a}_\mathcal{G}$ and $\mathbf{r}_{LS,\mathcal{G}}$, respectively), the components of the $\mathcal{B}$-frame axes in $\mathcal{G}$-frame coordinates are computed as follows

$$\hat{\mathbf{z}}_{\mathcal{B},\mathcal{G}} = -\frac{\mathbf{a}_\mathcal{G}}{\|\mathbf{a}_\mathcal{G}\|}; \tag{4.31}$$

$$\hat{\mathbf{y}}_{\mathcal{B},\mathcal{G}} = \frac{\hat{\mathbf{z}}_{\mathcal{B},\mathcal{G}} \times \mathbf{r}_{LS,\mathcal{G}}}{\|\hat{\mathbf{z}}_{\mathcal{B},\mathcal{G}} \times \mathbf{r}_{LS,\mathcal{G}}\|} \tag{4.32}$$

$$\hat{\mathbf{x}}_{\mathcal{B},\mathcal{G}} = \hat{\mathbf{y}}_{\mathcal{B},\mathcal{G}} \times \hat{\mathbf{z}}_{\mathcal{B},\mathcal{G}} \tag{4.33}$$

These three unit vectors are then converted into the $\mathcal{V}$-frame, with regards to which the attitude angles are defined. The resulting vectors $\hat{\mathbf{x}}_{\mathcal{B},\mathcal{V}}$, $\hat{\mathbf{y}}_{\mathcal{B},\mathcal{V}}$ and $\hat{\mathbf{z}}_{\mathcal{B},\mathcal{V}}$ represent the rows of the DCM describing the rotation from $\mathcal{V}$ to $\mathcal{B}$ (Eq. (3.16)). As such, one can then compute the Euler angles according to Eq. (3.17). The correct computation of the components of these vectors is verified by checking *a-posteriori* the time history of the dot product of the $\hat{\mathbf{z}}_\mathcal{B}$ axis with the (normalized) acceleration vector, and that of the $\hat{\mathbf{y}}_\mathcal{B}$ axis with the (normalized) vector of position relative to the LS (in $\mathcal{G}$-coordinates). These should be a constant -1 and a constant 0, respectively. The conversion from $\mathcal{G}$ to $\mathcal{V}$, and the extrapolation of the Euler angles from the DCM have already been verified.

The second assumption regarding the attitude computation block is about attitude rates, for which no maximum value is set. When a sudden change in the acceleration vector commanded by the guidance system occurs (e.g., following a LS retargeting), rotational velocity and acceleration

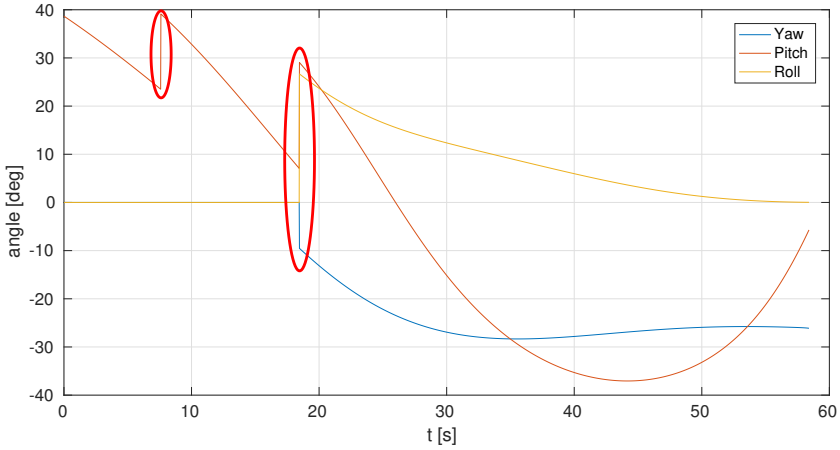


Figure 4.7: Time history of yaw, pitch and roll angles.

constraints, in a real case scenario, would lead to a gradual convergence to the desired attitude. This matter will be looked into later in this work (see Sections 6.2.4 and 7.2.4). For the moment, it is assumed that the vehicle can just slew into the target attitude from one time step to the next. However, this introduces an unrealistic situation, particularly after the retargetings, when extreme changes occur in commanded acceleration vector, which result in sudden changes in vehicle attitude, as can be seen in Fig. 4.7. One can see the disconinuities in the attitude angles profiles. The first of the two (leftmost) only affects the pitch angle, as the retargeting is in the forward direction, while the second retargeting, which has a non-zero crossrange component, has an effect on both yaw, pitch and roll.

This does significantly prevent a correct analysis of attitude kinematics only immediately *after* a retargeting. Also state estimation updates have an effect on this. This matter has been investigated, and the findings will be presented in Section 7.2.4. However, since LS visibility is an important parameter during the HDA system operations, one is mainly interested in studying it immediately *before* a retargeting. Of course, the portion of trajectory between the two retargetings would actually be affected by the attitude discontinuity at the first retargeting altitude, if rotational dynamic constraints were accounted for. However, pointing constraints will be included in this work only for altitudes above 1000 m (i.e., first retargeting altitude), as explained in Section 7.1.2. As such, this issue will not hinder LS visibility analysis in the desired altitude range, presented in Section 7.2.2.

4.7. SENSOR FOV

Once the vehicle attitude has been computed, it is possible to determine the footprint of the camera FOV on the surface. To this extent, the sensor configuration is first defined. In Figure 4.8a, one can see the LoS direction with respect to the structure of the lander and the $\mathcal{B}$-frame. In Figure 4.8b, the orange cone represents the engine plumes that the FOV must be clear of, while the blue polyhedron is the camera FOV, which is symmetrical with regards to the x-z plane of the $\mathcal{B}$-frame. Its width will be indicated as θ_{FOV}. The engine plumes cone angular radius is θ_{EP}.

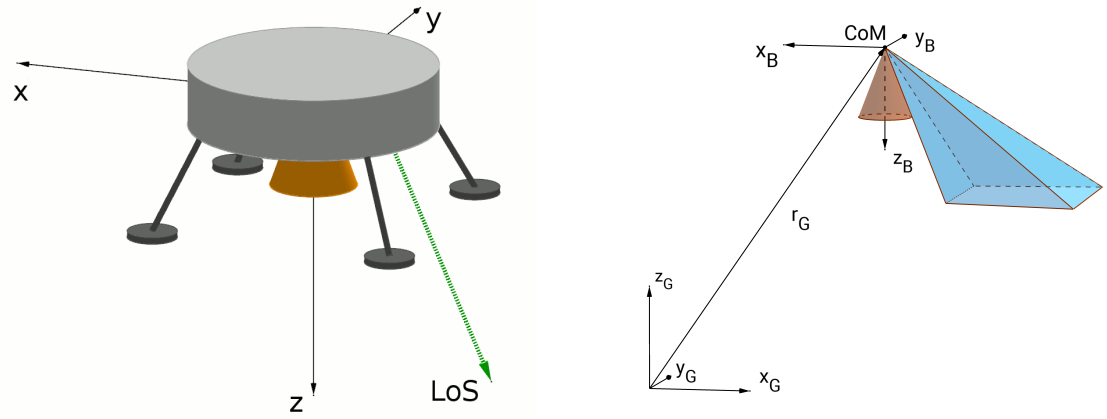
Four vectors starting at the origin of the $\mathcal{C}$-frame and parallel to each of the four FOV edges are defined as

$$\mathbf{v}_{1,\mathcal{C}} = \begin{pmatrix} \tan(\theta_{FOV}/2) & \tan(\theta_{FOV}/2) & 1 \end{pmatrix}^T \tag{4.34}$$

$$\mathbf{v}_{2,\mathcal{C}} = \begin{pmatrix} \tan(\theta_{FOV}/2) & -\tan(\theta_{FOV}/2) & 1 \end{pmatrix}^T \tag{4.35}$$

$$\mathbf{v}_{3,\mathcal{C}} = \begin{pmatrix} -\tan(\theta_{FOV}/2) & \tan(\theta_{FOV}/2) & 1 \end{pmatrix}^T \tag{4.36}$$

$$\mathbf{v}_{4,\mathcal{C}} = \begin{pmatrix} -\tan(\theta_{FOV}/2) & -\tan(\theta_{FOV}/2) & 1 \end{pmatrix}^T \tag{4.37}$$



(a) Geometry of the vehicle.

(b) Geometry of the camera FOV.

Figure 4.8: Vehicle and camera FOV geometry.

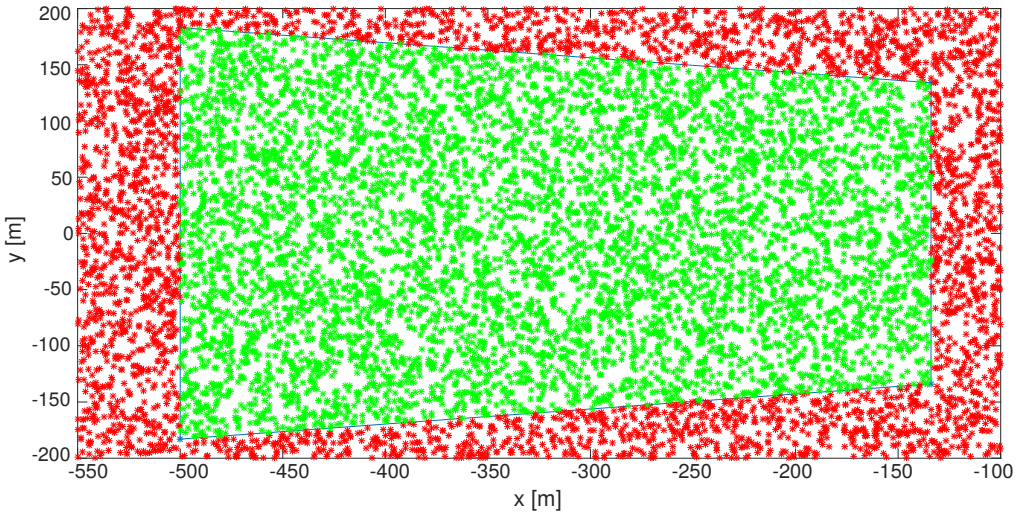


Figure 4.9: Demonstration of the inliers detection function.

These are then normalized, converted back into $\mathcal{B}$-coordinates, and then further into $\mathcal{G}$-coordinates. Given the position vector in $\mathcal{G}$-coordinates, these can be used to compute their projection points on the surface as

$$x_{proj,\mathcal{G}} = r_{x,\mathcal{G}} - \frac{v_{x,\mathcal{G}}}{v_{z,\mathcal{G}}} r_{z,\mathcal{G}} \tag{4.38}$$

$$y_{proj,\mathcal{G}} = r_{y,\mathcal{G}} - \frac{v_{y,\mathcal{G}}}{v_{z,\mathcal{G}}} r_{z,\mathcal{G}} \tag{4.39}$$

$$z_{proj,\mathcal{G}} = 0 \tag{4.40}$$

Once the coordinates of the projection points of the FOV edges are computed, one can check whether a point lies within the quadrilateral by means of the Matlab `inpolygon` function. One can see from Fig. 4.9, where the red points are the outliers, that the function correctly identifies inliers out of a randomly generated set of points.

The projection is verified by choosing position and attitude values for which the result can be computed analytically. The engine plumes semi-angle and the FOV width is chosen to be $\theta_{EP} = 15^o$ and $\theta_{FOV} = 30^o$. These are arbitrary values, chosen for a purely verification purpose. Given these

values, the position vector $\mathbf{r} = \begin{pmatrix} 0 & 0 & 1 \end{pmatrix}^T$ and the attitude angles $\begin{pmatrix} \psi & \theta & \phi \end{pmatrix}^T = \begin{pmatrix} 0^o & 30^o & 0^o \end{pmatrix}^T$, the expected projections of the LoS-vector and of the four edges are trivial, namely

$$\mathbf{x}_{LoS,proj} = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \tag{4.41}$$

$$\mathbf{x}_{1,proj} = \begin{pmatrix} -\tan(\theta_{FOV}/2) & \tan(\theta_{FOV}/2) & 0 \end{pmatrix}^T \tag{4.42}$$

$$\mathbf{x}_{2,proj} = \begin{pmatrix} -\tan(\theta_{FOV}/2) & -\tan(\theta_{FOV}/2) & 0 \end{pmatrix}^T \tag{4.43}$$

$$\mathbf{x}_{3,proj} = \begin{pmatrix} \tan(\theta_{FOV}/2) & \tan(\theta_{FOV}/2) & 0 \end{pmatrix}^T \tag{4.44}$$

$$\mathbf{x}_{4,proj} = \begin{pmatrix} \tan(\theta_{FOV}/2) & -\tan(\theta_{FOV}/2) & 0 \end{pmatrix}^T \tag{4.45}$$

These values are numerically verified. Furthermore, for a set of pitch angle values ranging from -10^o to 70^o the LoS projection can also be determined analytically. It is simply

$$\mathbf{x}_{LoS,proj} = \begin{pmatrix} -\tan(\theta - 30^o) & 0 & 0 \end{pmatrix}^T \tag{4.46}$$

Again, these are checked numerically. The output, including both sensor footprint and LoS projection, is plotted in Figure 4.10 for pitch angle steps of 20^o, assuming an altitude of 1000 m. One can see that the result is symmetric, as it should be, and that the sensor's footprint assumes an ever more trapezoidal shape, which is also expected. Besides, given that $2\tan(15^o) \sim 500$ m, one can see that the central footprint, which results from a downward pointing configuration (given the square shape), is approximately this size. As such, the projection function can be considered verified.

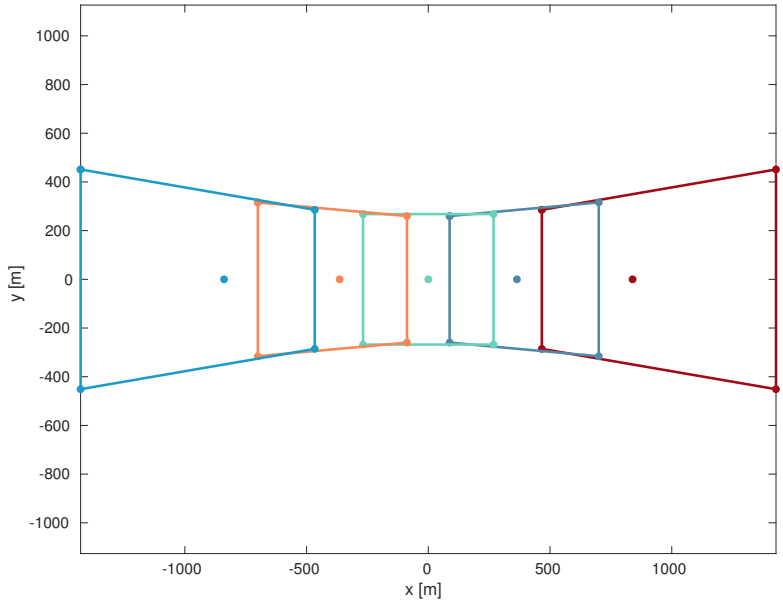


Figure 4.10: Sensor footprint and LoS projection for pitch angles $\theta = -10^o$, 10^o, 30^o, 50^o, 70^o.

4.8. ATMOSPHERE MODEL

The atmosphere data is taken from the MCD [Millour et al., 2015]. Mean and RMS values for density and wind will be made use of for two cases, a weak wind and a strong wind scenario. These data will serve as input to the wind turbulence model and dynamic pressure computation block.

The turbulence model is supposed to be applied to both density and wind, as these are interlaced. However, one can see from MCD data that density standard deviation values are about five orders of magnitude smaller than those of the wind. Besides, the dynamic pressure, which affects the magnitude of the aerodynamic force applied on the vehicle, scales linearly with density and

quadratically with airspeed-based velocity. The latter, given the low velocity during the final part of the descent, is even more affected by wind speed. As such, it can be safely assumed that density perturbations can be neglected for the purpose of this work. For the same reason, the vertical component of the wind vector, approximately three orders of magnitude smaller than the horizontal ones, is disregarded as well.

The model is a first-order, autoregressive model which can be presented as follows. The random variate $\mu(\mathbf{x}_k)$ expresses the deviation from the mean value at position $\mathbf{x}_k$, as normalized with respect to the standard deviation at that position. The variate is propagated as

$$\mu(\mathbf{x}_{k+1}) = r\mu(\mathbf{x}_k) + \sqrt{1-r^2}q(\mathbf{x}_k) \tag{4.47}$$

where q is a normally distributed random number with zero-mean and unit standard deviation, while r is the autocorrelation factor, which is assumed to be independent of horizontal displacement and time. It is thus computed according to an exponential function dependent on the altitude displacement only

$$r = \exp(-\delta z / L_z) \tag{4.48}$$

with $\delta z = |h_{k+1} - h_k|$ and L_z being the vertical scale parameter. The wind components are then computed as

$$w_{EW}(\mathbf{x}) = \bar{w}_{EW}(\mathbf{x}) + \mu_{EW}(\mathbf{x})\sigma_{w_{EW}}(\mathbf{x}) \tag{4.49}$$

$$w_{NS}(\mathbf{x}) = \bar{w}_{NS}(\mathbf{x}) + \mu_{NS}(\mathbf{x})\sigma_{w_{NS}}(\mathbf{x}) \tag{4.50}$$

for zonal and meridional components, respectively. As the former is defined to be positive towards the East, and the latter to be positive in the North direction, the wind vector in the $\mathcal{G}$-frame is written as

$$\mathbf{w}_{\mathcal{G}} = \begin{bmatrix} -w_{NS} \\ w_{EW} \\ 0 \end{bmatrix} \tag{4.51}$$

Figure 4.11 illustrates the result of this model. The top plot shows the mean density profile. One can see that the profile is somewhat inhomogeneous, as a consequence of the interpolation performed between the data points obtained from the MCD model. The bottom plots show the profiles of the mean and perturbed wind components, with 1-σ boundaries indicated in red. These plots are referred to the strong wind scenario, with a vertical scale parameter $Lz = 100\,\text{m}$. Finally, Figure 4.12 shows ten profiles obtained from different random sequences and randomized initial conditions for the random variate of Eq. (4.47).

4.9. AERODYNAMIC FORCE

Wind vector and atmospheric density are the output of the atmosphere model block. The wind vector is used to compute airspeed-based velocity, which is in turn fed, together with the density, into the dynamic pressure calculation block. These blocks are trivial, and are verified by repeating the calculations for a number of input values.

The airspeed-based velocity is then further converted into the $\mathcal{BA}$-frame by means of the already verified rotation matrices, so that it can be used to compute the total angle of attack $\alpha_{A,tot}$. Knowing the airspeed-based velocity vector in $\mathcal{BA}$-coordinates allows one to compute the components of the $\mathcal{AA}$-axes. This is done as follows

$$\hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} = \frac{\mathbf{v}_{A,\mathcal{BA}}}{\|\mathbf{v}_{A,\mathcal{BA}}\|}; \tag{4.52}$$

$$\hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} = \frac{\hat{\mathbf{x}}_{\mathcal{BA},\mathcal{BA}} \times \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}}}{\|\hat{\mathbf{x}}_{\mathcal{BA},\mathcal{BA}} \times \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}}\|} \tag{4.53}$$

$$\hat{\mathbf{z}}_{\mathcal{AA},\mathcal{BA}} = \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} \times \hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} \tag{4.54}$$

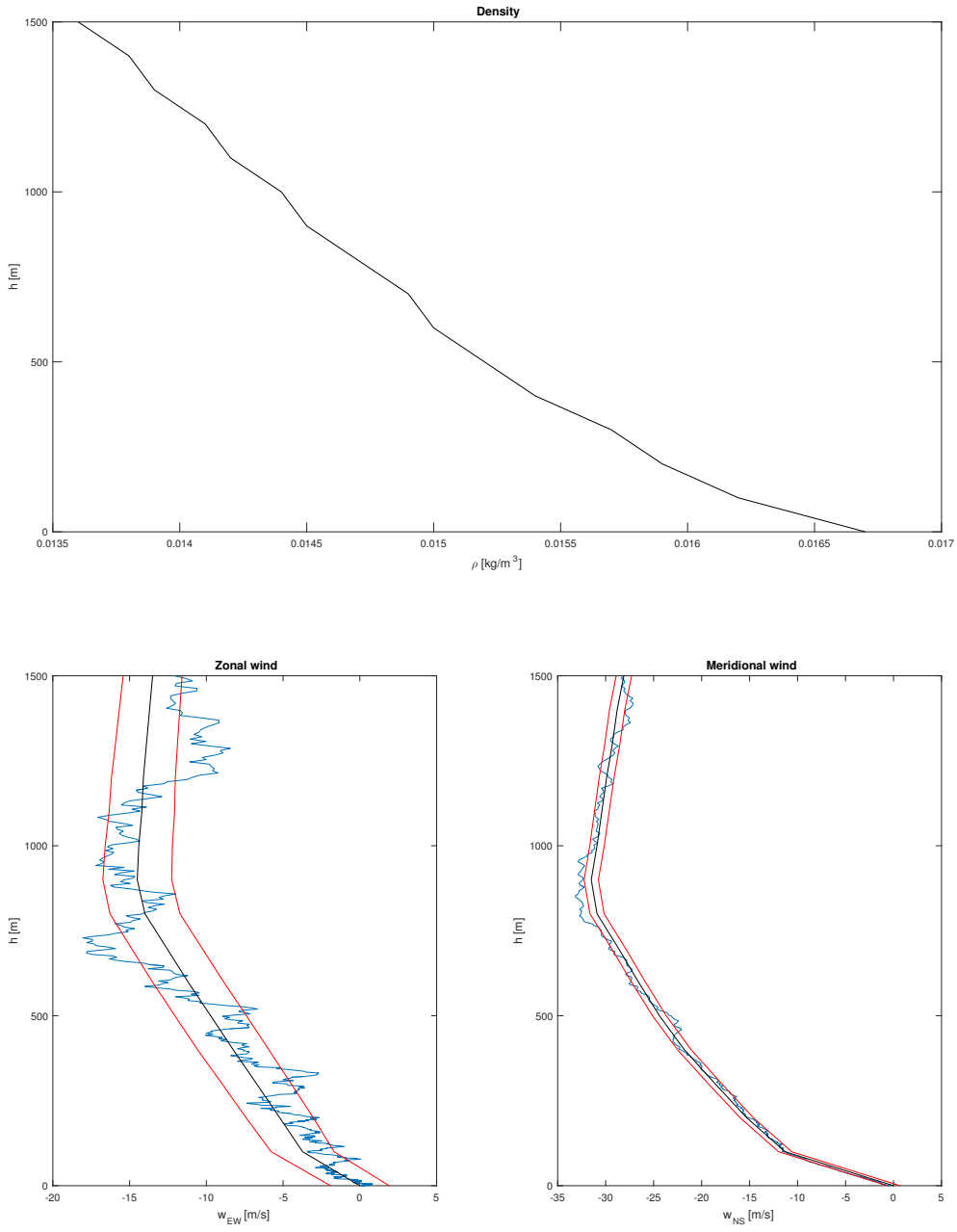


Figure 4.11: Density and wind profiles for the strong wind case, $L_z = 100\,\text{m}$.

where $\hat{\mathbf{x}}_{\mathcal{BA},\mathcal{BA}} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}^T$. Again, the rotation matrix from $\mathcal{BA}$ to $\mathcal{AA}$ is then written as

$$\mathbf{C}_{\mathcal{BA}}^{\mathcal{AA}} = \begin{bmatrix} \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}} & \hat{\mathbf{y}}_{\mathcal{AA},\mathcal{BA}} & \hat{\mathbf{z}}_{\mathcal{AA},\mathcal{BA}} \end{bmatrix} \tag{4.55}$$

On the other hand, the angle of attack is computed as

$$\alpha_{A,tot} = \arccos(\hat{\mathbf{x}}_{\mathcal{BA},\mathcal{BA}} \cdot \hat{\mathbf{x}}_{\mathcal{AA},\mathcal{BA}}) \tag{4.56}$$

The output of the Matlab `acos` function is always between 0 and π. The correct direction of the lift vector, given the sign of the lift coefficient shown in Figure 4.14, is thus ensured by the construction of the $\mathcal{AA}$-frame, in particular by the order of the cross product of Eq. (4.53), from which the orientation of the $\hat{\mathbf{z}}_{\mathcal{AA}}$ axis follows.

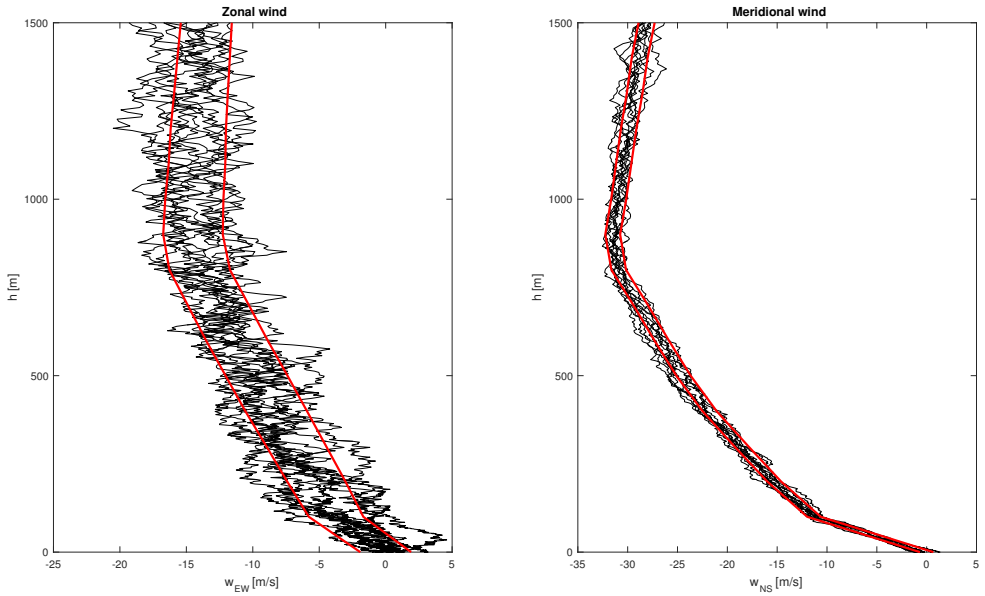


Figure 4.12: Wind profiles resulting from ten MC runs. The red lines indicate the $\pm 1\sigma$ boundaries.

The verification of this block is done by choosing a number of vectors in the $\mathcal{BA}$-frame, representing the velocity vector, and repeating the calculations in Eq. (4.52) to (4.54) and (4.56). Also, the unity of the determinant of the DCM in Eq. (4.55) is checked. Finally, for only a few cases the output is analyzed visually as well. One of these plots is shown in Fig. 4.13. From an inspection of the figure, one can check that:

- the velocity vector $\mathbf{v}_{AA}$ (black) is aligned with the x-axis of the $\mathcal{AA}$-frame (pink), and points in the same direction;
- the propulsion vector $\mathbf{F}_P$ (purple) is aligned with the z-axis of the $\mathcal{B}$-frame (orange), but points in the opposite direction;
- x- and z-axis of the $\mathcal{AA}$-frame, z-axis of the $\mathcal{B}$-frame, velocity vector $\mathbf{v}_{AA}$, and all the external forces vectors (propulsive, drag (green), lift (light blue), and, therefore, total aerodynamic(red)) lie on the same plane;
- the drag vector is aligned with the x-axis of the $\mathcal{AA}$-frame, but points in the opposite direction, and the lift vector is perpendicular to it;
- the lift vector points in the correct direction;
- the angle of attack is computed correctly (sanity check).

Once this has been done for a number of configurations, together with the numerical checks mentioned above, the block for the computation of $\mathcal{AA}$-frame and angle of attack can be considered verified.

AERODYNAMIC COEFFICIENTS

The aerodynamic coefficients for a lander (that is, the vehicle after heatshield jettison and backshell separation), necessary to complete the calculation of the aerodyanamic force, could not be found

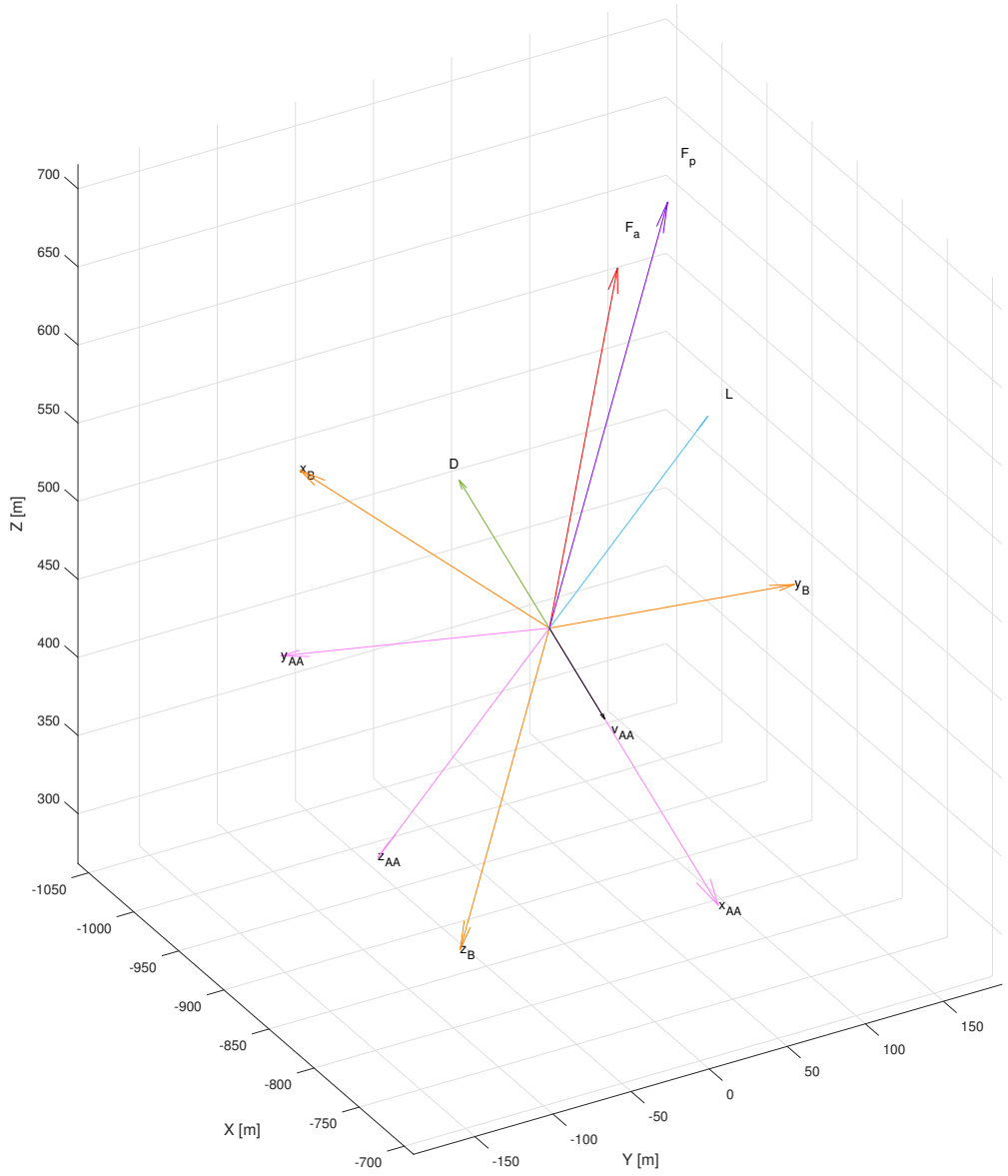


Figure 4.13: Aerodynamic frame and forces, together with body frame, and velocity and propulsive vectors.

for any Mars mission. Instead, an approximation is made starting from the nominal coefficient of the Beagle capsule $C_a = 1.425$. Given this parameter, he drag coefficient is modeled as:

$$C_d(\alpha) = \delta C_a + \frac{1}{2} C_a (1 + \cos(2\alpha)) \tag{4.57}$$

where δ is a percentage penalty for lander legs and other lander structures which are not present during the entry phase. This is additive, rather than multiplicative, so that the drag coefficient would not be equal to zero when $\alpha = \pi/2$. Subsequently, the lift coefficient is modeled as:

$$C_l(\alpha) = \frac{1}{2}(1 + 2\delta) C_a \sin(2\alpha) \tag{4.58}$$

where the multiplier of the penalty factor is chosen so that $C_d = C_l$ at $\alpha = \pi/4$.

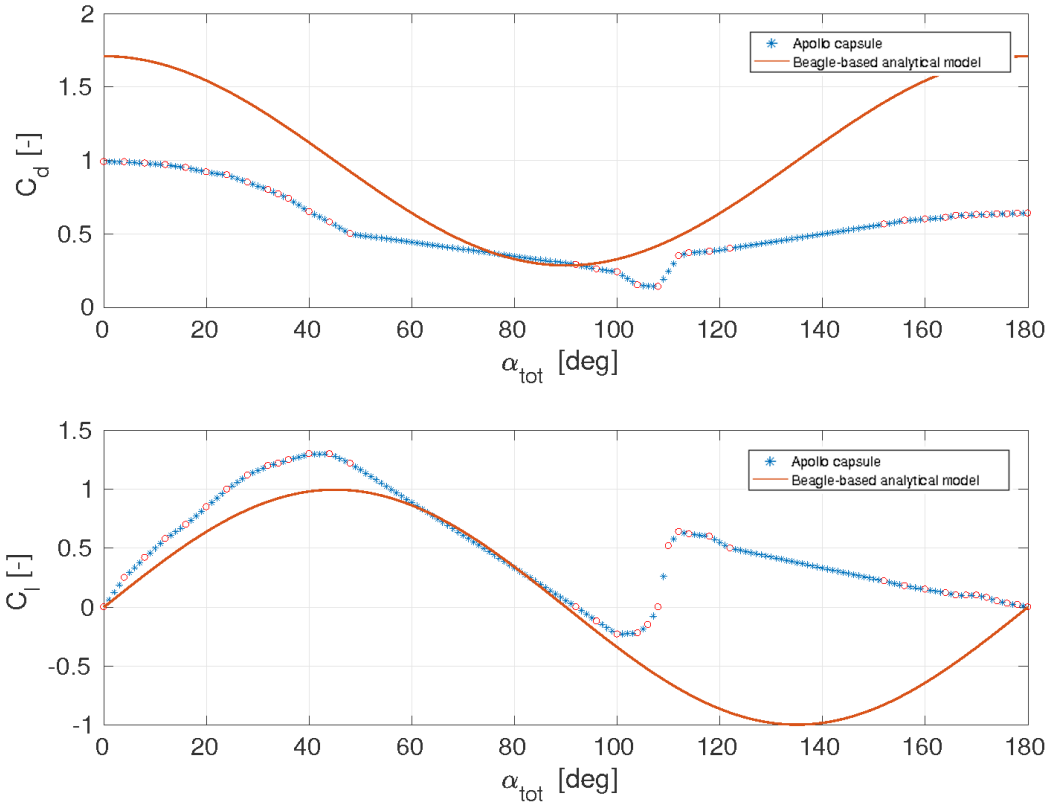


Figure 4.14: Aerodynamic coefficients as a function of the angle of attack.

The aerodyanamic coefficients plots are shown in Figure 4.14 for $\delta = 0.2$ as a function of the angle of attack α. The Figure also shows the comparison with the coefficients of a ballistic capsule, namely the Apollo capsule (values retrieved from Moseley et al. [1968]). One can see that for $\alpha < 90^o$ the lift coefficient is well in agreement. As a result of the higher nominal C_a of the Beagle capsule, large discrepancies in the drag coefficient are present for values of $\alpha = 0^o = 180^o$, but since the curves behave similarly approaching $\alpha = 90^o$, and the difference between them is approximately according to $C_d = 0.3 + K(C_d^{apollo} - 0.3)$, where K is a constant value, the model can be considered validated. A comparison for $\alpha > 90^o$ is less significant, as the lander is assumed to be more symmetrical with respect to the x-y plane than an entry capsule. Besides, values larger than 90^o are expected to be rare. In fact, at the beginning of the PDG, velocities are almost vertical. As such, the thrust vector should be in the downward direction to achieve a vehicle attitude resulting in $\alpha > 90^o$. On the other hand, later on in the descent, flight path angles can be smaller, with positive downrange velocities, but the vehicle will have started the pitch-up maneuver, thus also resulting in small angles of attack.

Once the aerodynamic coefficients are computed from Eqs. (4.57) and (4.58), the resulting aerodynamic force is verified by reproducing the calculations, given the input values for airspeed-based velocity, reference surface area, and the previously computed dynamic pressure. Input and output to the meta-block containing these smaller blocks are variables in $\mathcal{G}$-coordinates. Since Fig. 4.13 is relative to the $\mathcal{G}$-frame, this also verifies the transformations from $\mathcal{G}$-frame to $\mathcal{BA}$-frame, and vice-versa.

4.10. RETARGETING MODEL

Retargetings are modeled as a horizontal LS position shift at the retargeting altitude. One first retargeting is scheduled at 1000 m, while the second takes place at 500 m. The LS shift is randomly generated according to a uniform distribution within an area of 800 m radius, centered around the point $(x, y) = (350, 0)$m. The reason for this will be given in Section 7.1.1.

4.11. SENSOR MODELS

As already mentioned, a LKF was implemented as part of the navigation system. The filter, presented in the next section, blends the state estimate obtained by dead-reckoning of high-frequency (10 Hz) inertial data provided by the IMU, with low-frequency (1 Hz) non-inertial measurements coming from the TRN system. As rotational dynamics are disregarded for this work, only acceleration IMU data will be made use of. The accelerometer is fixed with respect to the $\mathcal{B}$-frame, and its measurements are modeled in such frame as

$$\mathbf{a}_{meas,\mathcal{B}} = \mathbf{a}_{real,\mathcal{B}} + \boldsymbol{\varepsilon} + \int \dot{\boldsymbol{\varepsilon}} \tag{4.59}$$

$$\dot{\boldsymbol{\varepsilon}} = \boldsymbol{\nu}_a \tag{4.60}$$

where $\boldsymbol{\varepsilon}$ is the three-dimensional accelerometer bias, while $\dot{\boldsymbol{\varepsilon}}$ is the corresponding bias drift. This is modeled as a zero-mean white Gaussian noise of standard deviation $\boldsymbol{\sigma}_{\nu_a} = \begin{pmatrix} 10 & 10 & 20 \end{pmatrix}^T \mu$g, in line with typical performances of space-rated accelerometer. Consistently, the constant accelerometer bias is randomized acrossing different simulations according to a zero-mean Gaussian white-noise with 1-σ values in $\mathcal{B}$-components equal to $\begin{pmatrix} 1 & 1 & 2 \end{pmatrix}^T$ mg. By means of a MC run, it is verified that the accelerometer bias is distributed as expected over the runs. The histograms plots of Figure 4.15 confirms this, whereas Figure 4.16 displays the nature of the random walk $\int \dot{\boldsymbol{\varepsilon}}$ (top plots) and of the underlying white-noise sequence $\boldsymbol{\nu}_a$ (bottom plots).

On top of accelerometer measurements, non-inertial measurements are simulated at 1 Hz as

$$\mathbf{r}_{meas} = \mathbf{r}_{real} + \boldsymbol{\nu}_r + \begin{pmatrix} \Delta x_{map-tie} \\ \Delta y_{map-tie} \\ 0 \end{pmatrix} \tag{4.61}$$

$$\mathbf{v}_{meas} = \mathbf{v}_{real} + \boldsymbol{\nu}_v \tag{4.62}$$

where $\Delta x_{map-tie}$ and $\Delta y_{map-tie}$ represent map-tie errors, which are generated according to a uniform distribution between $[-0.5, 0.5]$m at the beginning of each simulation, whereas the noises are

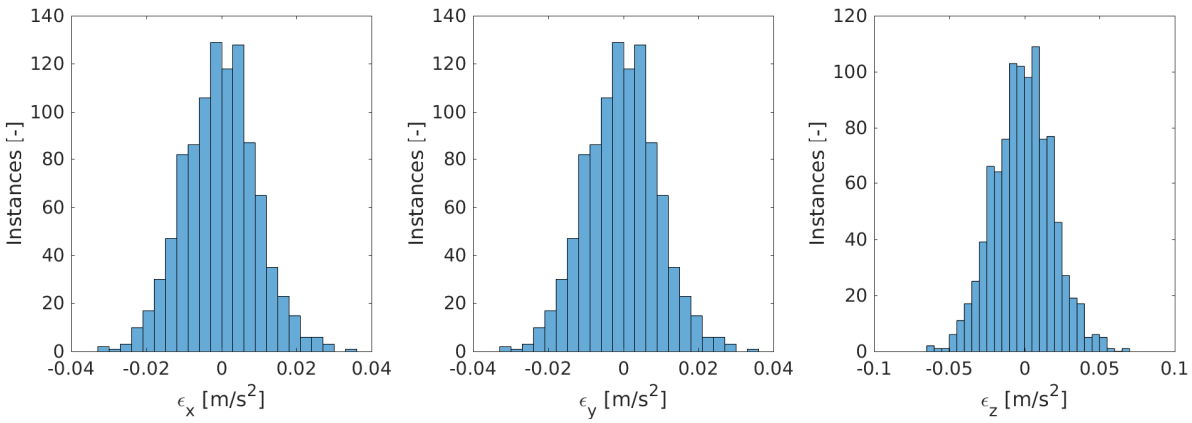


Figure 4.15: Histogram plots of the components of the accelerometer bias.

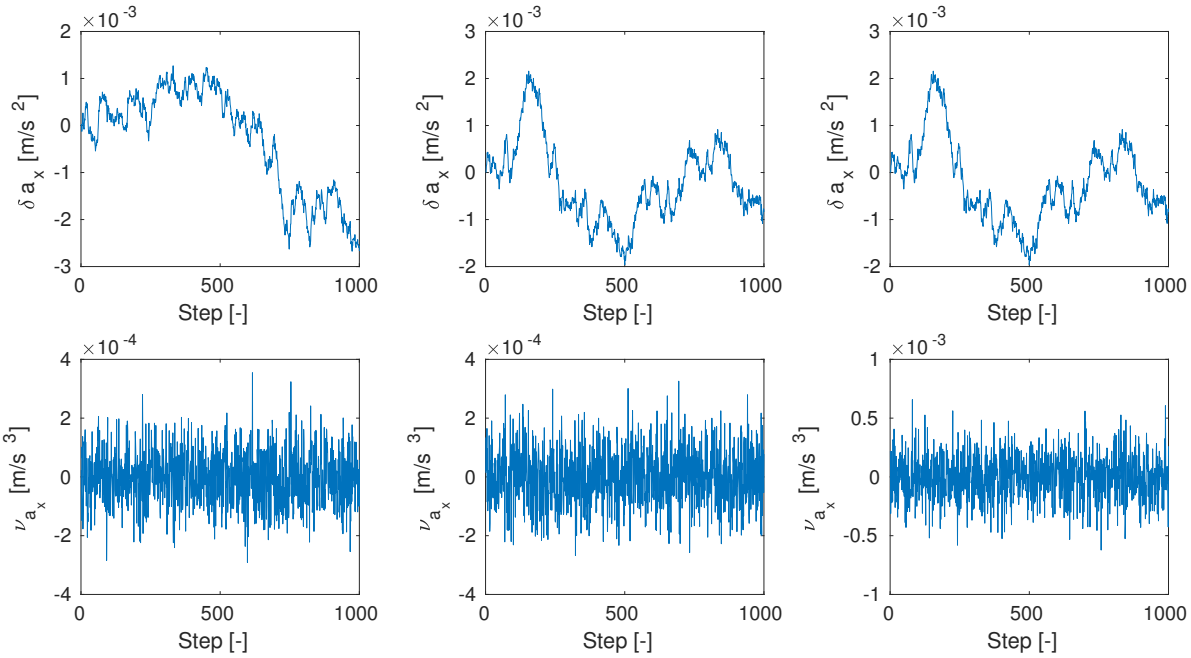


Figure 4.16: Accelerometer error random walk and underlying noise sequence.

modeled as

$$\boldsymbol{\nu}_r = \begin{pmatrix} U(-x_{TRN}, x_{TRN}) \\ U(-y_{TRN}, y_{TRN}) \\ N(0, \sigma_{z,TRN}) \end{pmatrix} \tag{4.63}$$

$$\boldsymbol{\nu}_v = \begin{pmatrix} U(-v_{x,TRN}, v_{x,TRN}) \\ U(-v_{y,TRN}, v_{y,TRN}) \\ N(0, \sigma_{v_{z,TRN}}) \end{pmatrix} \tag{4.64}$$

A uniformly distributed noise is thought to be more representative of the kind of output the image processing algorithm of the TRN system provides, while a normal distribution is more appropriate for the altimeter measurements.

4.12. NAVIGATION SYSTEM

The measurements whose models have been presented in the previous section are then blended by the LKF, which uses the non-inertial TRN measurements to update the state estimate obtained by dead-reckoning from IMU data. The mathematical formulation of the filter, as well as proof of its convergence, is given in the following [Choukroun, 2014]. The discrete state-space model on which the filter is based is defined as

$$\mathbf{x}_{k+1} = \mathbf{A}_k \mathbf{x}_k + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{4.65}$$

$$\mathbf{z}_k = \mathbf{C}_k \mathbf{x}_k + \mathbf{v}_k \tag{4.66}$$

where $\mathbf{A}$ is the state transition matrix, $\mathbf{B}$ is the input matrix, $\mathbf{C}$ is the measurement matrix, $\mathbf{x}$ is the state vector, $\mathbf{z}$ is the measurement vector, and the subscripts refer to the time step. The process and measurement noises are $\mathbf{w}_k$ and $\mathbf{v}_k$, respectively, with the following covariance matrices

$$\mathbf{Q}_k = E\{\mathbf{w}_k \mathbf{w}_k^T\} \tag{4.67}$$

$$\mathbf{R}_k = E\{\mathbf{v}_k \mathbf{v}_k^T\} \tag{4.68}$$

which are the main tuning parameters of the filter. In general, the lower the **Q**/**R** ratio, the less noisy the filter's steady-state estimate is, but also the slower its convergence, as well as its fragility to unmodeled dynamics. More about this will be said in Section 6.3. Another important covariance matrix is that of the state estimation error, defined as

$$\mathbf{P}_{k/k} = E\{\tilde{\mathbf{x}}_{k/k}\tilde{\mathbf{x}}_{k/k}^T\} \tag{4.69}$$

where $\tilde{\mathbf{x}}_{k/k} = \mathbf{x}_k - \hat{\mathbf{x}}_{k/k}$ is the state estimation error at time k given all measurements up to time k, with $\hat{\mathbf{x}}$ being the estimated state.

In general, the Kalman filter consist of two stages per time step: a prediction stage and a correction stage. For the LKF, these are expressed as following. First, the one-step ahead prediction is performed by means of the following time propagation equations

$$\hat{\mathbf{x}}_{k+1/k} = \mathbf{A}_k\hat{\mathbf{x}}_{k/k} + \mathbf{B}_k\mathbf{u}_k \tag{4.70}$$

$$\mathbf{P}_{k+1/k} = \mathbf{A}_k\mathbf{P}_{k/k}\mathbf{A}_k^T + \mathbf{Q}_k \tag{4.71}$$

Next, the measurement update stage is performed, which corrects the estimate based on new available measurements as

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1/k}\mathbf{C}_k^T\left[\mathbf{C}_k\mathbf{P}_{k+1/k}\mathbf{C}_k^T + \mathbf{R}_{k+1}\right]^{-1} \tag{4.72}$$

$$\hat{\mathbf{x}}_{k+1/k+1} = \hat{\mathbf{x}}_{k+1/k} + \mathbf{K}_{k+1}\left[\mathbf{z}_{k+1} - \mathbf{C}_{k+1}\hat{\mathbf{x}}_{k+1/k}\right] \tag{4.73}$$

$$\mathbf{P}_{k+1/k+1} = \left(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\right)\mathbf{P}_{k+1/k}\left(\mathbf{I} - \mathbf{K}_{k+1}\mathbf{C}_{k+1}\right)^T + \mathbf{K}_{k+1}\mathbf{R}_{k+1}\mathbf{K}_{k+1}^T \tag{4.74}$$

where **K** is the Kalman gain, which determines the blending proportions between process and measurement when producing the estimate update in Eq. (4.73), where the *a-priori* state $\hat{\mathbf{x}}_{k+1/k}$ is obtained from Eq. (4.70), while the term within the brackets represents the measurement estimation error. Finally, in Eq. (4.74) the covariance matrix is updated, after it had been propagated in Eq. (4.71).

Before proceeding with the implementation of the filter, values for the measurement model parameters presented in Section 4.11 are needed, and for a realistic navigation estimate modeling, the statistical properties of the TRN measurements would be necessary. However, an in-depth analysis of the performance of the TRN system is beyond the scope of this work. As such, a reverse-engineering approach will be used to determine the values of the parameters in Eqs. (4.63) and (4.64), as well as the Q/R ratio (later introduced in Section 6.3).

To this purpose, typical values for TRN horizontal position and velocity errors, as well as altitude/altitude rate errors during the PD were retrieved from literature. Specifically, velocity requirements were retrieved from Foessel-Bunting and Whittaker [2001], who mention 0.1-0.2 m/s for both horizontal and vertical velocity below 2000 m altitude. Altitude estimation requirements are set at 7.5 cm below 500 m altitude. The horizontal position estimation error is extrapolated from the horizontal velocity requirement. In fact, as they are both output of an image processing algorithm, which runs at 1 Hz, horizontal position estimation error is expected to be close to that of the horizontal velocity, in terms of dimensionless values. However, map-tie errors need to be added, which are not there in the case of velocity estimation, as they are filtered out by differencing. These considerations resulted in the following choice of parameters

$$x_{TRN} = 0.2\,\text{m} \tag{4.75}$$

$$y_{TRN} = 0.2\,\text{m} \tag{4.76}$$

$$\sigma_{z,TRN} = 0.02\,\text{m} \tag{4.77}$$

$$v_{x,TRN} = 0.2\,\text{m/s} \tag{4.78}$$

$$v_{y,TRN} = 0.2\,\text{m/s} \tag{4.79}$$

$$\sigma_{v_z,TRN} = 0.02\,\text{m/s} \tag{4.80}$$

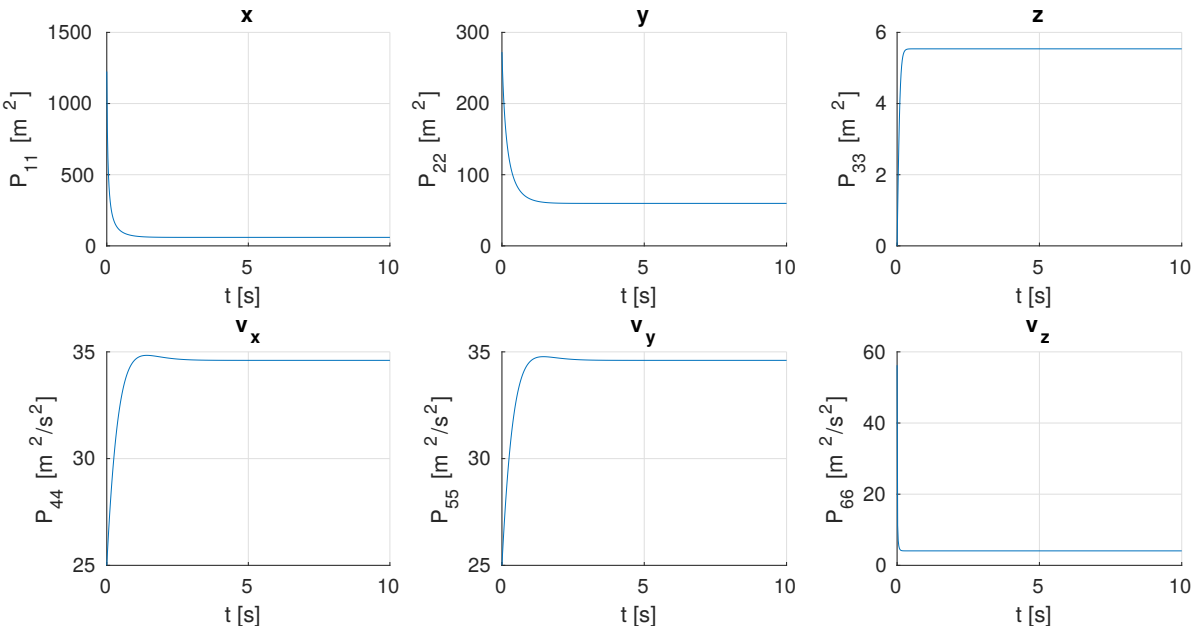


Figure 4.17: Convergence of the LKF estimation error covariance matrix diagonal components.

Now that values have been identified for the measurement model parameters, the filter must be tuned and initialized. As explained later in this section, the final tuning parameters will be given further on in the report, in Section 6.3. The initial state value is set equal to the nominal IC, while the estimation error covariance matrix is initialized as a diagonal matrix with diagonal components equal to the variances of the corresponding ICs, as presented in Section 2.2.2. This is the most logical choice for this initial guess, and was found to yield good convergence results. To check the filter's convergence, the time history of the diagonal components of the covariance matrix is shown in Figure 4.17. One can see that the curves correctly converge to their steady-state values. The top-right plot refers to the altitude component. As this IC component is fixed, its error variance has been initialized at a very low value of $0.01\,\mathrm{m}^2/\mathrm{s}^2$. To further verify the correct functioning of the filter, Figure 4.18 shows the evolution of the estimation error, for the entire process (top plots) and for the pre-convergence portion (bottom plots). One can see that the estimation errors converge to zero, or to their non-zero steady-state value in case of horizontal position (bottom-left plot). Convergence is achieved after approximately 5 s (bottom plots), and the error is stable throughout the entire descent (top plots), at least for the nominal trajectory the plots refer to.

The simulation is then run 1000 times for varying ICs, to obtain the navigation dispersions at TD, which are plotted in Figure 4.19. The plots give additional confidence that the target steady-state dispersions are successfully achieved, as horizontal position errors are contained within $\sim 0.7\,\mathrm{m}$, horizontal velocity components are lower than $0.2\,\mathrm{m/s}$, vertical errors are $< 0.1\,\mathrm{m}$ and $< 0.1\,\mathrm{m/s}$, approximately, and all the distributions are acceptably regular in shape.

MOTIVATION AND CONSIDERATIONS

The choice of a LKF is uncommon for a dynamic process like the one at hand. In fact, if measurement and process noises are zero-mean, Gaussian sequences, the LKF is optimal for linear systems. Except arguably for the process noise, none of these conditions is met, as the process is non-linear and some of the measurement noises are uniformly distributed with non-zero mean. Nevertheless, the LKF can work with biased non-Gaussian noises as well, although with a lower performance. The non-linearity of the process is a different matter, as a linear filter is expected to be susceptible to rapidly changing dynamics, such as those encountered during atmospheric flight. However, during the final part of the descent, dynamics are slower than for the entry phase of the EDL procedure, and

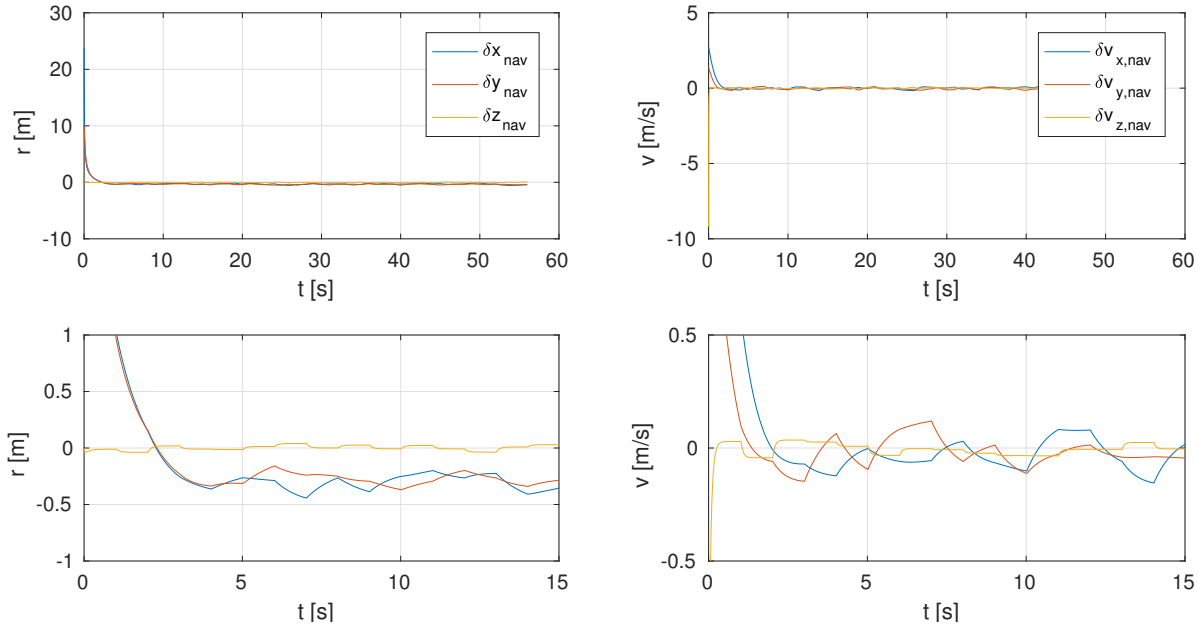


Figure 4.18: Time histories of the state components estimation error.

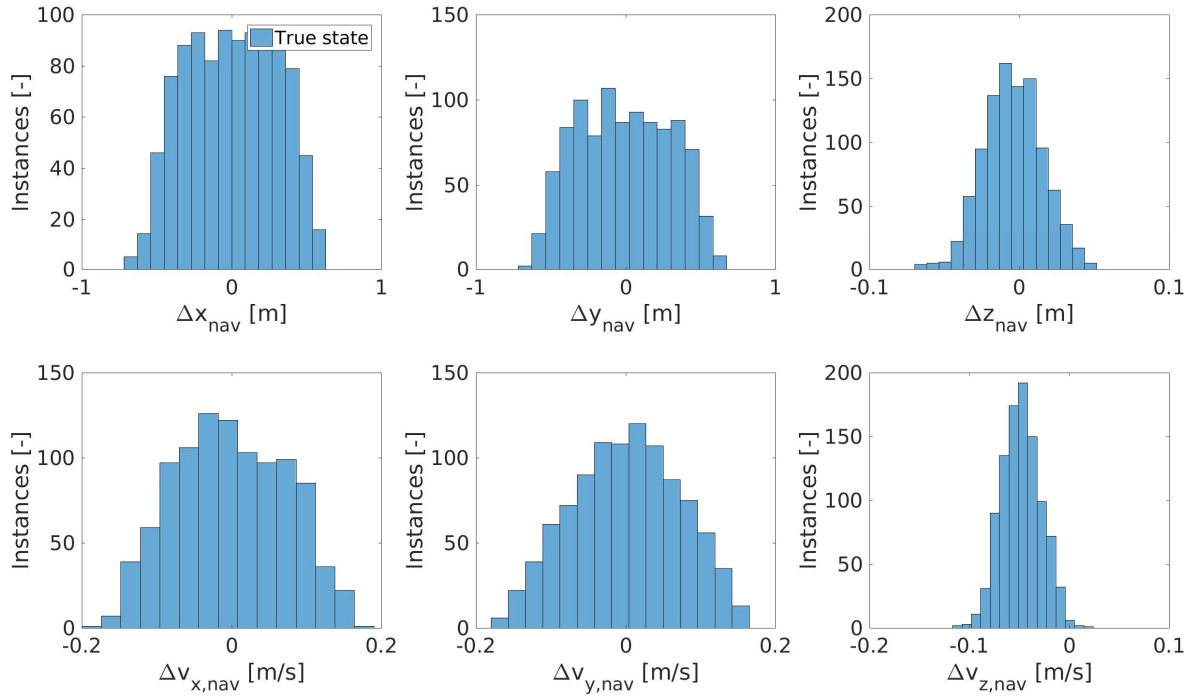


Figure 4.19: Histograms of the estimation error at TD.

the filter proved to be sufficiently robust against the non-linearities induced by the Apollo guidance scheme. In fact, Apollo produces relatively smooth trajectories, whose state components change slowly enough to be accurately estimated by the LKF running at 1 Hz. On the other hand, when the NN serves as the guidance system, the LKF runs into stability problems, as the network's output is changing more rapidly than that of the Apollo guidance, as discussed in Section 6.2.4. However, this issue has been investigated in Section 6.3, and a remedy has been found, which involves decreasing the simulation step-size and retuning the filter. The results presented above have been

obtained with the final tuning parameters, which will be introduced in Section 6.3, together with the rationale behind their choice. The selected step-size will ultimately be 0.02 s. Both navigation and guidance system are run at the maximum frequency given this step-size, that is, 50 Hz. As already said, this proved to be sufficient for ensuring LKF stability. As for the guidance system, Gaudet and Furfaro [2014] use a frequency of 100 Hz, which was not tested here. In fact, simulation times and datafile sizes are already demanding at 50 Hz for the largest simulation sets that have been carried out throughout this work. Besides, given that guidance errors are negligible with respect to navigation errors, as shown in Section 6.2.1, a higher frequency is believed to bring about similarly minimal effects on the final landed accuracy.

Ultimately, the reason behind the decision of implementating a navigation filter within a guidance-focused framework was that of providing a more realistic subset of input errors than a different model would have otherwise offered. In fact, the objective was that of replicating the delay in availability of accurate state knowledge due to the convergence phase of the filter, rather than the steady-state estimation error, which can be acceptably modeled by a Gaussian noise of lower variance. However, an Extended Kalman Filter (EKF) demanded an excessive time investment to implement, as no built-in Simulink block is available, which is the case for the LKF. This matter will be touched upon once more in Sections 6.3 and 7.3.3, where conclusion with this respect will be drawn.

4.13. ACTUATOR MODEL

The actuator model includes both thrust misalignment and thrust noise.

The thrust magnitude is corrupted with a white multiplicative noise. This noise represents inaccuracies in the actuation of the engine throttle to achieve the mass flow commanded by the control system. Since the latter is assumed to be perfect, the commanded mass flow directly follows from the acceleration vector commanded by the guidance system and system mass knowledge (assumed to be perfect as well). One can thus write

$$\|\mathbf{F}_P\| = \|\mathbf{F}_{P,cmd}\| + \|\mathbf{F}_{P,cmd}\|\nu_P \tag{4.81}$$

The random variable ν_P is modeled by a normal distribution of zero-mean and standard deviation σ_P. The output of this is easily verified by means of a Monte Carlo (MC) analysis. Given $\sigma_P = 0.001$, Figure 4.20 shows the quantity $\frac{\|\mathbf{F}_P\|}{\|\mathbf{F}_{P,cmd}\|}$ over the runs and its histogram plot, in which a normal distribution with the given standard deviation can be recognized.

The thrust misalignment is defined by means of thrust elevation ε_T and thrust azimuth ψ_T, where the former is the angle between the thrust vector and the z-axis of the $\mathcal{P}$-frame, while the latter is the direction of such vector as defined by a positive rotation around the z-axis, and referenced to the x-axis of the $\mathcal{P}$-frame. Given these definitions, one can write the misaligned thrust vector as

$$F_{P,x} = \|\mathbf{F}_P\| \sin(\varepsilon_T)\cos(\psi_T) \tag{4.82}$$

$$F_{P,y} = \|\mathbf{F}_P\| \sin(\varepsilon_T)\sin(\psi_T) \tag{4.83}$$

$$F_{P,z} = \|\mathbf{F}_P\| \cos(\varepsilon_T) \tag{4.84}$$

This can then be converted back into $\mathcal{B}$-coordinates. As this model is supposed to represent constant misalignments of the engine structure, the variables ε_T and ψ_T are constants randomized across different simulations, namely as

$$\varepsilon_T = |N(0,\sigma_T)| \tag{4.85}$$

$$\psi_T = U(0,2\pi) \tag{4.86}$$

One can see from Figure 4.21 that, with $\sigma_T = 0.01^{\circ}$, both variables do behave as expected, that is, ε_T as a folded normal distribution with standard deviation σ_T, and ψ_T as a uniform distribution between 0 and 2π.

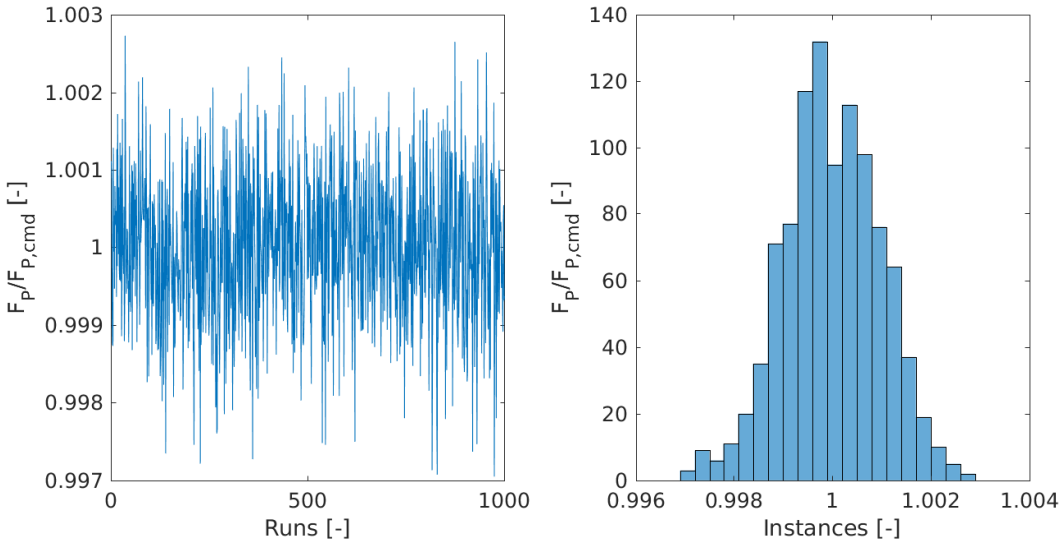


Figure 4.20: History and histogram plots of the ratio between corrupted and commanded thrust vectors.

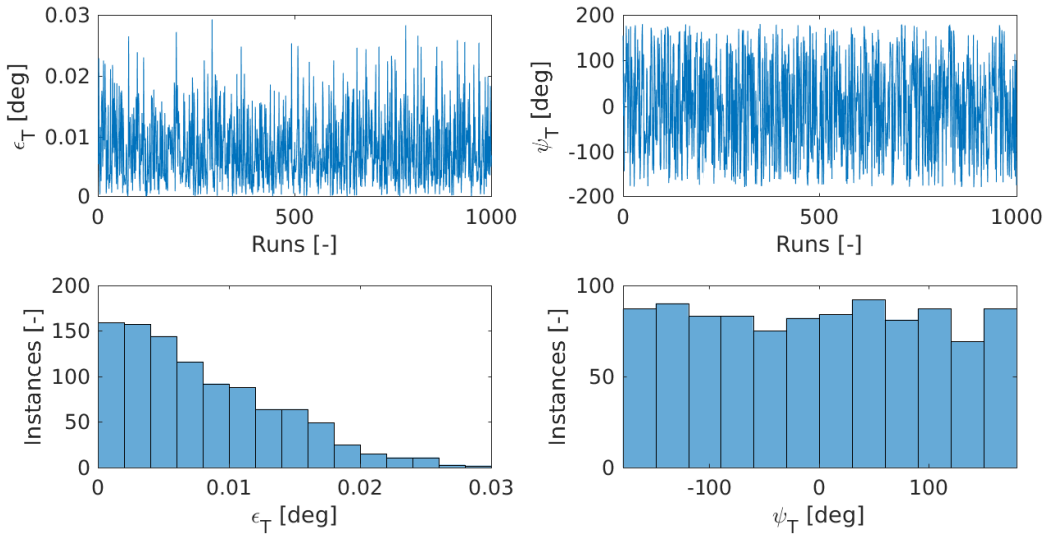


Figure 4.21: History and histogram plots of ε_T and ψ_T.

4.14. SYSTEM VERIFICATION

After the verification of the single blocks described above, a Verification and Validation (V&V) of the full model needs to be carried out. Unfortunately, a proper validation was not possible, given the lack of real flight data or already-validated, higher-fidelity simulators. As such, only a system verification will be carried out. As mentioned in Section 4.5, the implementation of the Apollo guidance allowed to test the new blocks in closed loop, as they were added. On top of their stand-alone verification, their integration with the already existing simulator components was thus gradually verified. In the concluding section of this chapter, a further verification of the entire system will be reported.

First, in Figure 4.22, one can see the comparison between true state and acceleration components for the ideal simulator kernel, already verified in Section 4.5 in closed-loop with the Apollo guidance, and for the full model. One can see that the profiles are quite close to each other, with some expected differences. Most noticeably, one can see the noise in vertical acceleration, mostly

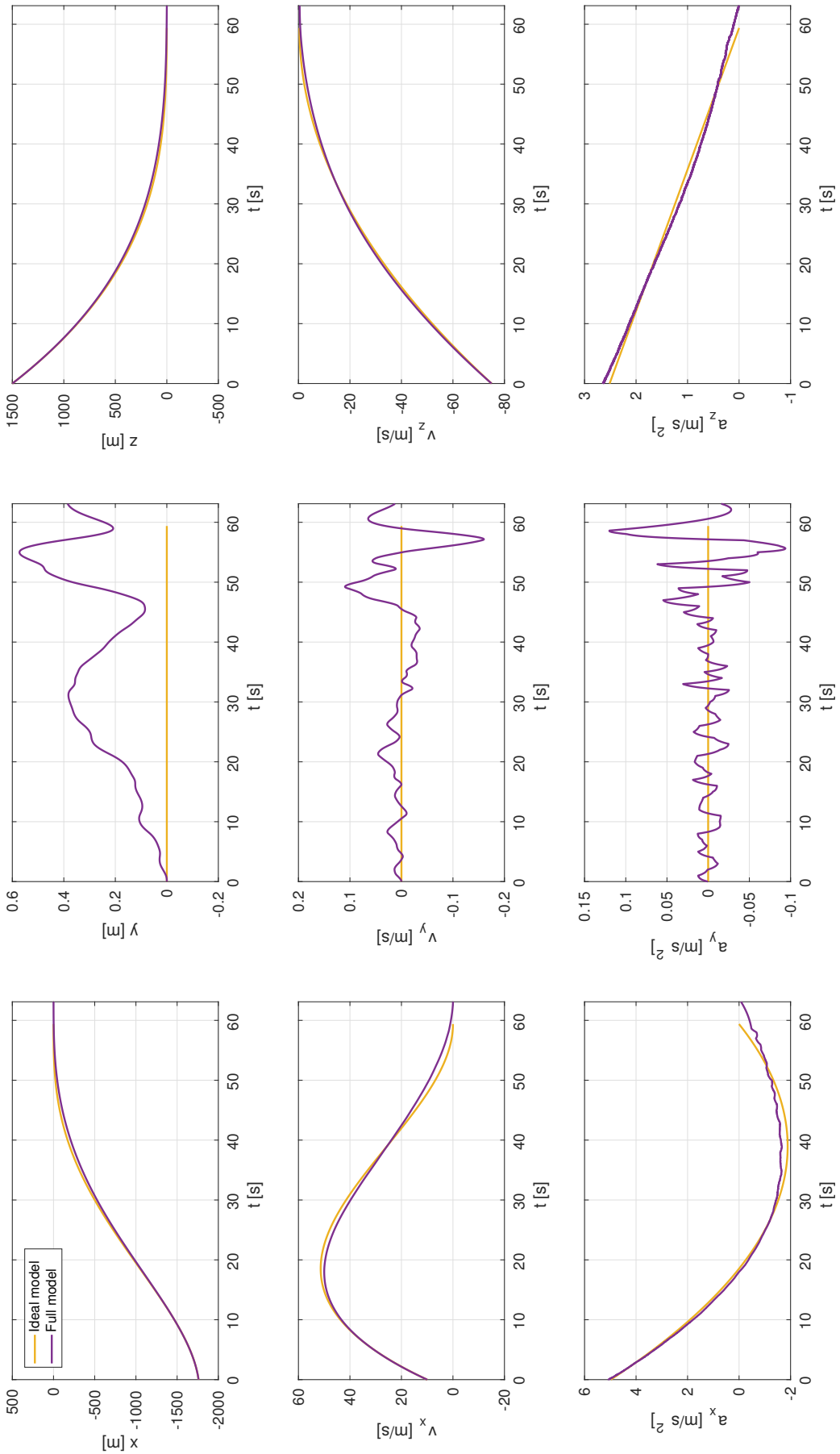


Figure 4.22: Simulator system verification plots.

caused by the commanded acceleration, and otherwise zero to numerical precision in the ideal case (as the trajectory is planar). The noise is due to the navigation system's steady-state noise, which affects the guidance output ever more as state components values become smaller. The acceleration is also propagated correctly, as cross-track position and velocity can be seen to vary accordingly. One can make similar considerations about downrange and vertical components. Although less visibly, given the larger components values, noise can be seen in these cases as well, and the integration can be said successfull from visual checks. Finally, the simulation time is a few seconds longer than it is in the ideal case, which is also expected, as counteracting the perturbations leads to longer flight times. Ultimately, the system seems to be working correctly for the purpose it has been designed and developed for. In fact, the full model yields results which are close to the already-verified simulator kernel, while featuring the effects of the additional blocks which have have been included subsequently.

5

GUIDANCE ALGORITHM DEVELOPMENT

The approach taken here to the development of an optimal guidance algorithm follows closely that of Gaudet and Furfaro [2014]. It makes use of RL to learn the control policy (that is, a series of actions) in a direct way, that is, without the use of a value function. The reason for this will be given in Section 5.3, where the optimization strategy, as well as the choice of Artificial Neural Networks (ANNs) as the policy modeling method, will be discussed. In the same section, the modifications and the additions to the reference approach will be highlighted. A brief overview of the method will be given here. The top-level architecture of the whole algorithm is shown in Figure 5.1, which will be used as a reference for the following introduction.

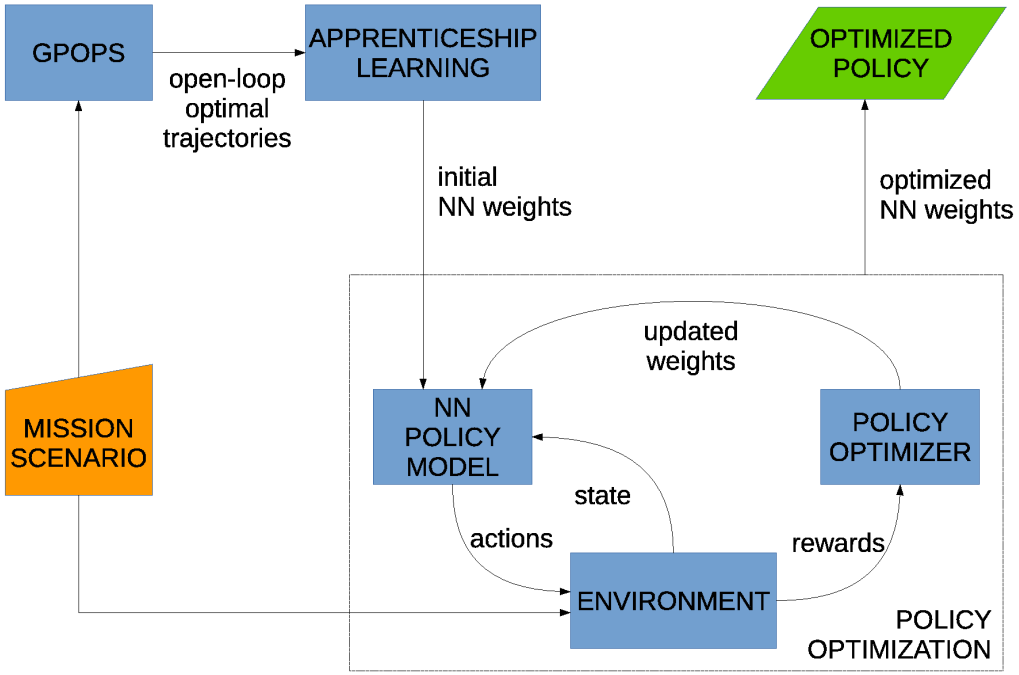


Figure 5.1: Top-level architecture of the optimization algorithm.

The chosen policy modeling method (i.e., ANNs, which can be identified as the *NN policy model* block in the optimization loop in Figure 5.1) is a function approximation method that provides a way to continuously map inputs to outputs, while making use of a relatively low number of parameters.

These parameters will then be the object of the optimization process. To this end, the parameters are first initialized by means of a series of open-loop optimal trajectories, generated with the Gauss Pseudospectral Optimal Control Software (GPOPS) package, which can be seen in the top-left part in Figure 5.1. This software, which will be introduced in Section 5.2, is an Optimal Control Problem (OCP) solver, able to compute fuel-efficient trajectories under ideal conditions. These can then be used as a training data set for a first, approximate optimization of the NN parameters by means of traditional supervised learning techniques, such as the Levenberg-Marquardt or the Bayesian regularization methods (Section 5.1). This process, referred to as *apprenticeship learning* in Figure 5.1 and discussed in detail in Section 5.2.2, is far from yielding a policy which is optimal in an uncertain environment, as the training data set is obtained from a deterministic optimization, but it contributes to speeding up the subsequent RL-based optimization.

Once the NN weights have been initialized, the policy optimization process begins (Section 5.3). The initial policy is used to simulate a set of sample trajectories in an uncertain environment (see Figure 5.1). A stochastic search algorithm will be used to improve the resulting utility. The NN parameters are perturbed and the simulation is repeated. If improvements in the utility are found, the parameters are updated accordingly (*policy optimizer* block in Figure 5.1). The stochastic nature of this search algorithm ensures that *exploration* takes place, which is a very important aspect of a good RL optimization method.

5.1. ARTIFICIAL NEURAL NETWORKS

The policy to be learned within the RL framework needs somehow to be modeled. That is, one needs a way to map from states to actions, so that for each potential lander state, the guidance can take an appropriate control action. ANNs have the capability of modeling complex nonlinear functions, which makes them appealing for the problem at hand.

In Section 5.1.1, the main types and features of ANNs will be introduced, and the mathematical formulation of the chosen type of ANN will be given in Section 5.1.2. Section 5.1.3 will look into the software used and its verification. The apprenticeship training of the network for the problem at hand will be covered in Section 5.2.2, after the GPOPS software will have been introduced.

5.1.1. OVERVIEW

First of all, the architecture of a NN needs to be defined. A neural network is made up by one input layer, one output layer, and ,possibly, a number of hidden layers. Figure 5.2 shows the general architecture of a feedforward ANN. As can be seen from the figure, each node of each layer is connected to every node of the adjacent layers. A weight is associated to each of these connections, whereas each node represents an activation function. This function is applied to the node's input, resulting from its incoming connections, and produces an output, which is propagated forward, via the outgoing connections, to the next layer. Often, a bias is associated to each node as well.

One can identify different types of ANNs. When no hidden layers are present, the network is a single-layer neural network. If the network has one or more hidden layers, it is referred to as a Multi-layer Neural Network (MNN). If a MNN has more than one hidden layer, it is called a deep neural network, whereas if it only has one hidden layer, it is a shallow neural network. In this work, single-layer MNNs will be used, as further discussed in Section 5.1.3.

5.1.2. MATHEMATICAL FORMULATION

In this section, the underlying mathematical formulation of NNs will be explained. Considering a shallow MNN such as the one in Figure 5.2, let us assign the subscripts i, j and k to refer to the nodes of the input, hidden and output layer, respectively. The weights from the input to the hidden layer are denoted as w^h_{ij}, while those from the hidden to the output layer are w^o_{jk}. Biases are referred to as b^h_j and b^o_k, for hidden and output layer, respectively.

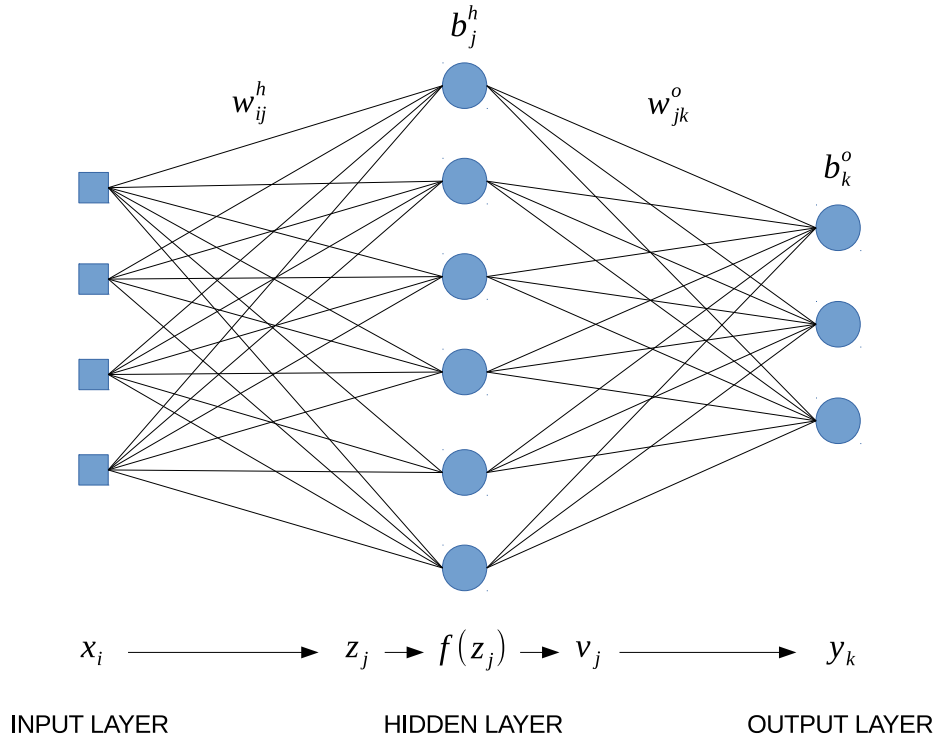


Figure 5.2: General architecture of a feedforward ANN with one hidden layer.

The input to the hidden layer nodes is a linear combination of the inputs x_i, which is denoted with z_j and is expressed as

$$z_j = \sum_{i=1}^{N_i} w_{ij}^h x_i + b_j^h \quad \text{for } j = 1, ..., N_j \tag{5.1}$$

where N_i and N_j are the number of nodes in the input and hidden layers, respectively. In algebraic form, this becomes

$$\mathbf{z} = \mathbf{W}^h \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \tag{5.2}$$

where $\mathbf{W}^h$ is the $N_j \times (N_i + 1)$ joint matrix of hidden layer weights and biases in matrix form, that is

$$\mathbf{W}^h = \begin{bmatrix} w_{11}^h & \dots & w_{N_i 1}^h & b_1^h \\ \vdots & \ddots & \vdots & \vdots \\ w_{1N_j} & \dots & w_{N_i N_j}^h & b_{N_j}^h \end{bmatrix} \tag{5.3}$$

An activation function is associated to each node of the hidden and output layers. Different types of activation functions exist. The most common ones for the hidden layer are sigmoidal function, hyperbolic tangent and linear saturated function. These functions' purpose is that of mapping the input of the hidden layer nodes to an output bound between specific thresholds, typically $(0, 1)$ for a sigmoidal function and $(-1, 1)$ for the hyperbolic tangent function and the linear saturated function, although they are actually arbitrary. The effect of different activation functions was not investigated for this work: the tangent hyperbolic function was used, which is also the standard activation function of the Matlab Neural Network Toolbox (see Section 5.1.3), under the name `tansig`. The activation function is applied to the weighted sums of the inputs z_j and produces the hidden layer nodes outputs v_j:

$$v_j = \tanh(z_j) \quad \text{for } j = 1, ..., N_j \tag{5.4}$$

or in algebraic form

$$\mathbf{v} = \tanh(\mathbf{z}) \tag{5.5}$$

The inputs of the output layer nodes are produced analogously to those of the hidden layer, that is

$$y_k = \sum_{k=1}^{N_k} w_{jk}^o v_j + b_k^o \quad \text{for } k = 1, ..., N_k \tag{5.6}$$

where N_j and N_k are the number of nodes in the hidden and output layers, respectively. In algebraic form, this becomes

$$\mathbf{y} = \mathbf{W}^o \begin{pmatrix} \mathbf{v} \\ 1 \end{pmatrix} \tag{5.7}$$

where $\mathbf{W}^o$ is the $N_k \times (N_j + 1)$ joint matrix of output layer weights and biases in matrix form, that is

$$\mathbf{W}^o = \begin{bmatrix} w_{11}^h & \cdots & w_{N_j 1}^o & b_1^o \\ \vdots & \ddots & \vdots & \vdots \\ w_{1N_k} & \cdots & w_{N_j N_k}^h & b_{N_k}^o \end{bmatrix} \tag{5.8}$$

Most commonly, the activation function for the output layer is simply a static function. Therefore, the inputs y_k to the output layer nodes are actually the outputs of the NN itself.

5.1.3. TRAINING METHOD AND SOFTWARE

Training of a neural network refers to the process of finding the NN's weights and biases that minimize a function of the error between the output of the network and the desired output. The baseline training method for ANNs is supervised learning. This means that weights and biases are learned from a set of input and output data, in a batch or sequential fashion. This is as opposed to unsupervised learning, which is substantially what happens during the RL policy optimization that will be presented in Section 5.4.

For the moment, let us focus on the concept of supervised learning. The most common and widely applied NN training algorithm is error back-propagation, introduced by Rumelhart et al. [1986]. This algorithm consists of a two-stage iterative approach. Given a training dataset of input-output pairs, a feedforward sweep is first made to compute the NN output from the training inputs with the current NN weights and biases, which are initialized randomly for the first iteration. This is done by means of the equations presented in Section 5.1.2, namely Eqs. (5.2), (5.5) and (5.7). Once the outputs have been computed, they are compared to the outputs from the training dataset, and the error is propagated backwards to adjust the weights and biases so as to decrease the output error itself. The cost function is typically a function of the squared error. Most commonly, this is:

$$J(\mathbf{w}) = \frac{1}{2} \sum_{k=1}^{N_k} e_k^2 \tag{5.9}$$

where $e_k = y_{tr} - y_k$, with y_{tr} being one sample from the training dataset outputs. Weights and biases are then adjusted proportionally to the gradient of this function with respect to the weights vector $\mathbf{w}$. This can be done for one sample at a time, in which case the training is sequential, or for the whole dataset at once, in the case of batch training. In this case, an average of the gradient is computed at the end of each sweep, and the weights adjusted accordingly.

For the original and simplest first-order backpropagation algorithm, the weight adjustment rule is as follows:

$$\mathbf{w}_{n+1} = \mathbf{w}_n + \alpha_n \nabla_{\mathbf{w}} J(\mathbf{w}_n) \tag{5.10}$$

where α is the learning rate or step-size, and $\mathbf{w}_n$ is the above-mentioned weights and biases vector at the n-th epoch (or sweep, in the case of batch training). This is made up as follows:

$$\mathbf{w} = \begin{pmatrix} \mathbf{w}^h \\ \mathbf{b}^h \\ \mathbf{w}^o \\ \mathbf{b}^o \end{pmatrix} \tag{5.11}$$

Higher-order gradient backpropagation algorithms differ in the way they compute the gradient, but the structure of the update rule is analogous to that of Eq. (5.10).

The most important design aspects of a MNN are the number of hidden layers and the number of nodes in such layers. As stated by Cybenko [1989], a shallow MNN is theoretically enough to approximate any kind of nonlinear continuous function, given a sufficient number of hidden-layer nodes. Adding a second hidden layer can improve fitting capabilities, but it takes longer to train, and the risk of over-fitting (i.e., fitting undesired noise in the target data) increases, at the expense of generalization [Babuska, 2014]. For these reasons, and because of time constraints which do not allow for a proper exploration of the topic, we will only concern ourselves with shallow MNNs for this work. As for the number of nodes in the hidden layer, it will be left open to investigation. Gaudet and Furfaro [2014] use a 32 hidden-layer nodes network. However, their study features a 2-dimensional landing scenario (4 inputs, 2 outputs), whereas the problem at hand is 3-dimensional (6 inputs, 3 outputs). It is thus likely that a higher number of hidden nodes will be needed for the problem at hand.

MATLAB NEURAL NETWORK TOOLBOX

The supervised training of the network will be done via the Matlab Neural Network Toolbox. The most widespread training method is arguably Levenberg-Marquardt, a backpropagation algorithm, which is also the default method of the Matlab toolbox. It is a second-order gradient algorithm, which is very fast for medium-sized networks, up to several hundred weights. The number of elements of the weights and biases vector is given by

$$N_{wb} = N_j(N_i + N_k + 1) + N_k \tag{5.12}$$

from which the total number of weights and biases is computed for different numbers of hidden-layer nodes, and is reported in Table 5.1. Since it is believed that a number of hidden-layer neurons below 100 will be sufficient for approximating the guidance policy for this problem, the Levenberg-Marquardt algorithm is a suitable method for these network sizes.

Table 5.1: Total number of weights and biases for different numbers of hidden-layer nodes, given $N_i = 6$ and $N_k = 3$.

N_j	N_{wb}
25	253
50	503
75	753
100	1003

However, the toolbox provides another training algorithm, called Bayesian regularization. This method is based on Levenberg-Marquardt, but it is specifically modified to avoid data overfitting and, thus, achieve better generalization. Although the generation of the optimal trajectories and

the subsequent apprenticeship learning represents the *offline*[1] part of the optimization process, it is nonetheless important to limit the time consumption of this part of the process. In fact, the generation of the optimal trajectories can be quite time demanding, and this can limit the amount of different apprenticeship learning cases one can test. As such, a training algorithm which can better generalize, thus allowing for learning from a smaller set of training data, is highly desirable. For these reasons, the Bayesian regularization algorithm is the chosen training method for carrying out apprenticeship learning.

5.2. OPTIMAL CONTROL SOFTWARE

As already mentioned, an apprenticeship training of the NN is necessary to considerably speed up the RL optimization time. This could be done, in principle, starting from any set of data which gets closer to the desired solution than just a random inizialization would. For instance, one could use Apollo guidance trajectories as the training dataset for apprenticeship learning. However, using optimal trajectories is expected to lead to a better result, as the sample pairs from these trajectories enclose not only safe and accurate TDs, but fuel optimality as well. This comes at the expense of much longer times to generate a dataset comparable in size. As mentioned in Section 5.1.3, as long as this time does not exceed reasonable limits, it does not represent an issue, as the apprenticeship learning procedure is only carried out a limited number of times.

The OCS chosen for performing this operation is the GPOPS package, presented by Rao et al. [2010], which implements a direct collocation method, called Gauss Pseudospectral Method (GPM), to solve OCPs. This choice is mainly due to GPOPS being a non-commercial, open-source software. However, the GPM is converted into a Non-Linear Programming (NLP) problem, which GPOPS is set up to solve with the Sparse Nonlinear OPTimizer (SNOPT) software package [Gill et al., 2005]. Besides being freely available, GPOPS has a number of further advantages. Firstly, its reusability makes it easily adjustable to the specific problem of interest. Additionally, it is efficient in vectorized environments such as Matlab. Finally, its user-friendly interface makes it fast to learn and easy to set up.

5.2.1. VERIFICATION

By following the manual by Rao et al. [2008] the OCP is set up. The state vector is

$$\mathbf{x} = \begin{pmatrix} x & y & z & v_x & v_y & v_z & m \end{pmatrix}^T \tag{5.13}$$

and the control vector is

$$\mathbf{u} = \begin{pmatrix} u_x & u_y & u_z \end{pmatrix}^T \tag{5.14}$$

The boundaries for the final time are chosen as $t_f \in [30, 60]$, which has been found by trial and error to leave enough room for all the intended initial conditions. The constraint on the thrust is added as

$$T_{min} < m\|\mathbf{u}\| < T_{max} \tag{5.15}$$

The cost function is expressed as

$$J = \int_{t_0}^{t_f} m(t)\|\mathbf{u}(t)\| dt \tag{5.16}$$

[1] Also the RL part of the optimization is actually carried out offline, as it happens prior to a hypothetical on-board implementation of the optimized policy. However, by *offline* it is here meant that optimal trajectory generation and apprenticeship learning take place out of the RL optimization loop, and are therefore only carried out "una tantum".

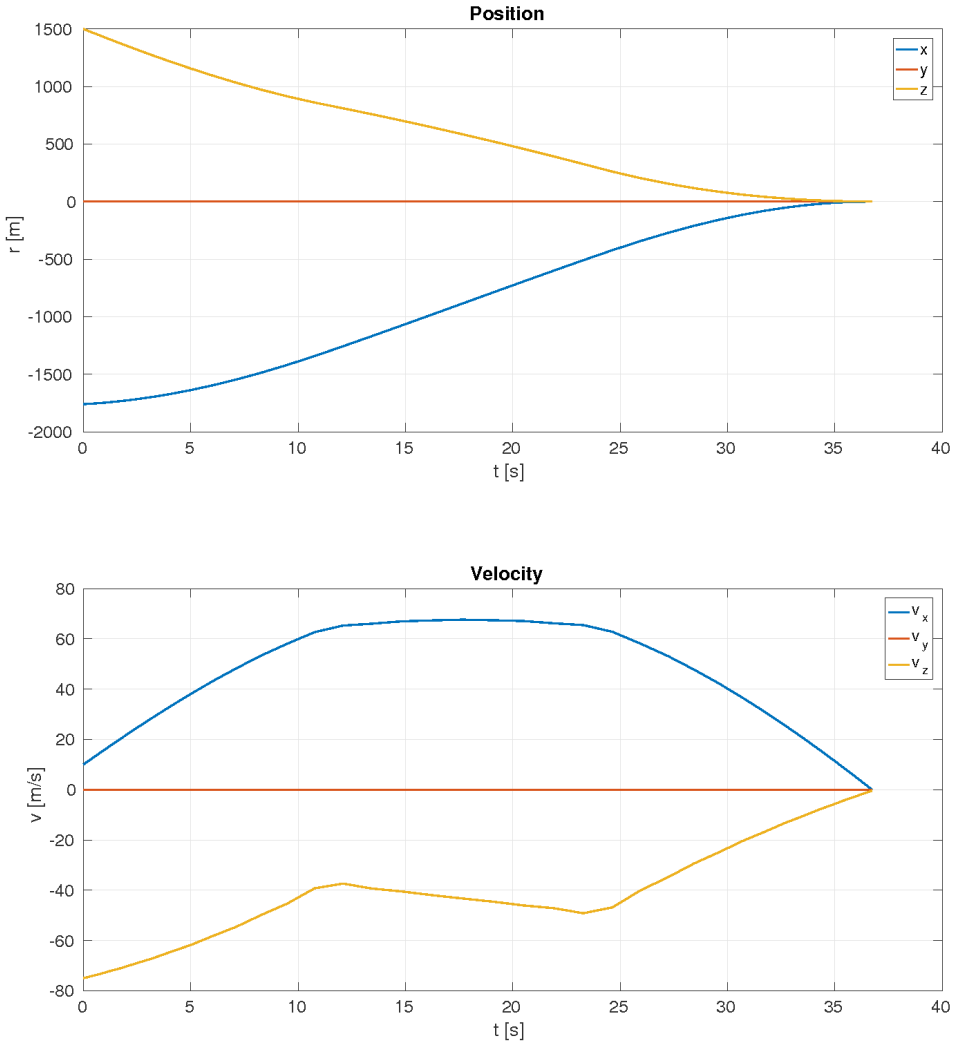


Figure 5.3: Position and velocity plots of the optimal control solution.

Finally, the differential equations are

$$\dot{x} = v_x \tag{5.17}$$
$$\dot{y} = v_y \tag{5.18}$$
$$\dot{z} = v_z \tag{5.19}$$
$$\dot{v}_x = u_x \tag{5.20}$$
$$\dot{v}_y = u_y \tag{5.21}$$
$$\dot{v}_z = u_z - g_{Mars} \tag{5.22}$$
$$\dot{m} = -\frac{m\|\mathbf{u}\|}{g_0 I_{sp}} \tag{5.23}$$

where $g_{Mars} = 3.7114\,\mathrm{m/s^2}$ [Gaudet and Furfaro, 2014].

Figures 5.3 and 5.4 show the plots of the state and control variables resulting from the solution

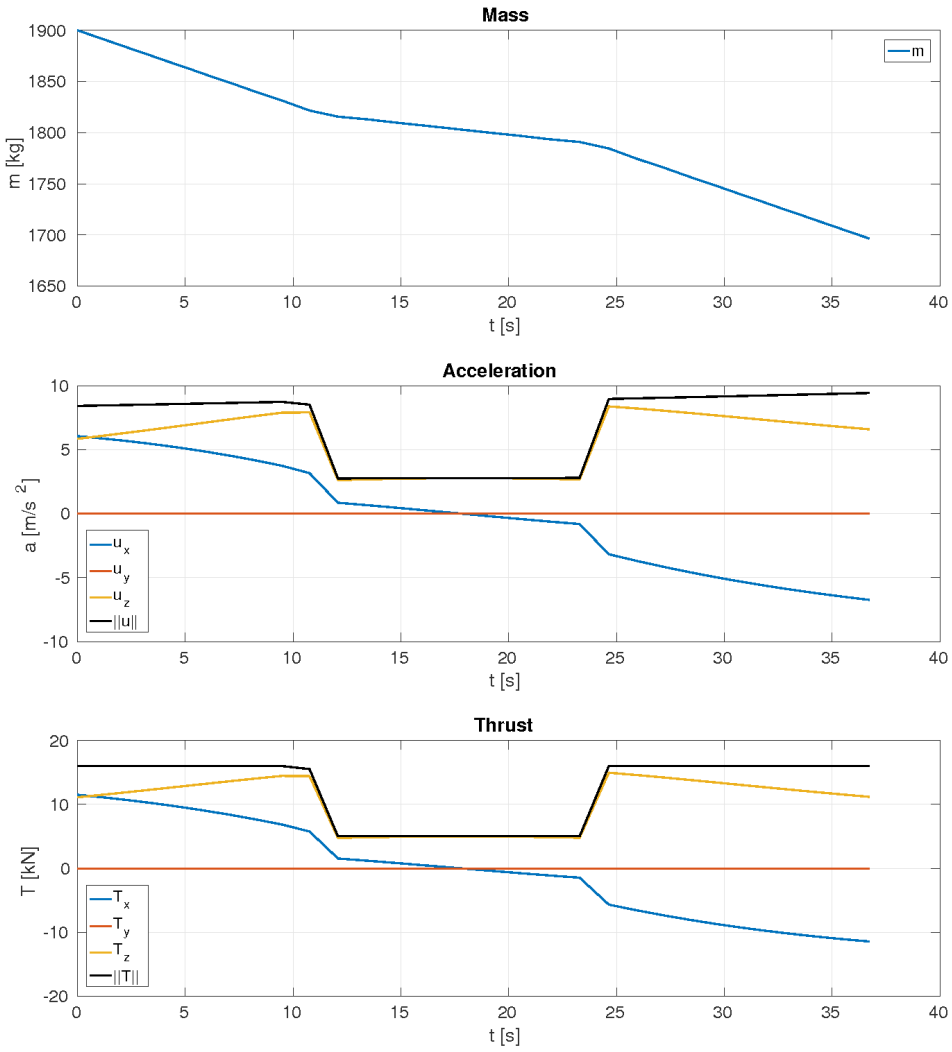


Figure 5.4: Mass, acceleration (control variable) and thrust plots of the optimal control solution.

of the OCP described above. The initial conditions were set to

$$\mathbf{r}_0 = \begin{pmatrix} -1760 & 0 & 1500 \end{pmatrix}^T \text{m} \tag{5.24}$$

$$\mathbf{v}_0 = \begin{pmatrix} 10 & 0 & -75 \end{pmatrix}^T \text{m/s} \tag{5.25}$$

$$m_0 = 1900\,\text{kg} \tag{5.26}$$

and the final conditions were

$$\mathbf{r}_f = \begin{pmatrix} 0 & 0 & 0 \end{pmatrix}^T \text{m} \tag{5.27}$$

$$\mathbf{v}_f = \begin{pmatrix} 0 & 0 & -0.4 \end{pmatrix}^T \text{m/s} \tag{5.28}$$

$$m_f = \text{free} \tag{5.29}$$

It can be seen that the states correctly converge to the target values, and that the thrust magnitude is correctly constrained within the set minimum and maximum values. The typical max-min-max thrust profile is also clearly identifiable. Minimum and maximum thrust values have been set to $\|T\| \in [5000, 16000]$m.

For further verification of the applicability of the GPOPS solution to the task of supervised training of the NN, it is useful to make a comparison between the mass consumption as obtained from

GPOPS and from the Apollo guidance benchmark algorithm. A grid of initial conditions, which will also be used for the supervised learning phase, as explained in Section 5.2.2, is set up with four equally spaced values per state component, except for altitude and mass, which are kept equal to their nominal values $h_0 = 1500\,\text{m}$ and $m_0 = 1900\,\text{kg}$. The maximum and minimum values for the other state components are chosen as follows:

$$\mathbf{r}_0^{minmax} = \mathbf{r}_0 \pm 3\boldsymbol{\sigma}_r^{IC} \tag{5.30}$$

$$\mathbf{v}_0^{minmax} = \mathbf{v}_0 \pm 3\boldsymbol{\sigma}_v^{IC} \tag{5.31}$$

where $\mathbf{r}_0$ and $\mathbf{v}_0$ are the same as in Eq. (5.26), and

$$\boldsymbol{\sigma}_r = \begin{pmatrix} 35 & 16.5 & 0 \end{pmatrix}^T \text{m} \tag{5.32}$$

$$\boldsymbol{\sigma}_v = \begin{pmatrix} 5 & 5 & 7.5 \end{pmatrix}^T \text{m/s} \tag{5.33}$$

as defined in Section 2.2.2.

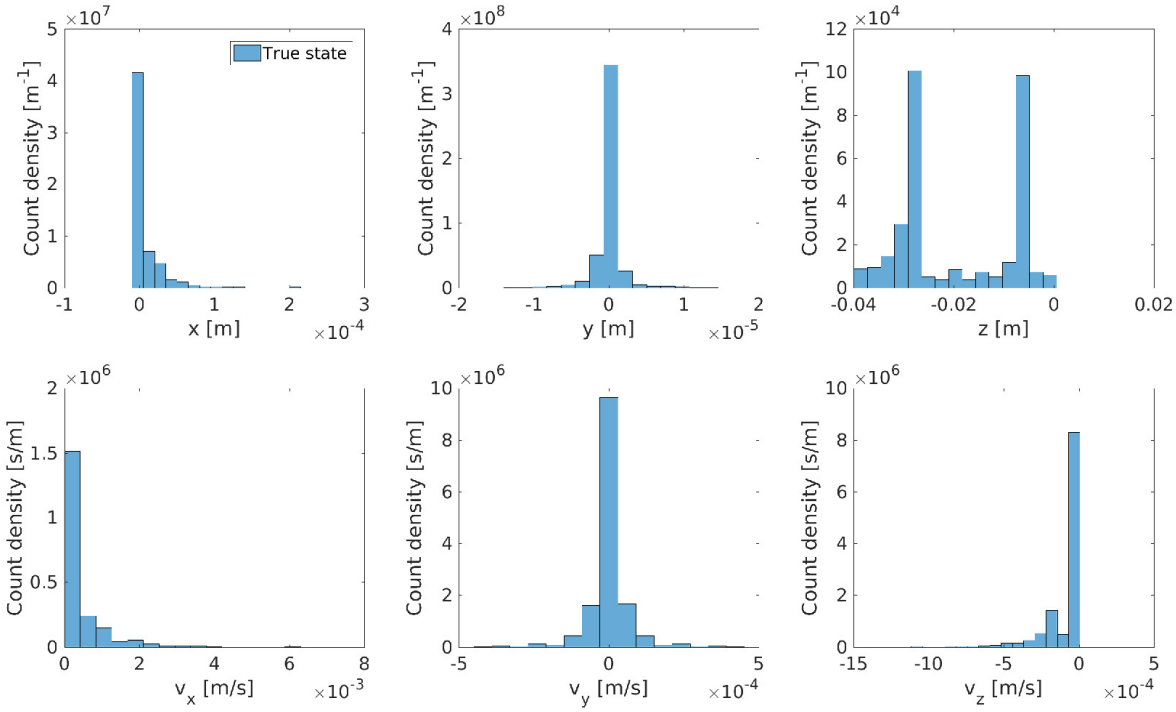
For these initial states, both the optimal control solutions and the Apollo guidance solutions are found, where the latter are obtained via perturbation-free simulations, to obtain comparable results[2]. For the same reason, the GPOPS solution is not directly compared to the Apollo guidance results. In fact, in the OCS, final conditions are constrained to be identically equal to the set ones. By simulating the open-loop solution and using the resulting residuals at TD, one can better compare them to the benchmark.

Figures 5.5 and 5.6 show the comparison between position and velocity errors and mass consumptions at TD. In Figure 5.5a one can see that there are some position and velocity residuals for the Apollo guidance trajectories. These are due to two reasons. First, as explained in Section 4.5, the implemented Apollo guidance stops computing the solution to the TPBVP when the expected time-to-go falls below a certain threshold (in this case, $t_{go}^{thres} = 5\,\text{s}$), to avoid the singularity that results from t_{go} approaching zero. After this moment, the algorithm is working in open-loop mode, and small errors at TD will thus result. Secondly, the simulator is programmed to stop the simulation when it detects the first negative altitude. Because of the fixed time-step integrator used (4-th order Runge-Kutta with $\Delta t = 0.1\,\text{s}$), this results in non-negligible negative final altitudes, which induce further errors in the other state components. However, these errors remain very small at maximum values of $\sim 10^{-5}\,\text{m}$ for horizontal position and $\sim 10^{-2}\,\text{m/s}$ for velocity.

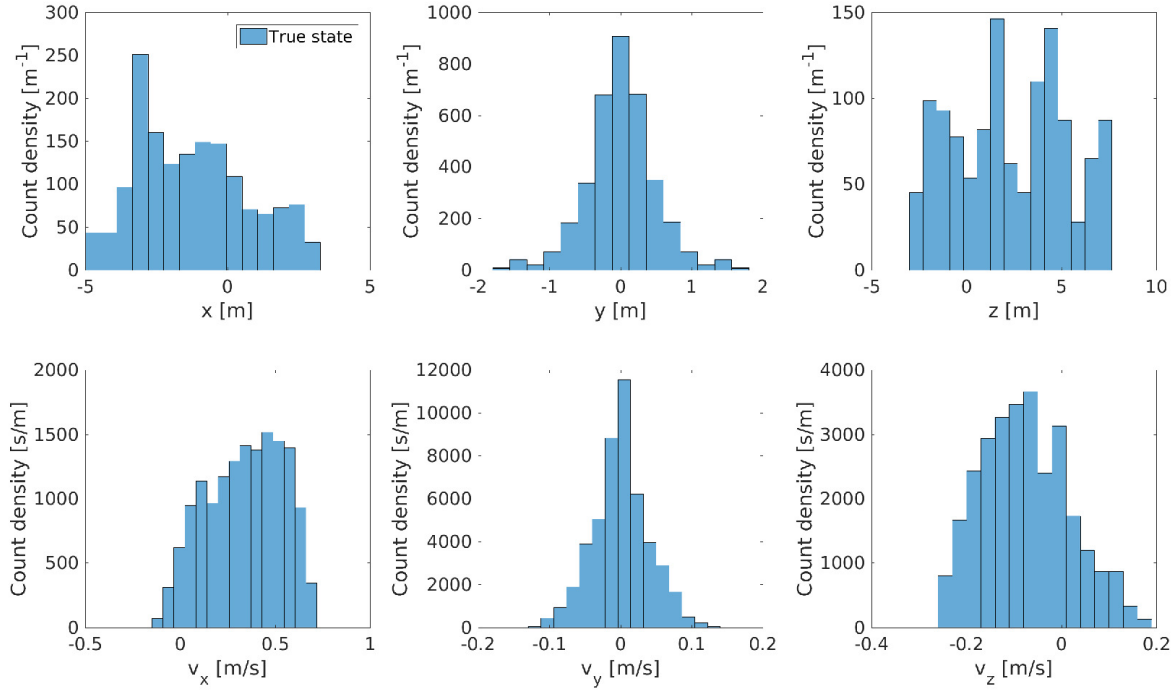
On the other hand, position and velocity residuals for the open-loop optimal trajectories are much larger. This could be due to multiple causes. First, the trajectories are indeed open-loop, contrary to the Apollo guidance. This means that, since commands are not recomputed at each time step for the current state, errors propagate and amplify as simulation time increases. Besides, the number of nodes in the GPM, which determines the points for which the optimal solution is computed, was limited to 22 for computational reasons. The solution between these points is found by linear interpolation over time when simulating the open-loop trajectories. The fewer these points, the larger the interpolation error will be. This explains why errors propagate to such large values at TD. Finally, also positive altitude errors are there due to this reason. In fact, contrary to the Apollo guidance case, the simulation is stopped when the final time, as obtained from the GPOPS solution, is reached, as there is no information about the optimal solution after that time.

To the end of validating the GPOPS optimal trajectories for the purpose of NN apprenticeship training, it is also interesting to look at the propellant consumption comparison between the two sets of trajectories. This is shown in Figure 5.6. One can see that the optimal solution outperforms the Apollo guidance, as expected. Besides, the Apollo guidance showed to be close to the optimal

[2]The optimal trajectories are obtained in an ideal environment, where the only forces acting on the S/C are a constant vertical gravitional force and the thrust force.



(a) Position and velocity errors for the Apollo guidance.



(b) Position and velocity errors for the open-loop optimal trajectories.

Figure 5.5: Position and velocity error histograms comparison between Apollo guidance and simulated open-loop optimal trajectories.

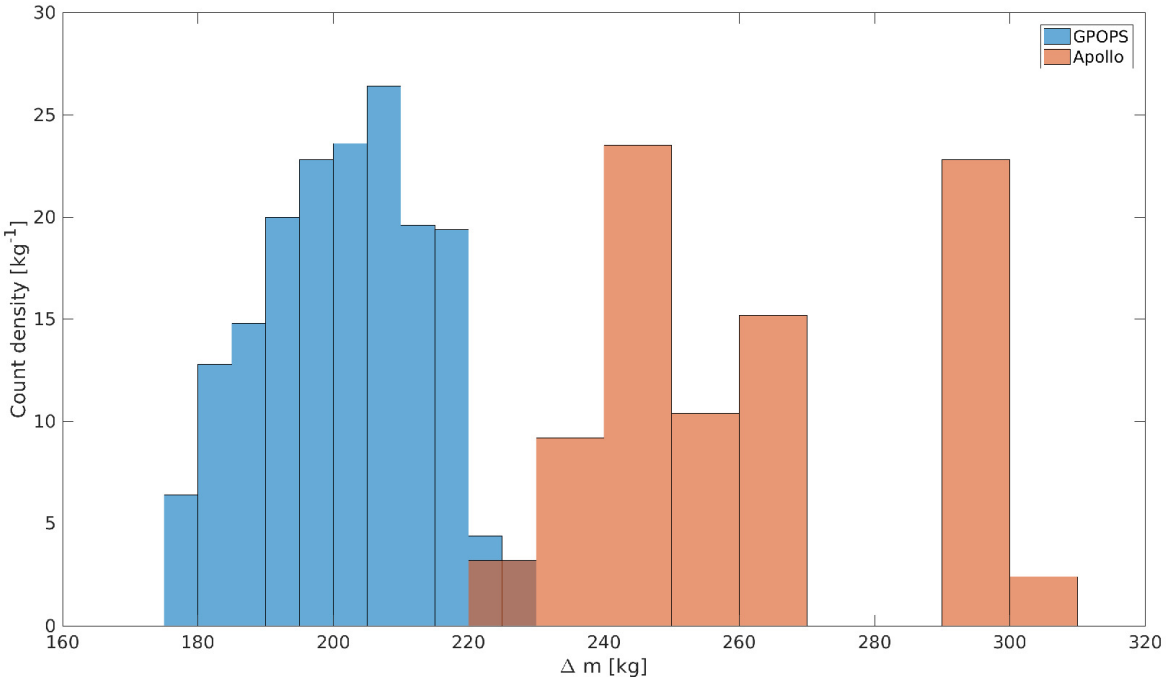


Figure 5.6: Mass consumption histogram comparison between Apollo guidance and simulated open-loop optimal trajectories.

value only when the commanded thrust is larger than the threshold, so that all the available thrust is used, as long as the excess is still small enough not to lead to failures, which happens for large negative values of initial vertical velocity. On the other hand, it gets ever farther from optimality for small initial downward velocities, as t_{go} increases and the commanded thrust is well within its bounds. These trajectories produce the isolated instances visible on the right-hand side of the plot in Figure 5.6.

From this comparison, one can see that the GPOPS trajectories present a clear advantage in terms of fuel usage as compared to the Apollo guidance. On the other hand, the advantage of using Apollo guidance trajectories as the apprenticeship training dataset is that they could be run in closed-loop in a stochastic environment, and thus provide a better policy approximation for position and velocity accuracy. In the first case, one needs to optimize the NN for robustness against environmental pertubations, whereas in the second case, the metric to improve would be propellant consumption. However, another issue with using Apollo guidance trajectories has to do with the envelope of feasible initial conditions for a given thrust constraint. In fact, the data shown in Figures 5.5 and 5.6 are obtained after pruning away the trajectories for which the Apollo guidance was not able to achieve a safe TD (~ 15% of ICs). Therefore, one can either leave the thrust bounds unchanged and shrink the part of state space for which compatible data can be generated, or increase the thrust bounds to keep the initial conditions constant. As neither of these options is desirable, the optimal trajectories were chosen as the input dataset for the supervised training of the NN, described in the next section.

5.2.2. APPRENTICESHIP LEARNING

Now that a way to generate a proper training dataset has been set up, it is possible to perform the supervised training of the NN by using the Matlab Neural Network Toolbox. The optimal solutions, whose mass consumption is shown in Figure 5.6, consist of a set of 1024 trajectories with 22 nodes

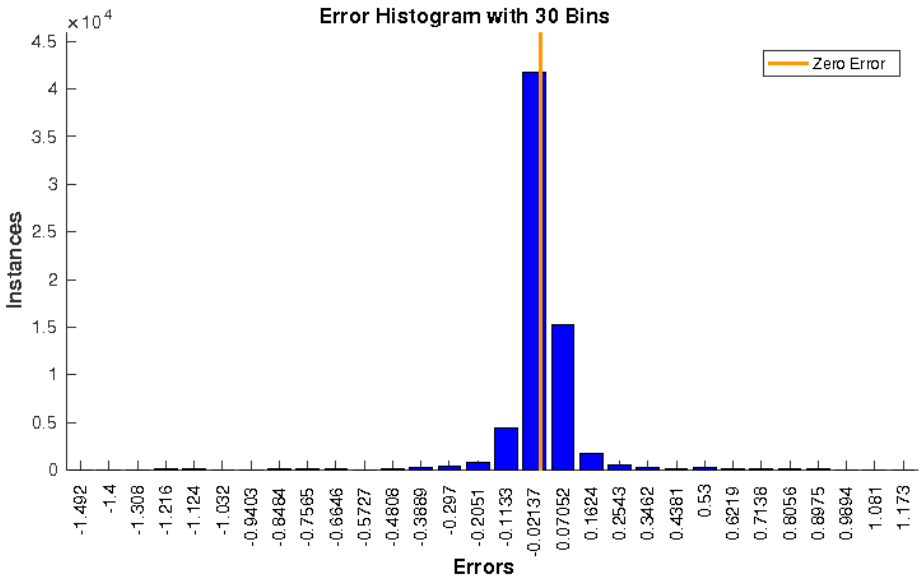


Figure 5.7: Apprenticeship learning error histogram.

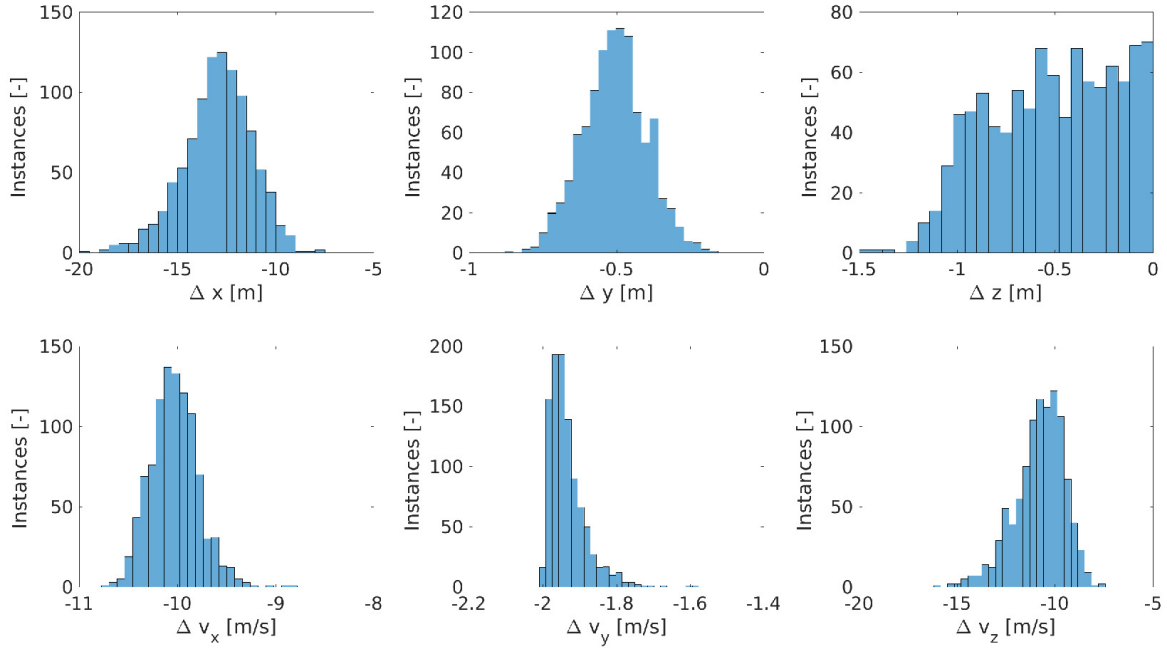
(Gauss points) each, which results in a total of 22528 input-output pairs. The inputs have dimension six (i.e. position and velocity components, the mass is not included as an input to the NN), while the output has dimension 3 (i.e. commanded acceleration vector components). These data are used to train the network by means of the Bayesian regularization algorithm. The resulting error histogram is shown in Figure 5.7 for a network with 70 hidden nodes. One can see that most of the residuals are close to 0, but that there are a few outliers with considerable errors (up to $\sim 1.5\,\mathrm{m/s^2}$).

Figure 5.8 shows the position and velocity errors and propellant consumption at TD as obtained by simulating the so-obtained NN in an uncertain environment. The following perturbations have been included: thrust noise and misalignment, atmospheric forces with a mild wind scenario from random direction and varying gravitational acceleration. The navigation system is assumed perfect in order to clearly isolate the accuracy of the guidance system.

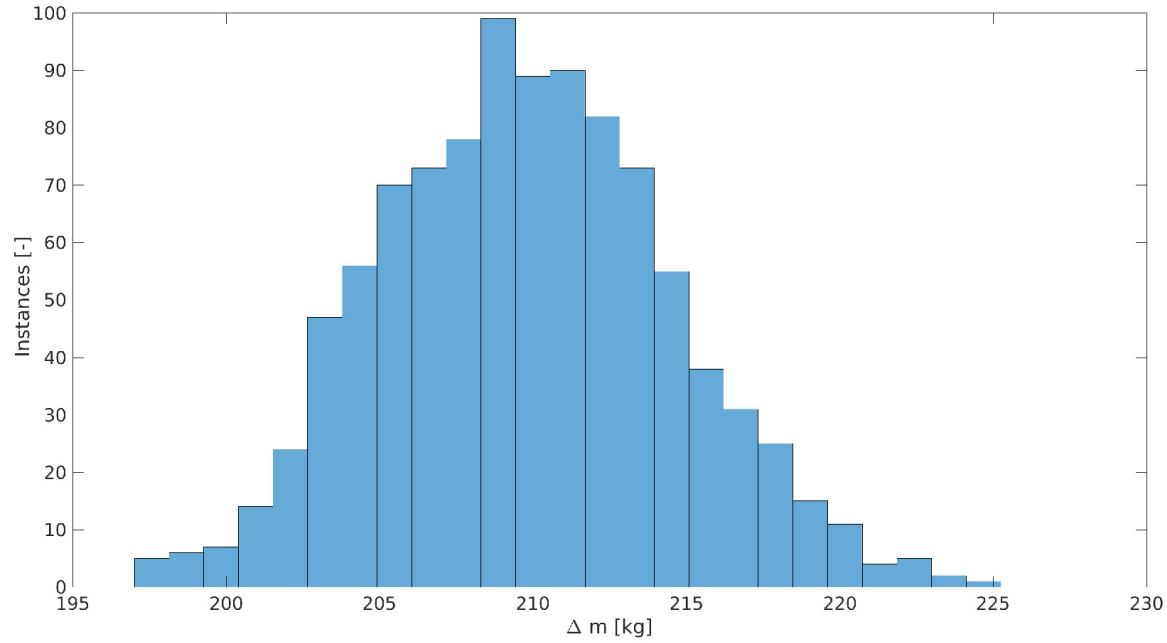
One can see that very large errors are present, the most problematic being the velocity errors, which would result in a hard impact on the surface (bottom plots in Figure 5.8a). This is likely due to both environmental disturbances and lower NN output accuracy in areas of the state space where training samples were not available and the NN approximated the policy by generalization. Environmental noise is per se already detrimental to the performance of a network trained from deterministic data. Besides, if such inaccuracies lead the state outside the training domain, a cascade effect might take place, increasing the final errors. Figure 5.8b shows the fuel consumption distribution. The NN does already perform considerably better than the Apollo guidance in terms of propellant, although this is partly due to residual energy at TD, as can be seen by the large terminal velocities in Figure 5.8a. However, this is expected at this point, as the network has been trained as starting from deterministic data, and these results refer to a simulation in an uncertain environment. As such, the network lacks the necessary robustness to achieve a precise landing, and it thus needs to be further trained within the RL optimization framework, introduced in Section 5.4.

5.3. POLICY OPTIMIZATION

Section 5.3.1 gives a brief overview of the theoretical framework underlying the RL problem, while Section 5.3.2 introduces the difference between direct and indirect policy optimization approaches, and explains the reasons and thought process for the selection of the chosen method.



(a) Position and velocity errors.



(b) Mass consumption.

Figure 5.8: Error histograms of the apprenticeship learning trajectories (in uncertain environment).

5.3.1. INTRODUCTION TO REINFORCEMENT LEARNING

Reinforcement Learning (RL) is a machine learning technique used to have an *agent* learn a specific task by interaction with the surrounding *environment*. When the agent takes *actions*, the state of the environment changes according to a *state transition probability* function (i.e. stochastic envi-

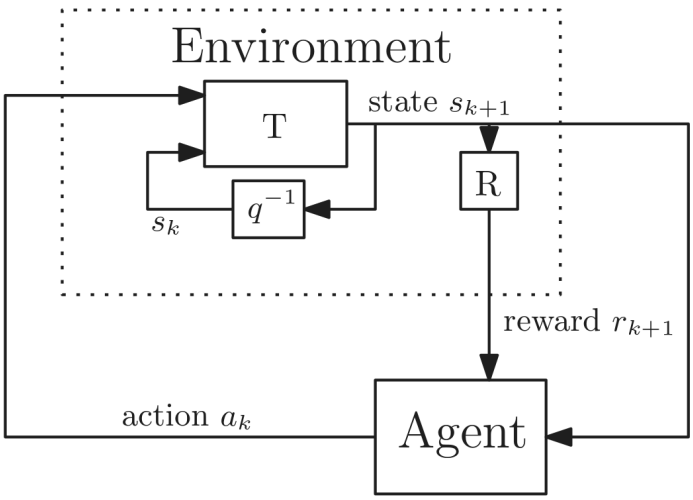


Figure 5.9: Architecture of a RL task [Babuska, 2014].

ronmental model), which might be known or not. Depending on the outcome the action taken led to, the agent receives a *reward* from the environment. The objective of the agent is that of learning a *policy* (i.e. adapting its behavior) that maximizes the discounted sum of the expected rewards, that is, the reward accumulated by the agent at the end of the episode (i.e. when the terminal state is reached).

As discussed by Sutton and Barto [1998], RL is defined as characterizing a *learning problem*, and not as the method used to solve such problem. The fundamental elements of this framework are:

- a set $\mathcal{S}$ of states s ($s \in \mathcal{S}$)
- a set $\mathcal{A}$ of actions a ($a \in \mathcal{A}$)
- a reward function $\mathcal{R}^a_{ss'}$
- a state transition probability $\mathcal{P}^a_{ss'}$
- a policy π

According to Babuska [2014], RL is most commonly set up as a discrete time problem. Therefore, actions and states at time step k will be labeled a_k and s_k, respectively. As can be seen from Figure 5.9, when an action a_k is taken at time k, state s_k transitions into s_{k+1}, according to the probability function $\mathcal{P}^a_{ss'}$, where s, a and s' are compact notations for s_k, a_k and s_{k+1}, respectively. This function (state transition block T in the figure) is defined as the transition probability distribution over the new state s' when action a is taken in state s. Once the environment is in the new state, the agent is given a reward (block R in the figure), dependent on the outcome of the action taken. One must distinguish between an *immediate reward*, which is given to the agent at every step, and a *total reward*, which is the cumulative reward received over a certain time span. The immediate reward is defined by the reward function $\mathcal{R}^a_{ss'}$, which maps each state, or state-action pair, to the set of real numbers, either deterministically or stochastically. It basically expresses how good it is to be in the current state, or to be in the current state and execute a certain action. The objective of the agent is then to adapt its behavior and learn a policy that maximizes the total reward, that is, the *optimal policy*. In general, a policy is a function that maps each state to an action. It can be deterministic — to each state corresponds an action — or stochastic, in which case the policy represents a probability distribution over the state-action space [Babuska, 2014].

REWARD FUNCTION

The immediate reward function $\mathcal{R}^a_{ss'}$ is a known function which depends on the state of the environment, and can only indirectly be influenced by the policy. This is analogous to the cost function of an OCP.

The sum of all the future immediate rewards (total reward) is called *return*, which is thus defined as

$$R_k = r_{k+1} + r_{k+2} + \cdots + r_{k+K} = \sum_{n=0}^{K-1} r_{n+k+1} \tag{5.34}$$

for a finite horizon problem, where K is the number of steps before the goal is reached. A discount rate $0 \leq \gamma < 1$ is used for infinite horizon problems

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots = \sum_{n=0}^{\infty} \gamma^n r_{n+k+1} \tag{5.35}$$

but it is convenient for finite horizon problems too (so-called episodic tasks), for the purpose of mathematically bounding the sum, and also because the discount γ can be seen as a sort of uncertainty about future events. In that case the expression becomes

$$R_k = r_{k+1} + \gamma r_{k+2} + \gamma^2 r_{k+3} + \cdots + \gamma^{K-1} r_{k+K} = \sum_{n=0}^{K-1} r_{n+k+1} \tag{5.36}$$

The objective is then that of maximizing the return. The agent, however, does not know the future rewards in advance, and, consequently, neither the return nor whether its current best policy would actually result in the highest return [Babuska, 2014]. Common methods of solving the RL problem make use of so-called value functions to map from state, or state-action pairs, to the expected return. This will be explained in greater detail in Section 5.3.2.

EXPLORATION VS. EXPLOITATION

As pointed out by Sutton and Barto [1998], two important aspects of RL are the efficient estimation of value, done via methods like value and policy iteration, and the *exploration vs. exploitation dilemma*. In fact, the easiest way to determine the policy is that of choosing the action that maximizes the action-value function Q at the current state. This procedure goes by the name of *greedy policy* [Babuska, 2014]. However, if the agent never explores actions which are sub-optimal in the short-term, it might never find the global optimum of the long-term reward, that is, of the value function itself.

A balance between exploration and exploitation is therefore necessary, with an emphasis on exploration in the case of a time-changing environment. Different exploration techniques are available, the simplest of which is defined as ϵ-greedy exploration. With this method, there is an ϵ chance that a suboptimal action is randomly chosen. Typical values for ϵ could be 0.05 or 0.1, but an adaptive value could be used as well.

5.3.2. DIRECT VS. INDIRECT POLICY OPTIMIZATION

Gaudet and Furfaro [2014] classify policy optimization methods into two categories, namely *direct* and *indirect* optimization. The latter entails that the policy is optimized by means of the value function, which is an expression of the *expected return*. That is, since the actual return is not known, the expected value of the sum of the discounted rewards is used instead. Direct optimization, on the other hand, does not compute value functions, but rather directly searches for the optimal policy. A method of this kind is used by Gaudet and Furfaro [2014], which represents the basis of the algorithm implemented for this work. In the following sections, the reasons for following this approach will be given.

As the original intention of the author was that of following an indirect optimization strategy, the following sections will also present the general features of this approach, and show a simple application of one such method, which ultimately led to the decision of abandoning this direction.

VALUE FUNCTION

In indirect policy optimization methods, a *value function* is introduced that maps states (state-value function) or state-action pairs (action-value function) to the *expected return*. Assuming an episodic task, a state-value function has the form

$$V^{\pi}(s) = E^{\pi}\{R_k|s_k\} = E^{\pi}\left\{\sum_{n=0}^{K-1}\gamma^n r_{n+k+1}|s_k\right\} \tag{5.37}$$

while an action-value function is

$$Q^{\pi}(s,a) = E^{\pi}\{R_k|s_k,a_k\} = E^{\pi}\left\{\sum_{n=0}^{K-1}\gamma^n r_{n+k+1}|s_k,a_k\right\} \tag{5.38}$$

The former expresses how good it is to be in state s and follow policy π, while the latter expresses how good it is to be in state s, performing action a, and subsequently following policy π.

The task is now that of finding the policy that yields the optimal value function V^* or Q^*, which is expressed by the *Bellman optimality equation*

$$V^*(s) = \max_a \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma V^*(s')\right] \tag{5.39}$$

for state-value functions, or

$$Q^*(s,a) = \sum_{s'} \mathcal{P}^a_{ss'}\left[\mathcal{R}^a_{ss'} + \gamma \max_{a'} Q^*(s',a')\right] \tag{5.40}$$

for action-value functions. When an explicit model of the environment is available (model-based RL), Equations (5.39) and (5.40) can be solved recursively by means of e.g. *value iteration* and *policy iteration*. As a stochastic state transition function is not available for the problem at hand, model-based methods will not be considered for this work.

In the case of model-free RL, the value function is optimized online (that is, by simulation in a realistic environment), most commonly by means of temporal differences methods (e.g. Sarsa and Q-learning), possibly in combination with actor-critic techniques [Babuska, 2014]. These methods are very appealing, in that they update the value function and the policy at every step of the simulation, without waiting for the end of the episode, or set of episodes. They also often make use of eligibility traces, which allow for a selective adjustment of value function and policy [Sutton and Barto, 1998]. For these reasons, it is the author's belief that a better optimization result could be achieved by means of these methods. Their implentatability has been investigated and tested. The next section shows the outcome of this implementation test, as applied to the mountain car problem.

MOUNTAIN CAR PROBLEM

The mountain-car problem, a typical test case in the RL community, is the setup used for the above-mentioned investigation. The problem consists in a car stuck in a trough. The geometry of the problem is shown in Figure 5.10. The state space is two-dimensional and continuous:

$$\mathcal{S} = \{x \in [-1.2, 0.6], v \in [-0.07, 0.07]\} \tag{5.41}$$

The action space is monodimensional and discrete:

$$\mathcal{A} = \{a \in \{-1, 0, 1\}\} \tag{5.42}$$

The objective is for the car to reach the top of the hill on the right-hand side, that is, $x = 0.6$. However, in order to achieve this state, simply taking the action $a = 1$ at every step is not sufficient, as the agent needs to first build up momentum. The left end of the position space is an inelastic wall, which restores velocity to $v = 0$ whenever hit. The agent receives a reward $r = -1$ for each step but the final one (i.e. when the goal is reached), for which a reward $r = 0$ is given. State propagation is defined as follows in discrete time:

$$v_{k+1} = v_k + 0.001 a_k - 0.0025 \cos(3x) \tag{5.43}$$

$$x_{k+1} = x_k + v_k \tag{5.44}$$

Initial conditions are randomized, for better generalization of the optimized policy.

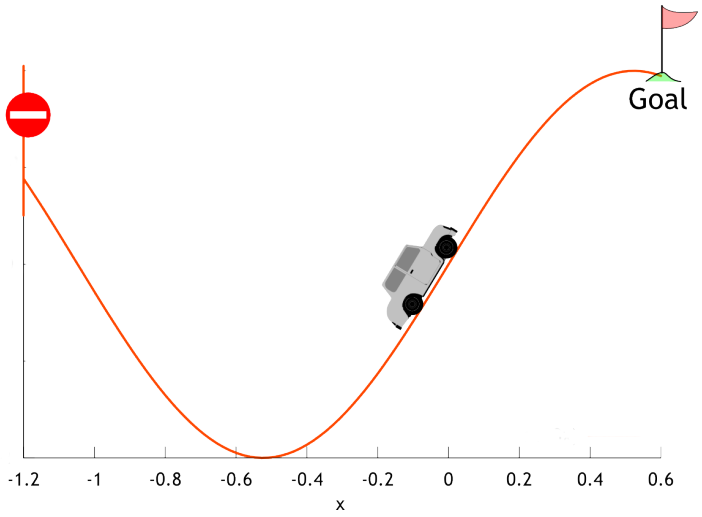


Figure 5.10: Mountain car problem geometry.

SARSA WITH FUNCTION APPROXIMATORS

Sarsa is arguably the most straightforward implementation of value function-based, model-free policy optimization for RL. The simplest and most classical implementation of this method makes use of lookup-tables for storing the value function. This, however, requires the state and action spaces to be discrete, or discretizable efficiently. This limits the application of this method to very small problems, which are rarely encountered in real life. One of these is the mountain car problem, whose state space, although continuous, is small enough to be discretized efficiently with acceptable resolutions. The purpose of this investigation, however, is that of testing the applicability of this method to the planetary landing problem, for which a tabular solution is not viable, for reasons given in the next section.

An alternative, more powerful approach, makes use of function approximators. Specifically, temporal difference methods, such as Sarsa, are known to work well with linear approximators, such as Radial Basis Functions (RBFs). Analogous tools can be used as well, such as tile coding, which uses binary features, as opposed to the continuous-valued RBFs, which will be used for the following verification.

First, a solver for the prediction problem is implemented. This does not involving learning the control policy, but rather simply estimating the value function, approximated by means of the RBF network, for a given control policy. The chosen benchmark policy is expressed as follows

$$\pi_{BM} : \begin{cases} a_k = 1 \text{ if } v_k \geq 0 \\ a_k = -1 \text{ if } v_k < 0 \end{cases} \tag{5.45}$$

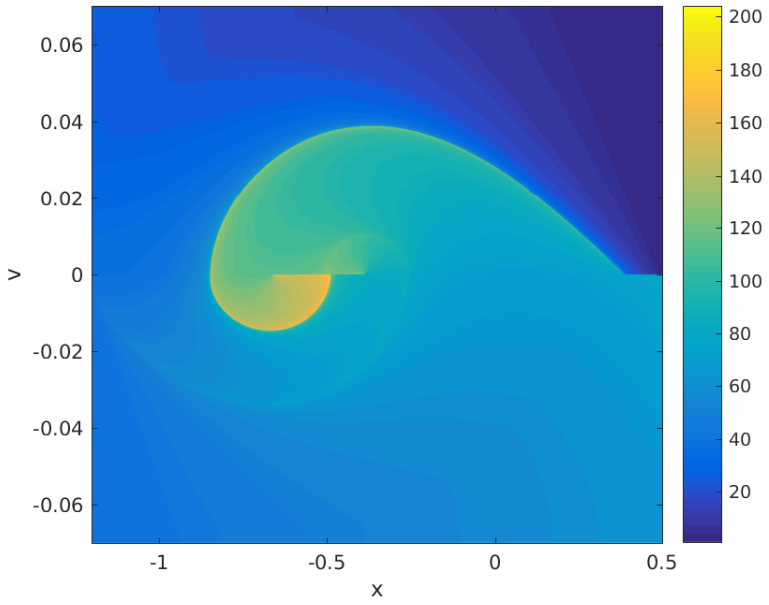


Figure 5.11: True value function for the benchmark policy π_{BM}.

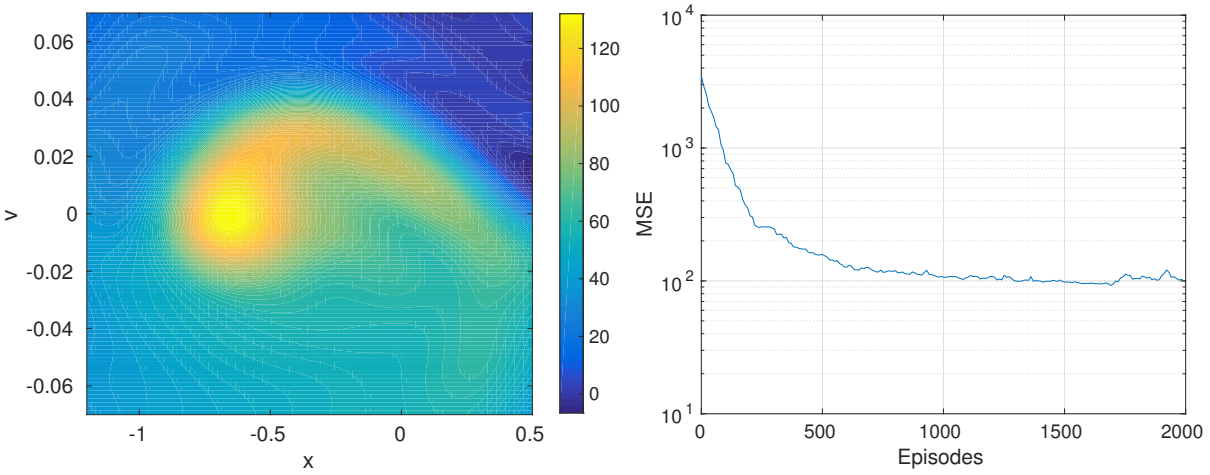
This policy is close to the optimal one, and always guarantees episode termination. Given the deterministic state transition of Eq. (5.44), one can compute the value function over the state space domain for the given policy. The true value function is shown in Figure 5.11. Given the reward function mentioned above, the value function represents the number of steps it takes the agent to reach the final state, starting from the corresponding initial state and following policy π_{BM}.

Now, this value function is estimated by using temporal differences, which is the principle on which Sarsa operates. The pseudo-code of the TD(λ) prediction algorithm is given in Algorithm 1. First, the Function Approximator (FA) parameter vector (an RBF network, in this case) is initialized, together with the eligibility traces, which express which RBF features are most eligible for learning, based on how recently, and how much, they have been activated. At the first step of each epoch, the initial state is generated, and its corresponding value computed, based on the current FA parameter vector. At each step, the action is given by the benchmark policy π_{BM}. In the eligibility traces update step (line 8), γ is the discount factor mentioned in Eq. (5.34), λ is the traces decay factor (from which the algorithm takes part of its name), while the last term is the FA output derivative with respect to its parameters. For a linear FA, this corresponds to its features vector. The state is then propagated, and the value at the new state computed. This enables the computation of the temporal difference as in line 11 of Algorithm 1, and, ultimately, the update of the parameter vector (where α is a step-size, or learning rate). The process is repeated until the terminal state, and then again for future episodes, until convergence is achieved or the maximum number of iterations reached.

The results obtained by running the algorithm for the mountain car task are shown in Figure 5.12. The left plot shows the estimated value function after 2000 episodes, while the right-hand plot shows the evolution of the MSE with respect to the true value function over the episodes. The RBF makes use of an 8×8 grid of features. The RL parameters have been set to $\alpha = 0.1$, $\lambda = 0.9$ and $\gamma = 1$. It is clear from Figure 5.12 that the RBF network is approximating the correct value function. Also, from the MSE history, one can see that the error decreases significantly over the episodes, but the steady-state MSE value is still relatively high. However, this is due to the resolution of the chosen RBF features. Increasing the feature grid resolution greatly increases the function fit. To a lesser extent, the choice of the learning rate α also plays a role, as larger values allow for faster

Algorithm 1 Pseudo-code of the TD(λ) prediction algorithm.

1: $\mathbf{w}_0 = rand()$ ▷ Initialize parameter vector
2: $\mathbf{e}_0 = \mathbf{0}$ ▷ Initialize eligibility traces vector
3: **repeat** for each episode
4: $\quad \mathbf{x}_0 = rand()$ ▷ Initialize state vector
5: $\quad V(\mathbf{x}_0) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_0)$ ▷ Compute state-value
6: $\quad$ **repeat** for each step
7: $\quad\quad a_k = \pi_{BM}(\mathbf{x}_k)$ ▷ Select action
8: $\quad\quad \mathbf{e}_k = \gamma\lambda\mathbf{e}_{k-1} + \nabla_{\mathbf{w}} V(\mathbf{x}_k)$ ▷ Update eligibility traces
9: $\quad\quad [\mathbf{x}_{k+1}, r_{k+1}] = \text{EoM}(\mathbf{x}_k, a_k)$ ▷ Propagate: new state vector and reward
10: $\quad\quad V(\mathbf{x}_{k+1}) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_{k+1})$ ▷ Compute state-value
11: $\quad\quad \delta_k = r_{k+1} + \gamma V(\mathbf{x}_{k+1}) - V(\mathbf{x}_k)$ ▷ Compute temporal difference
12: $\quad\quad \mathbf{w}_{k+1} = \mathbf{w}_k + \alpha\delta_k\mathbf{e}_k$ ▷ Update parameter vector
13: $\quad$ **until** terminal state
14: **until** convergence



(a) Estimated value function for the benchmark policy π_{BM}.

(b) Evolution of the MSE between true and estimated value over the episodes.

Figure 5.12: Results obtained for the prediction problem with the TD(λ) algorithm.

convergence, but result in larger and more unstable steady-state errors (as can be seen in the figure from the fluctuations in MSE after episode ~ 1700). The effect of the other parameters has not been explicitly investigated, given the purpose of this verification.

Once the prediction problem has been verified, the algorithm is extended to include its learning component. The pseudo-code is given in Algorithm 2, as taken from Sutton and Barto [1998] with adaptations for non-binary features from Främling [2007]. As compared to the prediction problem, this control algorithm has two major differences. Most importantly, the FA does not model the state-value function anymore, but rather an action-value (or state-action-value) function. As can be seen in lines 5-8 and 18-21 of Algorithm 2, for a given state, the value is computed for each of the actions in the action space $\mathcal{A}$. The action resulting in the highest value is then chosen. Besides, not all eligibility traces are updated equally. Those corresponding to the action which has been selected are updated as before, while those corresponding to the other actions are reinitialized to 0.

Exploration is implemented by means of the optimistic initial value strategy. The action-value function is initialized to a value larger than its true value (e.g. for the case at hand, $Q(\mathbf{x}, a) = 0$ for all $\mathbf{x} \in \mathcal{S}$ and $a \in \mathcal{A}$, given the parameter vector initialization in line 1). This makes it such that every

time a state is visited, the action-value of taking the chosen action in that state will decrease (the total reward is always negative for the mountain car problem). As such, the agent will opt for one of the other actions at its next visit of the same state, thus enabling exploration of the state-action space.

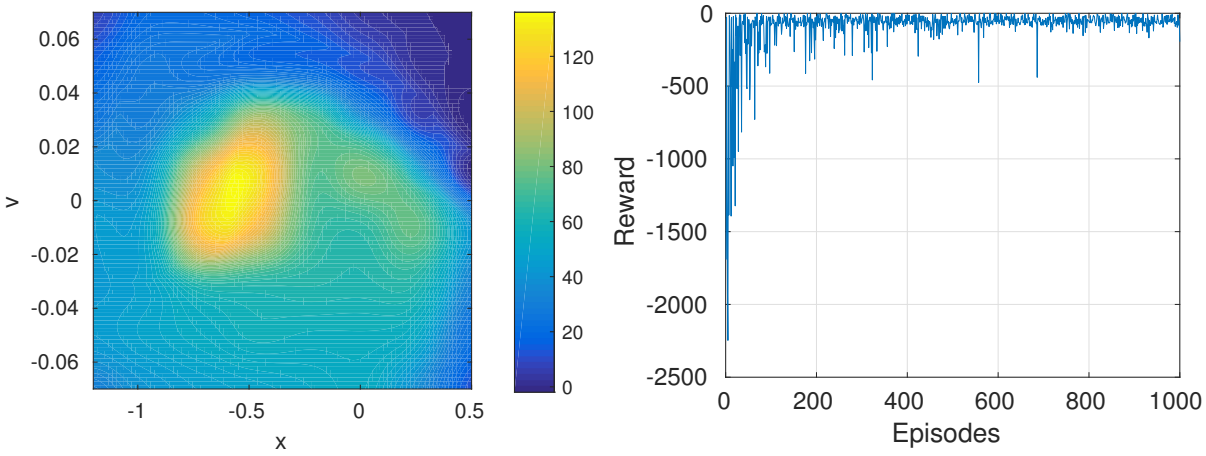
Figure 5.13 shows the results of the learning problem. The optimized action-value function shows a similar pattern to the value function obtained for the prediction problem, since the benchmark policy π_{BM} was close to the optimal one. The discrepancies between the two plots are due short optimization times in the learning problem. Allowing for a larger number of iterations would further improve the fit of the action-value function towards the true one. From Figure 5.13b one can clearly see the reward increase until it reaches its steady-state value, whose noise is due to the randomization of the ICs. The results obtained are quite in line with studies on Sarsa combined with RBFs, such as Främling [2007]. As such, the implementation can be considered successfull.

Algorithm 2 Pseudo-code of the gradient-descent SARSA(λ) algorithm.

$\mathbf{w}_0 = \mathbf{0}$ ▷ Initialize parameter vector
$\mathbf{e}_0 = \mathbf{0}$ ▷ Initialize eligibility traces vector
repeat for each episode
  $\mathbf{x}_0 = rand()$ ▷ Initialize state vector
  for $a_i \in \mathcal{A}$ **do**
    $Q(\mathbf{x}_0, a_i) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_0, a_i)$ ▷ Compute action-values
  end for
  $a_0 = \arg\max_a Q(\mathbf{x}_0, a)$ ▷ Select action with highest value
  repeat for each step
    for $a_i \in \mathcal{A}$ **do**
      if $a_i = a_k$ **then**
        $\mathbf{e}_k(\mathbf{x}_k, a_i) = \gamma\lambda\mathbf{e}_{k-1} + \nabla_{\mathbf{w}} Q(\mathbf{x}_k, a_i)$ ▷ Update eligibility traces (selected action)
      else
        $\mathbf{e}_k(\mathbf{x}_k, a_i) = \mathbf{0}$ ▷ Update eligibility traces (other actions)
      end if
    end for
    $[\mathbf{x}_{k+1}, r_{k+1}] = \text{EoM}(\mathbf{x}_k, a_k)$ ▷ Propagate: new state vector and reward
    for $a_i \in \mathcal{A}$ **do**
      $Q(\mathbf{x}_{k+1}, a_i) = \mathbf{w} \cdot \boldsymbol{\phi}(\mathbf{x}_{k+1}, a_i)$ ▷ Compute action-values
    end for
    $a_{k+1} = \arg\max_a Q(\mathbf{x}_{k+1}, a)$ ▷ Select action with highest value
    $\delta_k = r_{k+1} + \gamma Q(\mathbf{x}_{k+1}, a_{k+1}) - Q(\mathbf{x}_k, a_k)$ ▷ Compute temporal difference
    $\mathbf{w}_{k+1} = \mathbf{w}_k + \alpha\delta_k\mathbf{e}_k$ ▷ Update parameter vector
  until terminal state
until convergence

CONSIDERATIONS

Ultimately, to perform the entire RL task, one will always be making use of action values, as opposed to state values, which can only be used stand-alone for prediction problems. Hence, the need for a discrete (or discretizable) action space if a Sarsa framework is to be employed. This can be implemented in different ways. For instance, one could train as many networks as there are actions. Each network would then learn the value of a specific action as a function of the state. On the other hand, one could have a single network with as many output channels as there are actions, each channel returning the corresponding action value. One more option would be to have a network with



(a) Optimized value function.

(b) Evolution of the reward over the episodes.

Figure 5.13: Optimized value function and convergence plots for the learning problem with the Sarsa(λ) algorithm.

$N(\mathcal{S}) + N(\mathcal{A})$ input channels and one output channel returning the value of the input (s, a) pair. This would allow for a continuous action space in that the action-value for a given state could be obtained for an arbitrary value of the action. However, when actually choosing the action, one still needs to discretize the action space, compute the action-value for each discretization point, and then choose the one corresponding to the highest action-value. This is unhandy for the problem at hand, as a sufficiently thick discretization of the 3-dimensional action space would result in an overly large output for the function approximator.

Even assuming one can discretize the action space, the same problem shows up for the state space as well. In fact, as said before, linear approximators such as RBFs require the definition of feature points, which would again be inefficient for the state space of the landing problem. Therefore, a non-linear FA, such as a NN, which does not need the definition of features, is required. Unfortunately, it was not possible to successfully optimize a NN with Sarsa, not even for the prediction problem with discrete actions only. In fact, this method, as well as the other traditional RL optimization strategies, are known to diverge when applied to non-linear approximators. Although a method has been developed by Maei et al. [2009] that solves the problem of temporal-difference learning with non-linear function approximators, not much literature is present on practical applications.

Ultimately, since a method is needed which requires neither state nor action space discretization, the only remaining option are actor-critic methods. Actor-critic methods implement two function approximator, one for estimating the value function (critic), another for learning the optimal policy (actor). The critic learns from the reward received from the environment, just as it happens in the prediction problem. The critic feeds then this information back to the actor, which updates itself such as to maximize the state-value estimated by the critic.

To sum up the points made in this section, one can break down the considerations as follows. The first issue is the large number of points required to achieve a good resolution discretization of the state space. This imposes the use of non-linear approximators, which, in turn, do not work properly with common direct optimization methods. Some methods have been developed, but they have not been applied to complex, real world problems, to the best of the author's knowledge. The second issue is the action space. A discretization of this space is even more problematic than for the state space, as the whole action space would have to be search at every time step. These aspects ultimately call for the use of actor-critic methods in combination with non-linear FAs.

Given the lack of previous literature on the combination of these methods, the complexity of

their implementation, and the limited time available for the completion of this project, the pursuit of an indirect optimization strategy was abandoned, and a direct method was chosen instead.

5.4. ALGORITHM IMPLEMENTATION

In this section, the policy optimization algorithm, taken from Gaudet and Furfaro [2014], is explained. This corresponds to the policy optimization block in the architecture in Figure 5.1. The input to this loop is the NN weights vector as obtained from the supervised training phase. At the first loop of the iteration, a grid of ICs is created, which consists in two bins per state component with minimum and maximum values equal to

$$\mathbf{x}_{min} = \mathbf{x}_0 - 3 \begin{pmatrix} \boldsymbol{\sigma}_r^{IC} \\ \boldsymbol{\sigma}_v^{IC} \end{pmatrix} \tag{5.46}$$

$$\mathbf{x}_{max} = \mathbf{x}_0 + 3 \begin{pmatrix} \boldsymbol{\sigma}_r^{IC} \\ \boldsymbol{\sigma}_v^{IC} \end{pmatrix} \tag{5.47}$$

which results in a set of 32 ICs. The nominal trajectory has been included as well, for a total of 33 trajectories, which will be simulated at each loop of the optimization procedure. As shown in Figure 5.14, the initial weights are first simulated for these ICs and an initial cumulative reward is computed as the sum of the single rewards over the set of simulated trajectories

$$R = \sum_i^N r_i(\mathbf{r}_f, \mathbf{v}_f, m_f) \tag{5.48}$$

where $N = 33$ in this case, and the reward function's general formulation is

$$r_i(\mathbf{r}_f, \mathbf{v}_f, m_f) = -k_r f(\mathbf{r}_f) - k_v f(\mathbf{v}_f) - k_m f(m_f) \tag{5.49}$$

The different components of the reward function will be presented in Section 6.1.2. The initial cumulative reward sets the beginning of the optimization loop. At every epoch, the NN weights are perturbed according to a multiplicative Gaussian noise of unit mean and standard deviation equal to a scale factor w_{scale}.

$$\mathbf{w}_{k+1} = \mathbf{w}_k N(1, w_{scale}^{(k)}) \tag{5.50}$$

A new set of trajectories is then simulated with the so-obtained weight vector. If the new cumulative reward is higher then the current best, the new weights are retained and the value of the best cumulative reward updated. Otherwise, the latter is left unchanged, and the weights are reverted back to the original ones.

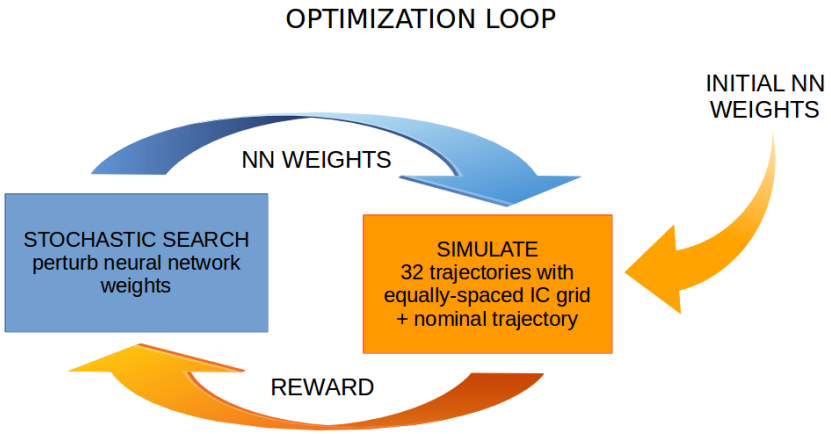


Figure 5.14: Policy optimization loop.

A few additional parameters are defined to decrease the value of w_{scale} when no more improvements can be made with the current value. The variable n_{const} represents the number of epochs without any improvement in cumulative reward. When this exceeds a threshold n_{max}, the scale factor is decreased proportionally to $w_{decrease}$ at every subsequent step with no improvement.

$$w_{scale}^{(k+1)} = w_{decrease} w_{scale}^{(k)} \quad \text{while } n_{const} > n_{max} \tag{5.51}$$

When an improvement occurs again, n_{const} is set back to zero, and the scale factor w_{scale} kept constantly equal to its latest value.

The pseudo-code in Algorithm 3 illustrates these steps. One can see the `simBatch` function, which simulates a batch of trajectories with initial conditions X_{init} by using the network with weight vector **w**, and returns the cumulative reward R.

Algorithm 3 Pseudo-code of the direct stochastic search optimization method for the RL problem.

generate IC grid X_{init}
$R = \texttt{simBatch}(X_{init}, w_0)$
$R_{max} = R$
$w = w_0$
$n_{const} = 0$
repeat
    $w_{old} = w$
    $w = w \cdot N(1, w_{scale})$
    if $n_{const} > n_{max}$ **then**
        $w_{scale} = w_{decrease} \cdot w_{scale}$
    end if
    $R = 0$
    $R = \texttt{simBatch}(X_{init}, w)$
    if $R > R_{max}$ **then**
        $R_{max} = R$
        $n_{const} = 0$
    else
        $w = w_{old}$
        $n_{const} = n_{const} + 1$
    end if
until convergence

5.4.1. VERIFICATION

To verify the correct implementation of this algorithm, it has first been applied to the mountain car problem, introduced in Section 5.3.2. For this task, the neural network weights are initialized randomly, which is fast enough for a problem like the mountain car, and allows for comparison with the Sarsa algorithm. In this case, the action space is treated as continuous, and the NN output is saturated between -1 and 1. A grid of 64 ICs has been used, with a network with 5 hidden nodes.

Figure 5.15 shows the control policy as a function of the state, before and after the optimization. The color shows the saturated output, which is used for state propagation. The level lines, on the other hand, show the actual NN output, which can be seen to exceed the saturation values. It can also be seen from Figure 5.16, which shows the mean reward over the epochs, that the algorithm achieves convergence. However, when comparing the policy shown in Figure 5.15 to the one obtained with the Sarsa algorithm, shown in Figure 5.17, one can see the lack in detail that modeling the policy with the NN entails. Finally, Figure 5.18 shows a comparison between reward histories as

obtained from randomized initial conditions with the optimized NN policy (Figure 5.18a) and during the last 500 epochs (out of 2000) of a Sarsa optimization (Figure 5.18b). It can be seen that the variance of the reward series is much lower for the Sarsa policy than for the NN one.

One can therefore conclude that the direct stochastic search algorithm implementation is verified, as the algorithm can achieve a good level of convergence for the mountain car problem. However, from this verification it also emerged that, at least for this task, a value function method is computationally lighter and more accurate in modeling the policy. One needs to consider that this is run with a linear FA, which should allow for better computational speed, but should limit accuracy. However, as the policy in Figure 5.17 is obtained from the differences between action-values as a function of state, a good resolution can still be achieved. Finally, the mountain car task is a very simple benchmark scenario. As such, the performance of the algorithm for this problem is not fully representative of its capabilities as applied to other, more complex tasks. However, it is believed to be good enough for a first verification of the correct implementation and functioning of this method. As such, it is now possible to move on to the application of the algorithm to the real problem object of this work, that is, the constrained PDG problem.

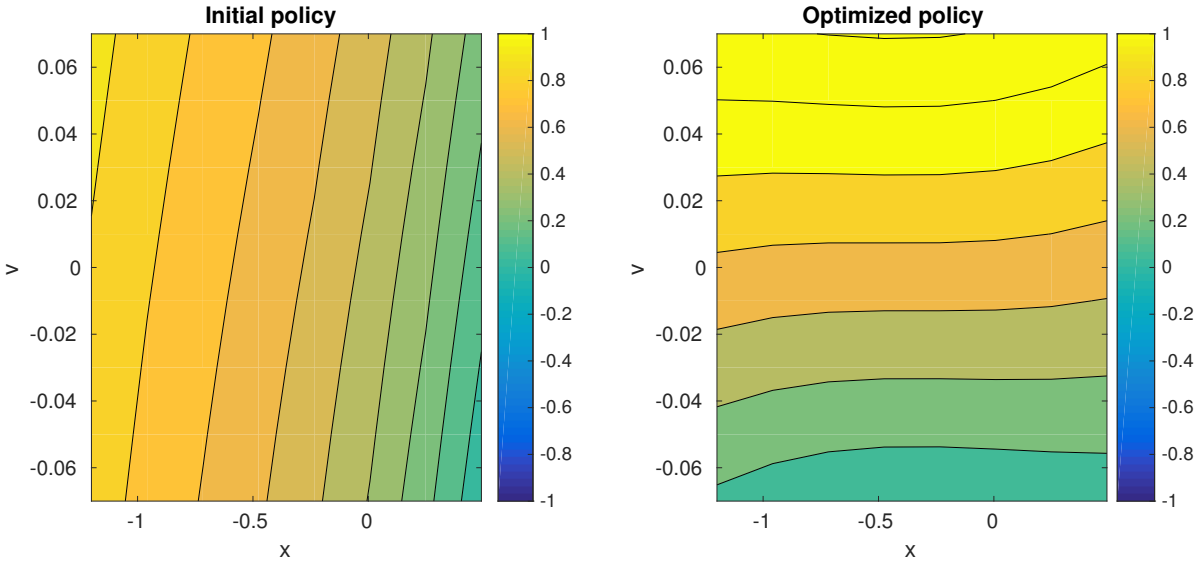


Figure 5.15: Control policy as a function of state, obtained with the direct stochastic search algorithm.

5.4.2. SIMPLIFIED SIMULATOR

The simulator presented in Chapter 4 can run one trajectory in approximately 3 s. Given 33 trajectories are to be simulated at each optimization loop, this would mean that 5000 epochs would take about six days. Gaudet and Furfaro [2014] can achieve convergence within that amount of iterations. However, larger networks are used for the problem at hand, which are expected to require a much longer optimization time. As such, a computational solution is necessary. Therefore, a simplified version of the simulator presented in Chapter 4 is used for the simulation of the small batch of trajectories that takes place during the optimization loop. The basic structure of the simulator was preserved, but some important simplifications were made.

First, the measurement model as presented in Section 4.11 is removed. Instead, the measured state is obtained by perturbing the real state with a multiplicative Gaussian noise

$$\mathbf{r}_{meas} = \mathbf{r}_{real} N(1, \sigma_r) \tag{5.52}$$

$$\mathbf{v}_{meas} = \mathbf{v}_{real} N(1, \sigma_v) \tag{5.53}$$

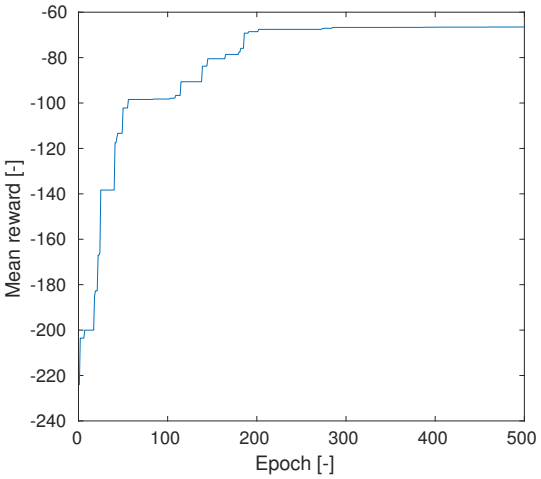


Figure 5.16: Mean reward over epochs, with the direct stochastic search algorithm.

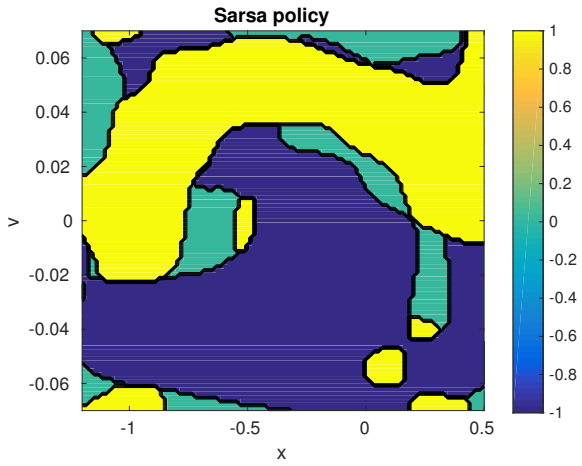
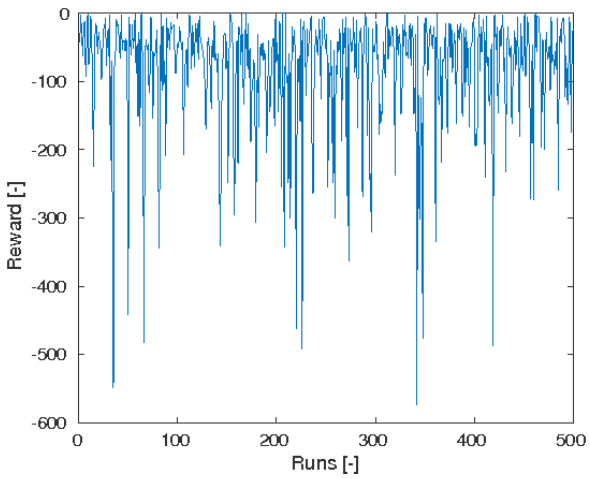
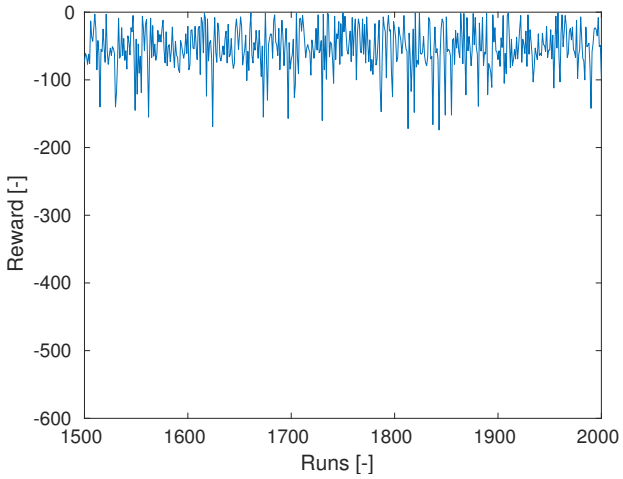


Figure 5.17: Control policy as a function of state, obtained with the Sarsa algorithm.



(a) Direct stochastic search.



(b) Gradient-descent Sarsa(λ).

Figure 5.18: Steady-state reward noise, as obtained for the direct stochastic search and from the Sarsa(λ) algorithm.

For the first part of the optimizations presented here, the standard deviation values are kept constant at $\sigma_r = \sigma_v = 0.01$ from the reference study by Gaudet and Furfaro [2014]. A modified measurement model for the training environment will be implemented for the second part of this work (Section 7.1.4), based on insight gained during the analysis of the first part of the results. Second, constant gravity is assumed. Also, thrust misalignment is removed, leaving only thrust noise, as far as the thrust force is concerned.

Third, the whole aerodynamic model block is removed and replaced by an environmental force model intended to account for all atmospheric forces. To build a suitable, yet simple model for this, a set of trajectories were simulated using the Apollo guidance algorithm with the set of initial conditions introduced in Section 5.2.1. The atmospheric force profiles are shown in the top plots of Figure 5.19. These were obtained after pruning away the trajectories for which the Apollo guidance was not able to achieve soft landing, as these would result in non-representative force profiles.

By looking at these plots, one can see that, for the downrange and crossrange components, the bounds of the profile distributions are approximately linear, with only a few outliers, particularly at lower altitudes. On the other hand, the vertical direction profiles behave in more non-linear way,

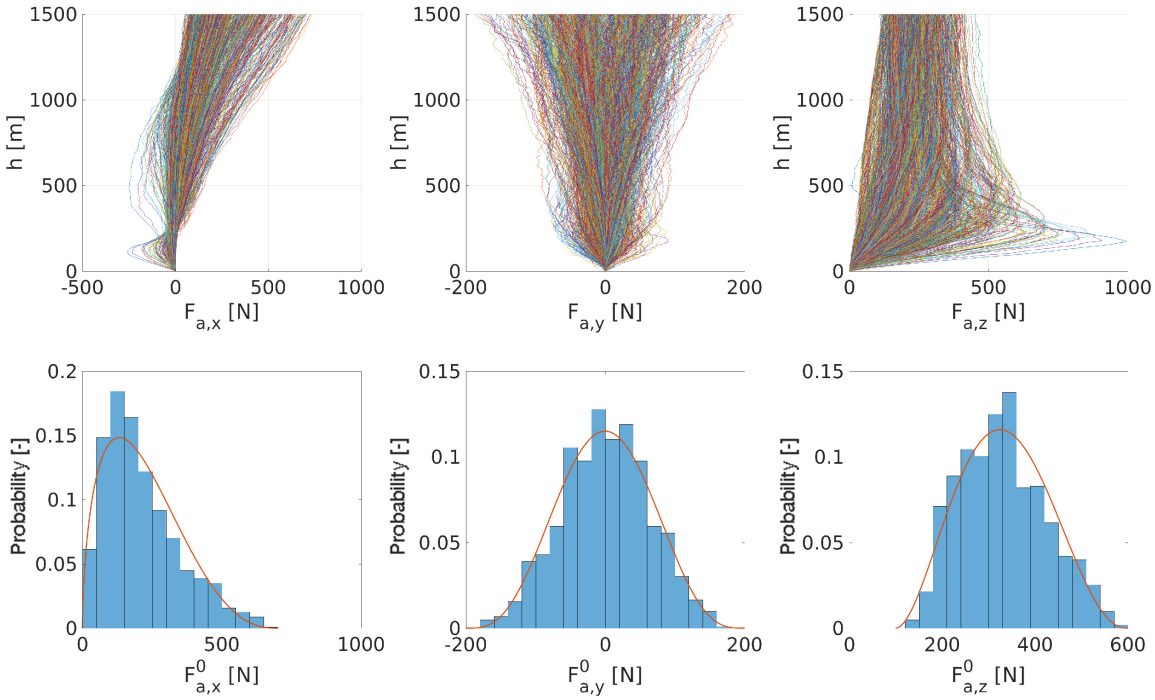


Figure 5.19: Typical atmospheric force profiles and histograms of their initial values fit by a beta distribution.

and show more outliers at low altitude. However, for the intended purpose of providing a coarse approximation of environmental forces for the training environment, it is believed that a model linear with altitude is a reasonable choice for all three components. In fact, it keeps the number of outliers low, while being simple at the same time, serving well the purpose of reducing the computational cost of this lightweight version of the simulation. The model formulation follows.

$$F_{env,x} = \frac{F^0_{env,x}}{h_0} h \tag{5.54}$$

$$F_{env,y} = \frac{F^0_{env,y}}{h_0} h \tag{5.55}$$

$$F_{env,z} = \frac{F^0_{env,z}}{h_0} h \tag{5.56}$$

where $F^0_{env,i}$ is the initial value of the i-th component of the atmospheric force. The histograms at the bottom of Figure 5.19 show the distribution of the components' initial values, including for those trajectories which had been previously pruned away[3]. A beta PDF was then fit to these histograms to allow for the generation of random samples from these distributions. The resulting parameters of the beta distributions are reported in Table 5.2 for the three components of the atmospheric force.

The values in the last two columns are used to map the random numbers from the [0,1] interval in which the beta distribution is defined to the interval of interest. This is done as follows.

$$F^0_{env,i} = F^{0,min}_{env,i} + \texttt{betarnd}(a_i, b_i)\left(F^{0,max}_{env,i} - F^{0,min}_{env,i}\right), \quad \text{for } i = x, y, z \tag{5.57}$$

where `betarnd` is the function that generates random samples from a beta PDF with the given parameters a and b.

[3]All of these initial conditions are expected to be handled by the NN guidance during the RL optimization phase.

Table 5.2: Parameters of the beta PDFs fitting the distributions of initial atmospheric force components, and lower and upper bounds for each component.

	$a[-]$	$b[-]$	$F_{env,i}^{0,min}$[N]	$F_{env,i}^{0,max}$[N]
$F_{env,x}^{0}$	1.5955	3.5563	0	700
$F_{env,y}^{0}$	4.3952	4.4066	-200	200
$F_{env,z}^{0}$	2.9235	3.3666	100	600

By generating the initial values randomly according to these beta PDFs, force profiles will be applied to initial conditions for which they never occured in the simulations used here. In fact, the initial atmospheric force is partly determined by the initial velocity, which is known for given initial conditions. As such, it would be sufficient to generate values for the initial wind force, and add it to the known component of the atmospheric force which is solely due to the S/C velocity. However, randomizing the entire extent of the initial atmospheric force adds robustness, as it is expected to generalize the results of the RL optimization to a larger set of environmental conditions. It also makes for an easier and lighter implementation, which was the main goal of the modifications made to the simulator. These modifications allowed to reduce the simulation time by nearly a factor 10.

5.4.3. COMPARISON TO ORIGINAL ALGORITHM

Although mostly similar to the method proposed by Gaudet and Furfaro [2014], some modification have been done, which proved to yield better results.

In particular, a multiplicative Gaussian perturbation of the weights' scale factor (Eq. (5.50)), instead of an additive one, has been found to provide much better convergence.

Besides, important modifications have been done on the mass component of the reward function, as the original formulation would not allow for an acceptable level of propellant optimality. These will be highlighted in Section 6.1.2.

Similarly to what has been done by Gaudet and Furfaro [2014], the algorithm is expected to be able to handle LS redesignations, that is, in-flight changes in the target point on the surface. This is introduced in Section 7.1.1. On the other hand, the original algorithm is not trained for handling attitude and attitude rate constraints. The feasibility of including these constraints has been investigated in the present study. The results obtained will be presented in Sections 7.2.2 and 7.2.4, respectively.

Finally, one major difference is the dimensionality of the state space. The reference study trains and tests the method in a two-dimensional simulation environment, whereas the present work makes use of a three-dimensional simulator, which allows for a more realistic analysis of the algorithm's capabilities.

6

RESULTS - BASELINE SCENARIO

In this chapter, the first part of the results will be presented. These include the outcome of the optimization procedure and of the corresponding MC simulations for the Baseline (BL) scenario, that is, when no Retargetings (RTs) occur, and neither attitude nor attitude rate are constrained. As such, this chapter will address research questions RQ-1 and RQ-4.1, presented in Section 2.3.

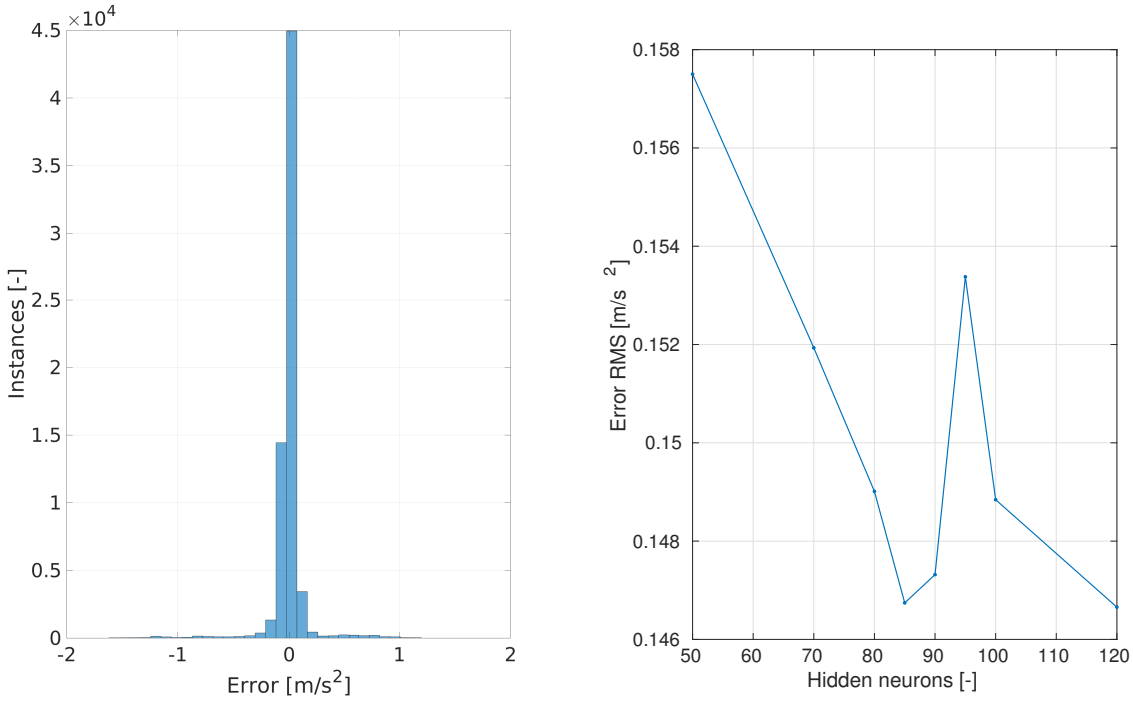
In Section 6.1, the focus will mostly be on the choice of the reward function, which determines the trade-off between position/velocity error and mass consumption (RQ4), and the choice of the number of hidden-layer neurons in the NN, which is also crucial for a good policy model. Section 6.2 focuses more on the overall performance of the optimized NN. In particular, it will be compared against the optimal solution (Section 6.2.1) and the Apollo guidance benchmark (Section 6.2.2). Finally, Section 6.3 discusses a interaction phenomenon between the NN and the navigation system, noticed during tuning of the filter, and Section 6.4 concludes the chapter with a summary and some considerations.

6.1. OPTIMIZATION ALGORITHM TUNING

This section presents the insight gained about the optimization algorithm during the tuning phase, the effect of the different parameters on the optimization results, and the convergence properties of the method. As such, it is mostly directed at answering RQ-4.1. The effect of different numbers of hidden-layer neurons will be discussed in Section 6.1.1, while the choice of the most suitable reward function will be addressed in Section 6.1.2. Finally, additional parameters tuning, such as scale factor and scale factor decay rate, as well as the convergence properties of the algorithm will be discussed in Section 6.1.3.

6.1.1. HIDDEN-LAYER NEURONS

The number of neurons in the NN's hidden layer is an important parameter for a successfull optimization. Gaudet and Furfaro [2014] use a network with 32 hidden layer nodes. However, given the smaller dimensionality of state and action spaces of their problem as compared to the one at hand, such a number of hidden neurons proved to be insufficient to successfully approximate the control policy. Therefore, different values for this parameter were tested, ranging from 50 to 120. For each of these values, the network is first trained offline via supervised learning. Figure 6.1a shows the error distribution after apprenticeship learning of a network with 85 hidden nodes, which is representative of the other networks as well. The distributions obtained for different numbers of hidden nodes will thus be omitted for practical reasons. One can see that the mean of the errors is negligible when compared to their standard deviation. Although there are outliers with errors up to $\sim 1.5\,\mathrm{m/s^2}$, this is expected at this stage, and the training results can be considered satisfatory.



(a) Apprenticeship learning error histogram with 85 hidden neurons.

(b) Error RMS after apprenticeship learning for different hidden-layer neurons numbers.

Figure 6.1: Training errors after apprenticeship learning for different numbers of hidden-layer neurons.

On the other hand, Figure 6.1b shows the Root Mean Square (RMS) error between training data and NN output. The minimum and maximum visible at $H = 85$ and $H = 95$, respectively, are due to stochastic variations during the training process, such as randomization of the initial NN weights, and the random choice of the actual training samples out of the entire training dataset (only 70 % of the training data is actually used, leaving the remainder for validation and testing). By running further training instances, one would find that the trend becomes monotonically decreasing with increasing NN size, given a sufficient number of iterations. However, the RMS value is quite similar between the different networks, with variations lower than 10 % between worst and best cases. In spite of this, the paramater H does make a different during the RL optimization process, as shown in the following.

At least one RL optimization has been carried out for each of these cases. For the values of 70, 80, 85, 90 and 95 hidden neurons, for which the optimization outputs were closer together, three instances have been run. Although a larger number of optimizations would be required for a statistically meaningful comparison, this was not achievable due to time constraints. For each optimization instance, the mean over the set of 33 trajectories has been computed of final position and velocity, fuel consumption and total reward. The plot in Figure 6.2 shows these quantities as a function of H. For the values for which more than one optimization instance was run, the metrics are averaged over the instances as well. From 70 to 95 hidden neurons, the results are quite close to each other, but the 95 neurons network performs better both in terms of position and velocity, and is second in terms of mass only to the 70 neurons case. Such a trade-off of mass optimality for a safer and more accurate TD can be considered acceptable. A sharp degrade in performance takes place for values larger than 95. The 120 neurons network is the only one that performs better in terms of mass consumption. However, its performance in position and velocity is really low, which explains the low propellant use as well: larger residual velocities mean less kinetic energy is removed, hence

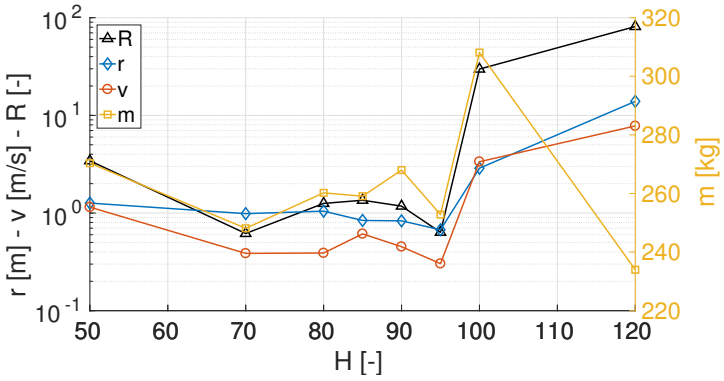


Figure 6.2: Results of the optimization procedure for different numbers H of hidden-layer neurons.

less propellant consumption. While it is safe to assume that values larger than 100 and smaller than 70 are not as good as the other options (although performance seems to be degrading more gently for low values than for high ones), values between 70 and 95 could all yield comparable results from a more extensive analysis, or a clear best might even emerge. Ultimately, one cannot clearly conclude what the best value for this parameter is, but 95 hidden neurons seems to be the best choice one can make with the available data. The reason why larger H values lead to worse performance could be found in the number of maximum iterations n_{max} (see Section 5.4) before the scale factor w_{scale} is decreased. In fact, increasing n_{max} would allow for a more extensive exploration of the parameter space, beneficial for larger H values. However, given the already high computational demand of the algorithm, this option could not be properly tested.

6.1.2. REWARD FUNCTION

Together with the number of hidden nodes in the network, another most important choice is that of the reward function. The general structure of the reward function is given in Eq. (5.49). The right-hand side functions in Eq. (5.49) are defined by Gaudet and Furfaro [2014] as follows

$$f(\mathbf{r}_f) = \|\mathbf{r}_f - \mathbf{r}_t\|^2 \tag{6.1}$$

$$f(\mathbf{v}_f) = \|\mathbf{v}_f - \mathbf{v}_t\|^2 \tag{6.2}$$

$$f(m_f) = m_0 - m_f \tag{6.3}$$

This formulation did not yield satisfactory results, as the choice of a linear propellant reward component results in the convergence of position and velocity at the expense of fuel consumption, which remains at very high values. Therefore, by introducing three additional parameters, the mass component of the reward function is generalized to the following:

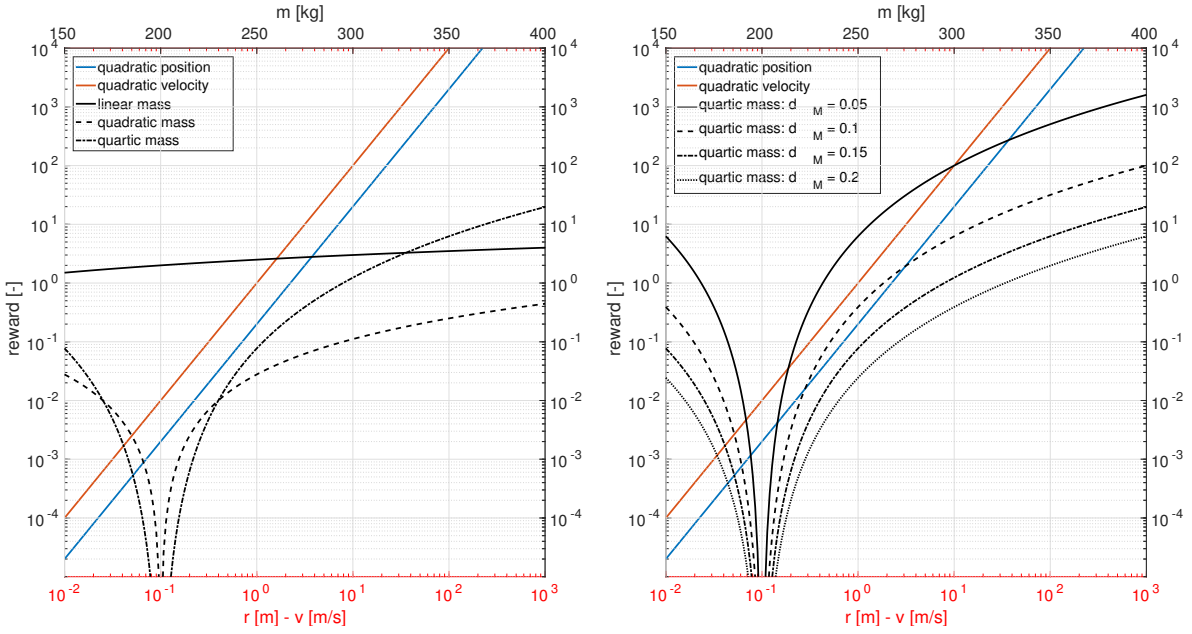
$$f(m_f) = \left(\frac{m_0 - m_f - c_m}{d_m}\right)^{m_m} \tag{6.4}$$

When $c_m = 0\,\text{kg}$, $d_m = 1\,\text{kg}$ and $m_m = 1$, Eq. (6.4) reduces to (6.3). The paramter c_m allows to define a zero-point for referencing fuel consumption rewards. In fact, the mean value of mass consumption will never fall below the optimal one of $\sim 200\,\text{kg}$, as can be seen from Figure 5.6. If it does, it means that not all the kinetic energy is removed, which means that a non-desirable trajectory is being dealt with. Therefore, a linear function passing through the origin fails to effectively translate propellant consumption changes into rewards. Besides, while d_m is a simple multiplier just like the gain k_m in Eq. (5.49) when $m_m = 1$, it becomes useful when increasing the latter, allowing to decide at which fuel consumption value the non-linear function starts growing more rapidly. Therefore, while the c_m can be seen as the minimum mass consumption one can hope for, d_m can be interpreted as the maximum acceptable distance from such value.

Figure 6.3 shows the comparison between different function choices for the mass component of the reward. The negative reward is shown, for an easier interpretation of the plots. It can be seen from Figure 6.3a that the linear function ($k_m = 0.001$, $c_m = 0\,\text{kg}$, $d_m = 1\,\text{kg}$, $m_m = 1$) used by Gaudet and Furfaro [2014] presents very little variation throughtout the interval of interest of the mass consumption, as compared to the position (in red) and velocity (in blue) components of the reward, which vary across several orders of magnitude. These are plotted against a logarithmic axis, as the improvements to be achieved for these metrics are much larger than for fuel consumption, especially at the beginning of the optimization procedure. Figure 6.3a also shows a quadratic and a quartic function ($k_m = 0.01$, $d_m = 30\,\text{kg}$, $c_m = 200\,\text{kg}$, $m_m = 2$ and 4, respectively). One can see that the two curves intersect at $m = c_m + d_m = 230\,\text{kg}$, as expected. Above this value, the quadratic function has a lower steepness than a quartic function. This proved still not sufficient to prevent the trade-off of too much mass for position and velocity accuracy. A quartic function, on the other hand, proved able to successfully limit mass consumption at the early stages of the optimization, while decreasing fast enough for values of $m < c_m + d_m$, thus making room for further improvements in position and velocity. In fact, if the mass reward component remains large once the limit of the stochastic search (given a limited optimization time) has been reached, this will stall convergence preventing position and velocity to be further improved at the cost of small amounts of propellant consumption. As such, a quartic function is chosen.

As for c_m, the most logical choice is the optimal mass consumption value m_i^*, as computed by the OCS for the given trajectory. In fact, for a good trajectory (that is, one that results in soft landing), the propellant consumption cannot fall below this value, which is therefore set as the true zero. As the set of trajectories simulated at each optimization loop has fixed initial conditions, such values can be computed for each of these trajectories under ideal conditions. In a stochastic environment, mass consumption will most likely be higher than in the deterministic one, so that the same value can be used in that case as well.

Finally, different values have been tried out for d_m, whose value has been defined in function of the corresponding m_i^*. Figure 6.3b shows different options. One can see that lower values of this



(a) Mass reward function for different values of m_M.

(b) Mass reward function for different values of d_M.

Figure 6.3: Different mass reward functions. Position and velocity components are shown as well.

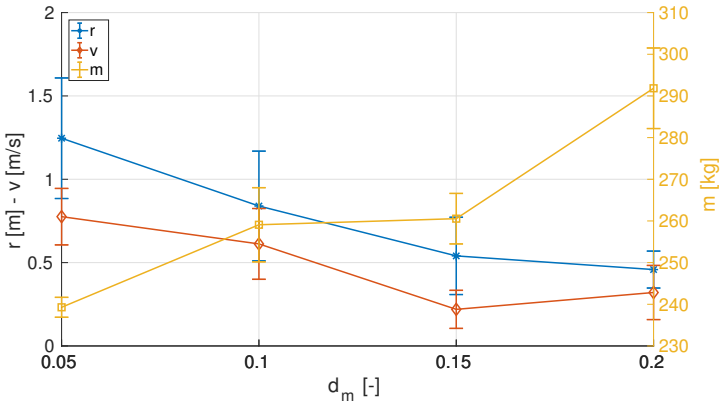
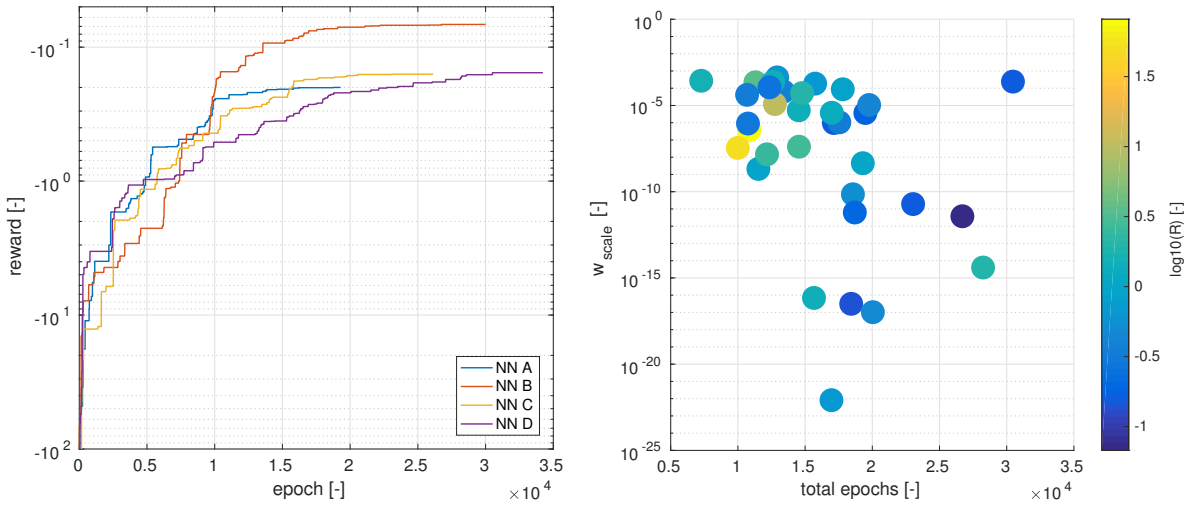


Figure 6.4: Results of the optimization procedure for different values of the d_m parameter.

parameter result in steeper functions with higher values (given the same $k_m = 0.01$). The *relative steepness* between position/velocity and mass curves determines how much mass can be traded for accuracy, while the *rate of change* of this relative steepness determines how hard the constraint on the mass is. The value and value range at which this change happens determines the outer bound of acceptable mass consumption. As the figure shows, for incresing d_m, the value at which the mass function steepness is equal to the position and velocity functions steepness moves toward higher values of m, making for a larger propellant consumption allowance. On the other hand, the constraint becomes softer, as the steepness is not as large for a given value of m. Therefore, given constant k_m, a trade-off arises between constraint stricness and the point at which the constraint is relaxed. To have a harder constraint, one must necessarily have a smaller propellant consumption margin. Vice versa, it is not possible to have a soft constraint with a strict propellant margin. This can become undesirable, as one might want to relax the requirement on the acceptable propellant consumption, to achieve better accuracy, while keeping a well-defined constraint above this value, to prevent excessive increases in mass consumption at the beginning of the optimization. It has been tried to shift the value of c_m upwards from the optimal m_i^*, yet no apparent improvements were noticed in this direction. Also, this could be influenced by means of k_m, which allows for a finer tuning. However, given the stochastic nature of the search algorithm, it proved hard to observe variations of performance from variations in c_m and k_m, as the number of optimizations that could be run was limited. As such, m_m, c_m and k_m were kept equal to the chosen values. For the position and velocity components, $k_r = 0.1$ and $k_v = 1$ (as in the original formulation by Gaudet and Furfaro [2014]), to give more weight to velocity accuracy, as can be seen from the vertical shift in the two curves in Figure 6.3. Finally, only d_m was varied, and its effect investigated, as presented in the following.

For each of the d_m values shown in Figure 6.3b, three, or at times four, optimization instances have been run. The results are shown in Figure 6.4. Despite this being a low number of sample for a proper statistical analysis, standard deviations of these 4-value series have been included in Figure 6.4, to give an idea of how consistent the results are. As expected, mass consumption mean and standard deviation are smallest for $d_m = 0.05 m_i^*$ and largest for $d_m = 0.2 m_i^*$. For $d_m = 0.05 m_i^*$, however, the position and velocity residuals are quite large, with mean values of about 1.2 m and 0.8 m/s, respectively. These are improved to sub-metric position errors and velocity residuals of about 0.6 m/s at the cost of 20 kg more propellant when $d_m = 0.1 m_i^*$. When increasing the parameter further to $0.15 m_i^*$, the improvement is significant. Position and velocity errors are reduced by 0.3 m and 0.4 m/s at almost no cost in terms of propellant consumption. Finally, setting $d_m = 0.2 m_i^*$ is not very convenient, as propellant consumption increases substantially, while providing little or no improvement in position and velocity accuracy. As was the case in the previous section, the available



(a) History of the total reward over the epochs, for four different optimization instances.

(b) Correlation between final scale factor and total number of optimization epochs. The color shows the final total reward.

Figure 6.5: Convergence comparison between different optimization instances, and correlation between final scale factor and total number of optimization epochs.

samples are not sufficient to define a statistically best parameter value. However, in this case the difference between different parameters is more clear, as $d_m = 0.15 m_i^*$ shows the best performance given the data at hand.

6.1.3. ADDITIONAL TUNING AND CONVERGENCE ANALYSIS

A few additional parameters are available for tuning the algorithm. In fact, the weights of the neural network are perturbed according to Eq. (5.50), which requires setting an initial value for the scale factor. This value does not have a clear impact on the end result of the optimization, but rather determines the convergence speed at the beginning of the process, before any adjustments are applied to it. A value of $w^0_{scale} = 0.01$ was found to yield good reward improvements since the early epochs, so this parameter was not investigated further.

On the other hand, the decay rate, which determines the rate at which w_{scale} decreases once n_{max} is reached, is set to $w_{decrease} = 0.99$. A value of $w_{decrease} = 0.9$ had been used by Gaudet and Furfaro [2014], but it has been increased here to prevent the scale factor from decreasing too fast, thus slowing down the subsequent optimization. This makes the convergence process a little slower, but allows for a more extensive search of the parameter space, which is needed for larger networks such as the ones used for this problem.

Finally, the maximum number of iterations before the decay is applied is set to $n_{max} = 1000$ for this first batch of optimizations. This is larger than the value of 500 used by Gaudet and Furfaro [2014], given the larger network sizes, and seems to allow for a sufficient exploration of the parameter space before the scale factor is decreased. This value will then be increased to 2000 later on, when pointing constraints and retargetings are included in the optimization, as a more thorough search will then be needed. Values higher than this will not be tested, as they would lead to excessively long optimization times with the available computational capabilities.

After the number of iterations with no improvement (n_{const}) surpasses n_{max}, the value of w_{scale} will start decreasing. If, even so, no improvement occurs, the scale factor will decrease to a low enough value that no more improvements can be deemed obtainable, as the perturbations of the NN weights become too small. This criterion has been used for stopping the optimization. In Figure 6.5a, which shows the total reward history for different network optimizations (the letter tags given

to the networks refer to Table 6.1), one can see the straight segments which forego improvements in reward, as a consequence of the decrease in scale factor. One can also see that all the profiles end with a straight segment, as w_{scale} has become too small and no more improvements are possible. This means that the number of iterations performed when the optimization is stopped is actually larger than the actual one needed to obtained the final optimization results.

Since a heuristic was used to determine the w_{scale} at which to stop the iterations (usually $w_{scale} < 10^{-10}$), it is not reliable to analyze convergence based on the total number of epochs. Instead, one should look at the epoch number at the last reward improvemnet. Figure 6.5b shows the correlation between this quantity and the corresponding value of the scale factor at that epoch. One can see that, in general, larger numbers of iterations correlate with low final scale factors, and vice versa, as is expected. Besides, the color shows the logarithm of the total reward, which can be seen to be lower for longer optimization times. This correlation between w_{scale} and number of epochs will be more clearly pointed out for the full scenario in Section 7.2.5.

By looking at the x-values in Figure 6.5b, one can see that the number of epochs it takes the algorithm to converge is mostly between 10000 and 30000, but always above 15000 for the best instances (darker blue dots). Besides, one must account for the additional iterations needed to check whether convergence is actually achieved, as was seen before in Figure 6.5a. As one epoch consists of 33 trajectories, whose simulation takes approximately 15 s on an Intel Core i5-5200U Central Processing Unit (CPU) at 2.20 GHz[1], these optimization took approximately between 50 and 150 h to run. This is an extremely long time for a preliminary analysis phase, which makes the algorithm design process slow and tedious. More about convergence speed will be said in Sections 7.1.3 and 7.2.5, once the full scenario will have been introduced.

6.1.4. SUMMARY OF OPTIMIZATION RESULTS

For the choice of the optimization parameters, one would like to have a statistically significant set of data, which allows for a probabilistically best choice. On the other hand, regardless of the availability of these data, when it comes to the selection of the actual best obtained network, one must rather rely on the results of the single instances. Therefore, for the sake of this selection, the results of all the optimizations run by varying the above-mentioned parameters are summarized in Table 6.1. Also, after the selection of the values for d_m and H, an additional set of five optimizations has been run with the chosen values. The results of these optimizations are also included in the table. The first two columns report the parameters values used. The results are first grouped according to the value of the parameter d_m, as the total reward (last column) of these instances is directly comparable, since they use the same reward function parameters. To compare results with different d_m values, one will need to compare the values of the single performance metrics, listed in columns three to five. Columns three and four show the magnitude of the position and velocity errors, respectively. Column five refers to propellant consumption.

According to the guidance requirements, set in Section 2.2.1, the cases which feature position errors below 0.5 m (GR-03) and velocity errors below 0.25 m/s (GR-01 and GR-02) have been highlighted in yellow and green, where the latter perform better than the former in terms of fuel consumption. Instances marked in orange are cases for which one of these two metrics slightly exceeds the required value. This should give an idea of the overall performance consistency when using a random search method. In fact, in many cases, parameter sets identified as better than others in the analyses presented in the previous sections, produced various worse instances, and vice versa.

The green and yellow cases are illustrated in greater detail in Figure 6.6, where the letter tags on the abscissa identify the different instances (as defined in the table), which have been sorted by increasing mass consumption, to ease the comparison. The plot shows mean and standard devia-

[1]Another machine was used as well, mounting an Intel Xeon E5-2683 v3 @ 2.00 GHz. Performance on the two systems was comparable.

Table 6.1: Summary of the optimization results.

$d_m[c_m^*]$	$H[-]$	Δr[m]	Δv[m/s]	Δm[kg]	$R[-]$	CASE
0.05	85	1.11	0.602	237.3	1.50	
		1.66	0.786	242.0	2.42	
		0.969	0.940	238.5	1.96	
0.1	50	1.27	1.15	270.4	3.43	
	70	1.68	0.421	253.2	0.910	
		0.620	0.496	247.6	0.684	
		0.656	0.243	243.6	0.259	
	80	1.15	0.381	257.5	0.672	
		1.18	0.575	274.0	2.77	
		0.811	0.213	249.2	0.325	
	85	0.470	0.371	249.1	0.505	
		0.948	0.695	262.1	1.51	
		1.10	0.771	266.1	2.05	
	90	1.09	0.561	263.8	1.02	
		0.628	0.324	263.6	0.869	
		0.783	0.472	276.6	1.64	
	95	0.627	0.183	247.4	0.288	
		0.905	0.529	270.7	1.42	
		0.491	0.198	240.4	0.200	A
	100	2.92	1.08	300.9	10.8	
		2.83	5.62	315.4	48.8	
	120	13.9	7.79	234.0	81.2	
0.15	85	0.839	0.376	263.8	0.380	
		0.273	0.174	264.4	0.159	C
		0.512	0.220	251.5	0.138	
		0.537	0.108	262.5	0.168	
	95	0.315	0.590	296.1	1.22	
		0.240	0.0874	249.4	0.0677	B
		0.299	0.126	264.0	0.155	D
		0.219	0.413	277.0	0.487	
		0.428	0.312	274.6	0.406	
0.2	85	0.586	0.508	295.6	0.585	
		0.402	0.224	280.9	0.159	E
		0.387	0.230	299.1	0.342	F

tion values over the small set of 33 trajectories which is simulated at every loop of the optimization process. The reported values refer to the last optimization step which resulted in an improvement of the total reward and, therefore, produced the ultimate network weights. It is quite clear that optimization instance B is the best among these, as it features the best position and velocity residuals, together with the second best value of mass consumption. Trading off position and velocity for 10 kg of propellant savings is not deemed worth, and NN B is therefore chosen for the analysis and comparisons documented in the next section.

6.2. PERFORMANCE COMPARISONS IN THE BL SCENARIO

In this section, the performance of the chosen NN will be investigated. In Section 6.2.1, propellant consumption will be compared to the optimal values obtained from GPOPS. Subsequently, the NN will be compared to the benchmark Apollo guidance in terms of accuracy and fuel optimality (Section 6.2.2), robustness against off-nominal ICs (Section 6.2.3), attitude at TD and attiude rate

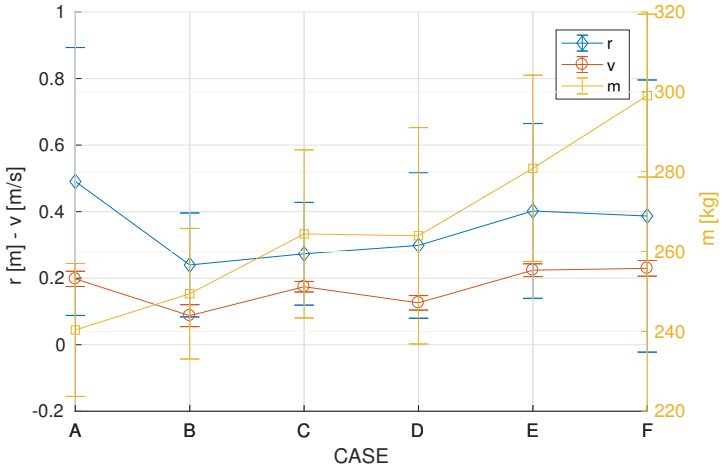


Figure 6.6: Mean and standard deviation values of position and velocity errors and propellant consumption for the best optimization instances.

behavior (Section 6.2.4), and on-board CPU times (Section 6.2.5). As such, this section will fully answer RQ-1.

6.2.1. COMPARISON WITH OPTIMAL SOLUTION

In this section, the selected NN resulting from the RL optimization is simulated with the purpose of assessing its fuel optimality. To this aim, the same grid used to generate the optimal training dataset (Section 5.2.1) is used for the ICs. First, the trajectories are simulated in an ideal enviroment, so that the fuel consumption can be compared as truthfully as possible. Afterwards, the same trajectories are simulated in presence of disturbances, so that the effect of the latter on fuel consumption can be assessed.

The results are presented in Table 6.2, where the optimal values are included for comparison. The last column shows the fuel penalty, that is, the ratio of the mean fuel consumption to its optimal counterpart. For an ideal environment, the neural network produced a mean mass consumption value 21 % larger than the optimal one. It can be seen that the remaining mass penalty values show inequality relations. The exact figure could not be computed given the lack of optimal solutions in the perturbed environment. However, in presence of perturbations, it is quite likely that, in average, mean optimal values will be larger. Given this assumption, one can say that the actual propellant inefficiency is within 20 % of the optimal value for all the cases featuring environmental disturbances (rows 2 to 4 in the table). One can also see that all metrics slightly improve when perturbations are included, except for standard deviation values. This is a consequence of the optimization environment, which includes all these disturbances. Standard deviation, on the other hand, is higher due to the larger range of conditions encountered in a stochastic environment, which leads to more fuel consumption variability. Given these data, one can say that the resulting NN is very robust against these conditions. So, performance consistency over various nominal and off-nominal conditions comes at the cost of not being able to exploit more specific environmental conditions at the fullest. This phenomenon of *specialization vs. generalization* is expected, and is a nice parallel to the *exploration-exploitation* dilemma typical to the RL problem, according to which suboptimal actions must sometimes be taken in specific conditions for an optimum to be reached under more global conditions. Finally, one can say that the NN's propellant consumption is well above the original requirement GR-04, but can still be considered within reasonable bounds.

Mass optimality could not be tested with this set of trajectories for the Apollo guidance because of insufficient control authority, combined with the incapability of the algorithm to handle control constraints. In fact, the set of trajectories producing the values of Table 6.2 stem from worst case

Table 6.2: Propellant optimality comparison under ideal and perturbed conditions.

	m[kg]				$\frac{m}{m^*}$
CASE	μ	σ	BEST	WORST	
IDEAL	248.6	13.06	227.1	278.6	1.21
IDEAL NAVIGATION	245.8	14.00	221.0	277.3	≤ 1.20
NO WIND	246.0	13.94	219.3	278.5	≤ 1.20
ALL PERTURBATIONS	246.0	13.91	220.3	279.5	≤ 1.20
GPOPS	205.6	15.02	175.0	240.3	-

Table 6.3: Comparison between optimized NN and Apollo guidance under different environmental conditions.

		PERTURBATIONS							
		IDEAL NAV - NO WIND		IDEAL NAV		NO WIND		ALL	
		NN	APOLLO[2]	NN	APOLLO	NN	APOLLO[3]	NN	APOLLO
$\|\Delta r\|$ [m]	μ	0.2284	0.008185	0.2248	0.008210	0.5116	0.6388	0.5100	0.6266
	σ	0.08600	0.00395	0.08504	0.003951	0.2524	0.2957	0.2515	0.3164
	WORST	0.5839	0.02959	0.5582	0.02171	1.283	1.528	1.515	1.687
$\|\Delta v_h\|$ [m/s]	μ	0.05440	0.006362	0.05426	0.006247	0.1068	0.09631	0.1052	0.09639
	σ	0.009804	0.002535	0.008225	0.002648	0.05300	0.04285	0.05271	0.04247
	WORST	0.1397	0.02494	0.1261	0.02343	0.3266	0.2349	0.2926	0.2226
v_z [m/s]	μ	-0.4228	-0.4496	-0.4226	-0.4495	-0.4231	-0.4511	-0.4229	-0.4514
	σ	0.01032	0.001226	0.008472	0.001275	0.04434	0.01961	0.04468	0.01860
	WORST	-0.5036	-0.4530	-0.4995	-0.4537	-0.5808	-0.5203	-0.5582	-0.5193
Δm [kg]	μ	243.4	258.2	243.5	258.4	243.2	258.5	243.6	258.0
	σ	6.455	9.841	5.994	10.03	6.169	9.902	6.481	9.820
	BEST	227.8	232.8	227.9	233.7	229	234.1	226.2	234.0
	WORST	266.2	301.2	263.5	316.1	268.7	306.0	270.5	314.1

scenario ICs. Specifically, large negative vertical velocities induce a large Apollo commanded acceleration (see Eqs. (4.24) to (4.26)). If the excess between the commanded thrust value and the maximum one is too large, this results in unfeasible trajectories. However, a comparison between the NN and the Apollo guidance under conditions which are feasible for both systems will be made in the following section.

6.2.2. ACCURACY AND OPTIMALITY UNDER GAUSSIAN ICS

A set of MC analyses has been carried out to compare NN and Apollo guidance. Different perturbation cases have been tested with the aim of investigating the effect of wind and navigation uncertainties on the relative performance of the algorithms. The results are presented in Table 6.3. As it can be seen from the footnotes, some of the MC simulations for the Apollo guidance still produced failures. These have been defined as trajectories resulting in position and velocity errors larger than 2 m and 0.5 m/s, respectively. The requirements have been relaxed with respect to the original requirements GR-03, GR-02 and GR-01, presented in Section 2.2.1, as a result of the introduction of the navigation system in the loop, which produces final dispersions in true position and velocity comparable to its steady-state error. In spite of this, the NN was able to successfully achieve a safe TD for all the MC shots.

[2]Results obtained after removing 1 failure.
[3]Results obtained after removing 2 failures.

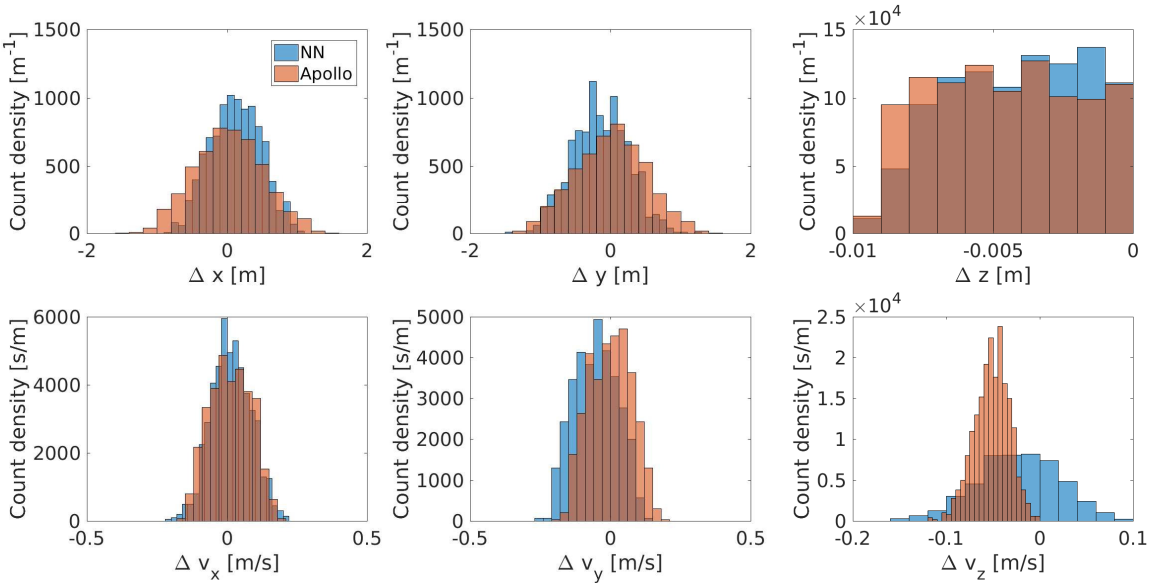


Figure 6.7: Comparison of state dispersion at TD for NN and Apollo guidance.

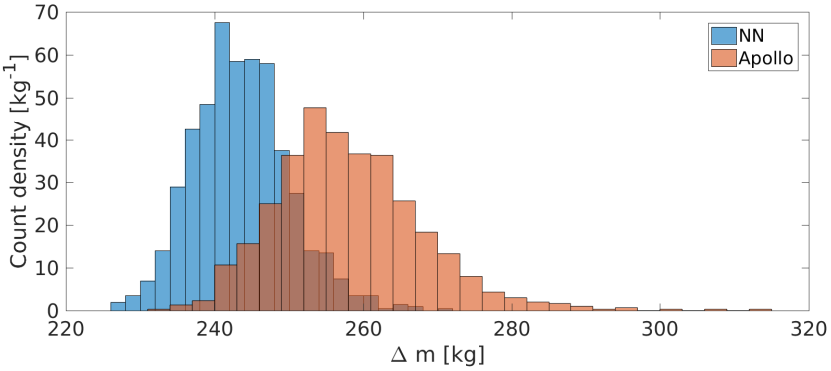


Figure 6.8: Comparison of mass consumption for NN and Apollo guidance.

By looking at Table 6.3, one can see that, once again, mass consumption is little dependent on the environmental conditions, both for the neural network and, suprisingly, for the Apollo guidance. The former, however, is consistently better than the latter. On the other hand, by looking at position and velocity, the change in relative performance for different perturbation cases is evident. Without navigation uncertainties, the Apollo guidance outperforms the NN guidance both in terms of mean, standard deviation and worst case values for position and velocity errors. This is understandable, as position and velocity are final conditions of the TPBVP solved by the Apollo guidance, so that, without strong perturbations, Apollo can meet these conditions to a high level of accuracy. Under ideal navigation, one can compare the performance of the algorithms against the original requirements GR-01 to GR-03. Both algorithms are well within bounds in terms of mean values, while the NN guidance slightly exceeds horizontal velocity and position requirements in terms of worst case values. Position is exceeded by $\sim 6-8$ cm, which does not represent a problem, as it does not compromise landing safety. Horizontal velocity is exceeded by $\sim 2-4$ cm/s, which is also a very low value and can be deemed acceptable, especially considering that navigation system uncertainties are larger than that, and will thus dominate the total error once in the loop. In fact, this is confirmed by last two perturbation cases shown in the table, which feature the navigation sysytem in-the-loop.

Under these conditions, and even more clearly when wind is included, the performance of the two systems is comparable in terms of mean and standard deviation values of position and velocity

residuals. Figures 6.7 and 6.8 show the distribution of state components and mass consumption. The Apollo guidance performs slightly better in terms of worst values for both velocity components. Also, the NN histograms show a non-zero mean distribution for both position and velocity in the cross-track direction, as it will be shown by additional data in the next sections. On the other hand, the NN guidance is a little more accurate in position, with a 3σ value of ~ 0.75 m. While comparable in terms of accuracy, the figure shows the clear difference between the two systems in terms of mass optimality, as the NN not only consumes ~ 15 kg less in average, but also shows less fuel consumption variance. In fact, one can see that for three trajectories, the Apollo guidance uses well more than 300 kg, up to 314.1 kg in the worst case scenario. Given the worst case value for the NN of 270.5 kg, this makes for a considerable difference of 45 kg of worst case propellant usage, whichi is ultimately what constraints the sizing of the tanks.

6.2.3. ROBUSTNESS AGAINST OFF-NOMINAL ICS

Research questions RQ-1.1, RQ-1.2 and RQ-1.3 have been answered in the two previous section. An additional set of simulations has been carried out for each algorithm starting from a grid of initial conditions twice the size the one used to train the NN within the RL process (that is, $\mathbf{x}_0 \pm 6\boldsymbol{\sigma}_0$ were used as outer bounds of such grid). The purpose of this test is that of investigating the robustness of the NN against off-nominal initial conditions, and compare it to the Apollo guidance capabilities, thus addressing RQ-1.3.

First, robustness against initial position variations is looked into. The failure maps in Figure 6.9 show the ratio of unsuccessful-to-total trajectories for different initial horizontal positions, or failure rate γ. As in the previous section, the relaxed values of 2 m in position and 0.5 m/s in velocity have been used to identify failures. The value for each position point is computed as the average over all the possible initial velocity combinations starting from that position. One can see that the variations in failure rate as a function of position are very small as compared to the mean value. As such, velocity can be said to have the most effect on failures. However, the plots serve the purpose of giving insight into how varying initial positions influence the performance of the algorithms. In general, both algorithms' performance degrades gracefully with increasing downrange distance (x-axis), and values are quite comparable between the two cases, ranging between 0.25 and 0.35. However, the NN shows some asymmetries in the crossrange direction (y-axis), as was already pointed out in the discussion of Figure 6.7, and its failure rate increases slightly for smaller downrange distances, which is in contrast with the general trend. These areas of the position space are not optimized for, and performance is therefore not expected. However, the behavior of the NN cannot even be simply predicted, given the stochastic nature of the RL algorithm. The NN parameter vector resulting from this specific optimization instance happens to produce this behavior, whereas another network would likely produce a different result. This requires tailored verification of each specific network obtained through this method. Also, it makes it hard to come up with a suboptimal, yet feasible, solution if the conditions are outside the known domain. While this could be solved for horizontal position anomalies by a shift in target LS position, which would bring the LS-relative horizontal position back within the known domain, velocity deviations must necessarily be dealt with directly. The robustness against these state components is investigated in the following.

For each point in the three dimensional velocity space, a value is associated which is computed as the mean failure rate over the points in the position space. Figure 6.10 shows the result. One can see that the variations are more significant than they are when position is the varying variable. The most striking difference is the distribution of the failures, which has a regular pattern and is symmetric in the crossrange direction for the Apollo guidance, while it is more randomly distributed for the NN, in line with what has been seen so far. On the other hand, the two plots are similar in that failure rate is $\gamma = 0$ (no failures) for both guidance systems for values close to nominal, that is, the two central rows for vertical and downrange velcity, regardless of crossrange velocity. The other sim-

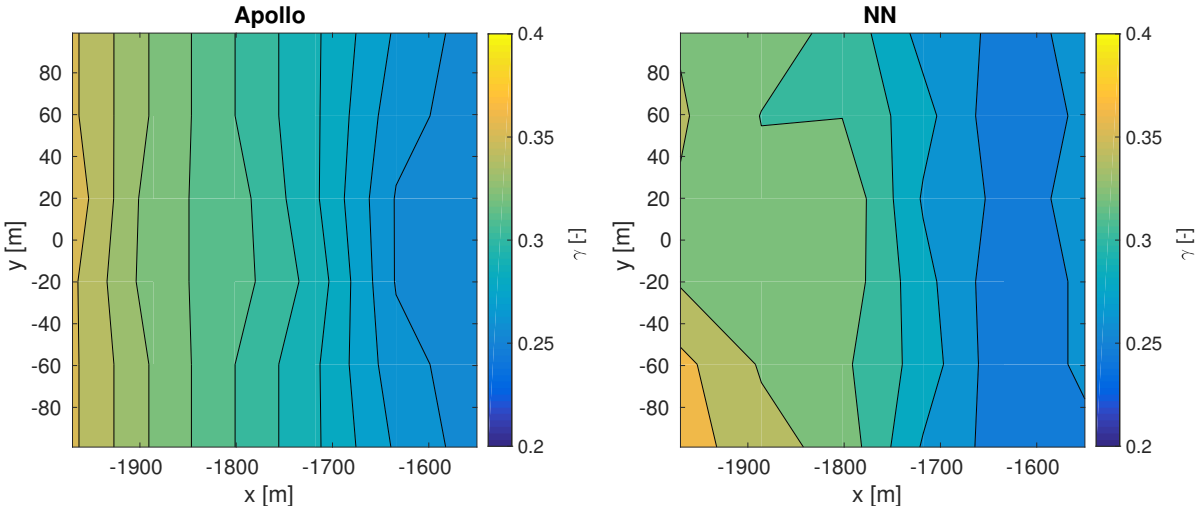


Figure 6.9: Comparison of robustness against initial position variations.

ilarity is a failure rate of $\gamma = 1$ (always fails) for the largest negative value of vertical velocity. As far as the Apollo guidance is concerned, that is because of its incapability to handle control constraints, as explained in Section 6.2.1. In fact, as Apollo's commanded thrust exceeds by too much the maximum thrust value, the algorithm is not able to close the loop anymore, thus producing failures. On the other hand, the NN fails because it has not been trained to operate in those areas of the velocity space.

In other areas of the velocity space, however, visible differences are present between the two algorithms. In particular, it is surprising how well the Apollo guidance performs for the upper-half of the inital velocity space. Regardless of the value of downrange velocity, the Apollo guidance never fails for small vertical velocities, whereas the NN shows a very varying behavior. Depending on the downrange velocity, failure rates range between 0 and 50 %. Again, this shows the unpredictable behavior of the network when trained with a stochastic method, as it might generalize very well to certain off-nominal regions, but not to others. As such, this also explains other conditions under which the NN results in failures, while the Apollo guidance it is $\gamma = 0$ (e.g., small downrange velocities and $v_v \sim -50\,\mathrm{m/s}$). Also, this specific network, for this specific grid of ICs, behaves as expected within the training domain, but examples will be presented in Section 7.2.3 where this is not the case, as the network fails also under conditions contained in the domain bounded by the training trajectories, which further complicates the verification of the optimization output.

Finally, other areas of the velocity space, show noticeable difference in favor of the NN. This mostly happens for vertical velocities which exceed mildly the training domain in the negative direction, and in particular when downrange velocities are large in the negative direction. In fact, for the second largest negative vertical velocity, one can see the Apollo failures progressively increase for decreasing downrange velocities. Failure rates are high (50 % to 70 %) also for the largest negative downrange velocity and $v_v \sim -80\,\mathrm{m/s}$. This is because these conditions induce an even larger commanded thrust and, therefore, larger excess values with respect to the saturation value. On the other hand, the NN is less sensitive against these conditions, as it shows, in average, a lower failure rate. However, like the Apollo guidance, it is also less robust when both vertical and downrange velocities take on large negative values. What is surprising about the behavior of the network in this area of the velocity space is how sensitive it can be to variations in crossrange velocity, which do usually lead to asymmetric behaviors, but in a smoother way. For these particular combinations, on the other hand, one can see that failure rate can vary from 0 to 100 % for variations of only a few meters per second in cross-track velocity. For instance, when $v_{v,0} = \begin{bmatrix} 16 & -30 & -102 \end{bmatrix}^T \mathrm{m/s}$, $\gamma = 1$, whereas

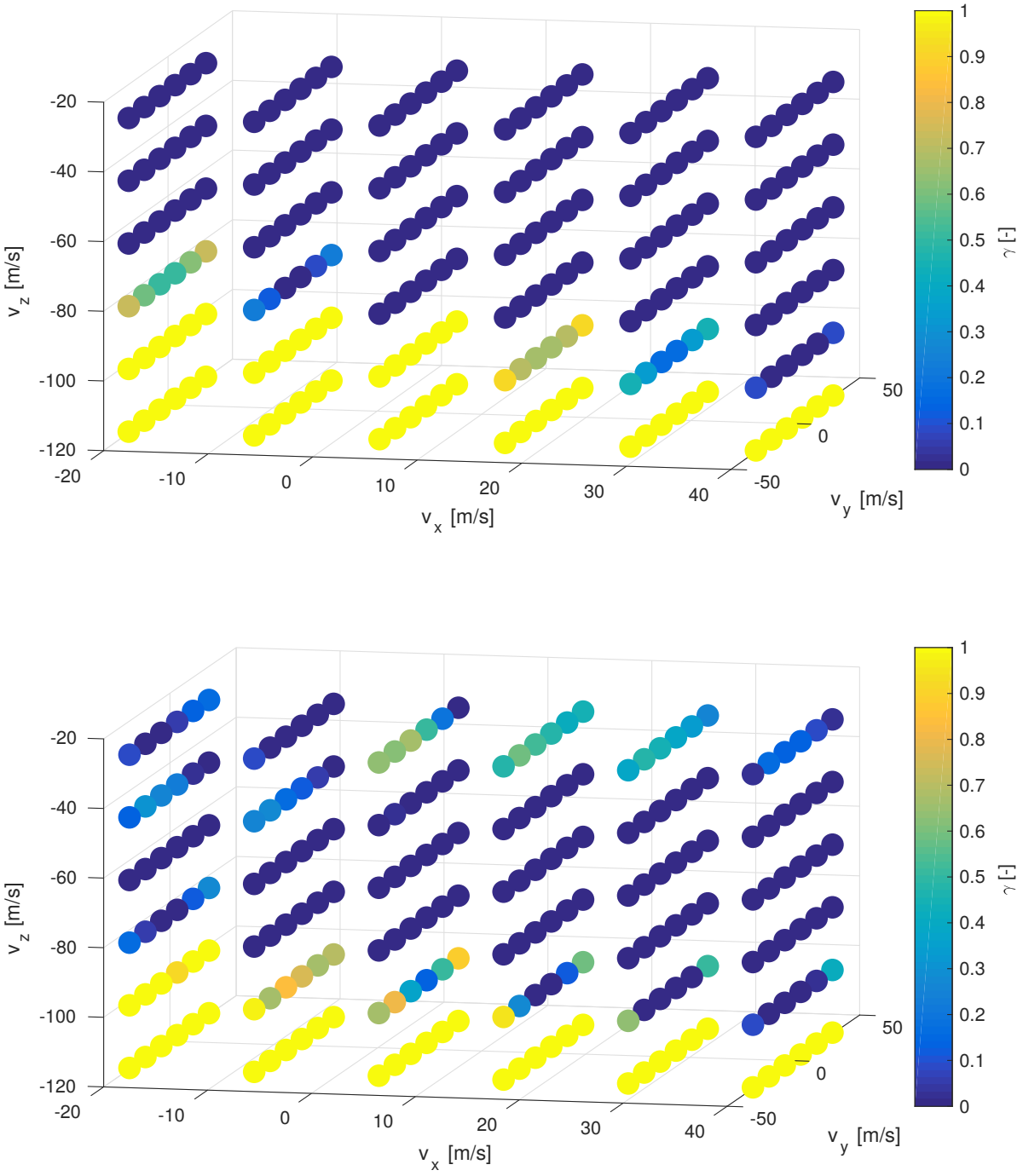


Figure 6.10: Comparison of robustness against initial position variations.

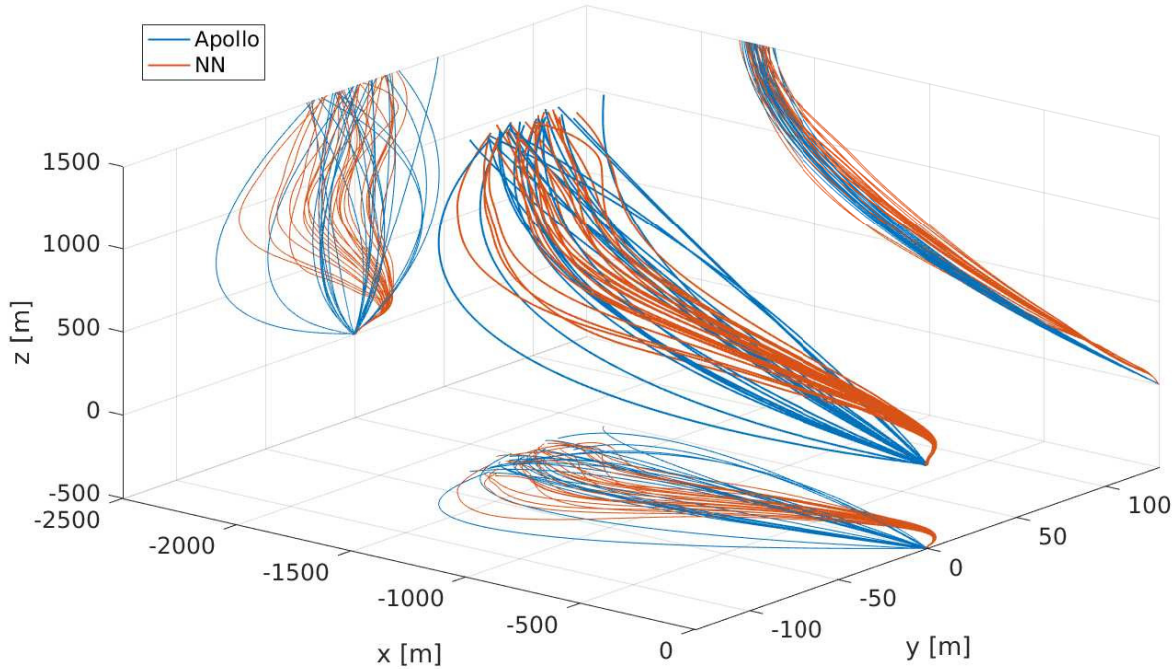


Figure 6.11: Comparison of 25 Apollo guidance and NN trajectories.

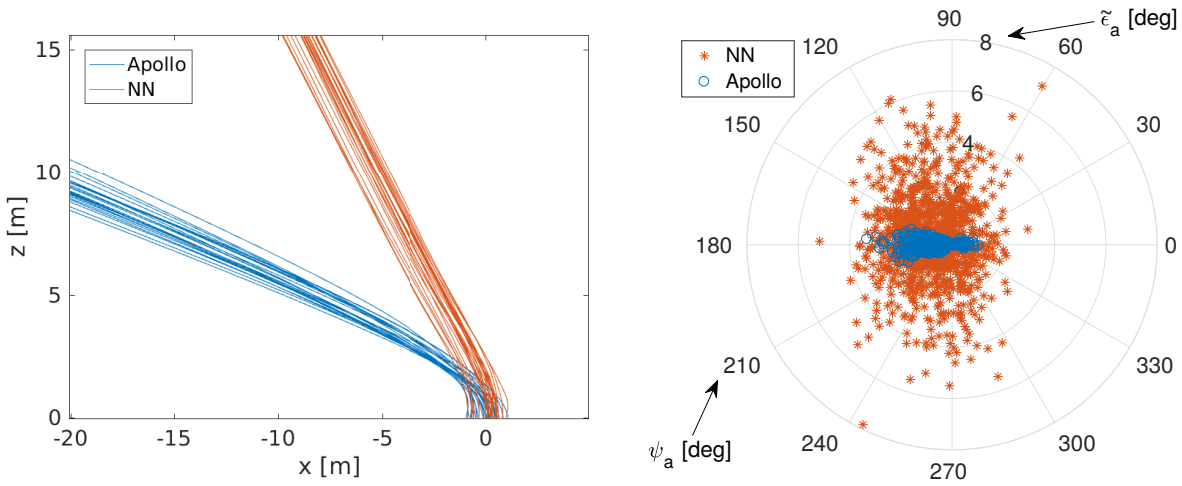
when $v_{v,0} = \begin{bmatrix} 16 & -6 & -102 \end{bmatrix}^T$ m/s, $\gamma = 0$.

As the robustness of the NN mainly depends on the size of the domain within which it has been trained and the density of visited states within such domain, if trained in an extended area of the velocity space, a lower failure rate for large negative values of vertical and downrange velocity could in principle be achieved, given that a sufficiently thorough search of the parameter space is performed. Whether it could possibly achieve zero failure rate for these states depends on the optimal solution for the given maximum thrust value.

6.2.4. ATTITUDE AND ATTITUDE RATE

Figure 6.11 shows the comparison between Apollo guidance and NN trajectories for a small, yet representative, batch of 25 MC simulations. The central set of lines shows the three-dimensional trajectories, while the graphs on the side and bottom show the projections of such trajectories on the respective planes. One can see that the NN trajectories are wavy in the cross-track direction (y-axis), while keeping in mind that the magnitude has been exaggerated tenfold to make the effect more visible. On the other hand, the x-z plane projection shows that the NN trajectories fly higher than the Apollo ones, which is beneficial for the elevation angle ε_{SC} of the S/C, as seen from the LS, as it minimizes distortion and maximizes resolution of the optical sensor.

From Figure 6.11, one can also see that the NN approaches the LS from a more vertical direction than the Apollo guidance does, which follows from the higher trajectory flown hitherto. This can also be seen in the zoom-in of Figure 6.12a. This reduces the pitch-up maneuver to be performed before TD to achieve landing in a vertical vehicle configuration. To this regard, an analysis on the off-vertical attitude residuals at TD has been carried out for the sake of verifying whether requirement GR-08 is met. The results are shown in Figure 6.12b, where the variables $\tilde{\varepsilon}_{TD}$ and ψ_{TD} are the



(a) Zoom-in of the x-z plane projection plot of Figure 6.11, highlighting different flight path angle and pitch-up maneuver upon LS approach.

(b) Comparison of Apollo and NN attitude at TD. The plot shows residual angle with respect to the vertical.

Figure 6.12: Comparison of LS approach trajectories and attitude residuals at TD.

magnitude and azimuth of the attitude residuals, respectively. These are computed as:

$$\tilde{\varepsilon}_{TD} = \arccos\left(\frac{\mathbf{a}_f^{cmd}}{\|\mathbf{a}_f^{cmd}\|} \cdot \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}^T\right) \tag{6.5}$$

$$\psi_{TD} = \arctan2\left(\frac{a_{f,y}^{cmd}}{a_{f,x}^{cmd}}\right) \tag{6.6}$$

where $\mathbf{a}_f$ is the commanded acceleration vector at the final step. One can see that the NN worst case value is very close to the maximum allowed value of 8^o, but still within the requirement set by GR-08. The Apollo guidance residuals, on the other hand, are well within 4^o. The difference is particularly visible in the crossrange direction, as a result of the asymmetric behavior of the NN visible in Figure 6.11. On the other hand, the Apollo distribution is strongly asymmetric in the downrange direction, with most of the samples concentrated in the backward direction ($\psi_{TD} = 180^o$). This is likely because, as can be seen in Figure 4.5b, the GMM-2B gravitational acceleration is slightly larger that the reference surface gravity $g_{Mars} = 3.7114\,\mathrm{m/s^2}$, which is used by the Apollo guidance logic to compute the effective vertical acceleration component from the net value obtained by solving the TPBVP. As such, after the Apollo guidance switches to open-loop mode, TD occurs earlier than expected, while the vehicle is still slightly pitching backwards, trying to null the residual downrange velocity. This is confirmed by checking the mean of the distribution shown in the bottom-left plot in Figure 6.8, which proves to be $\sim 1\,\mathrm{cm/s}$ in the positive downrange direction.

The final attitude distributions of the two guidance schemes seems to be in contrast with what has been said before about a larger flight-path angle being more desirable to limit the extent of the required pitch-up. First, this would be true in a real case scenario, where the attitude rate is limited by the lander's rotational control authority. Besides, the final acceleration is a boundary condition in the Apollo's TPBVP. This explains why, under unconstrained rotational dynamics, the Apollo guidance is able to better meet this condition even after running a few seconds in open-loop and approaching with a shallower flight path angle.

As resulting from a stochastic optimization, every network behaves differently. Showing the results presented above for every network that has been obtained would be unpractical. The final

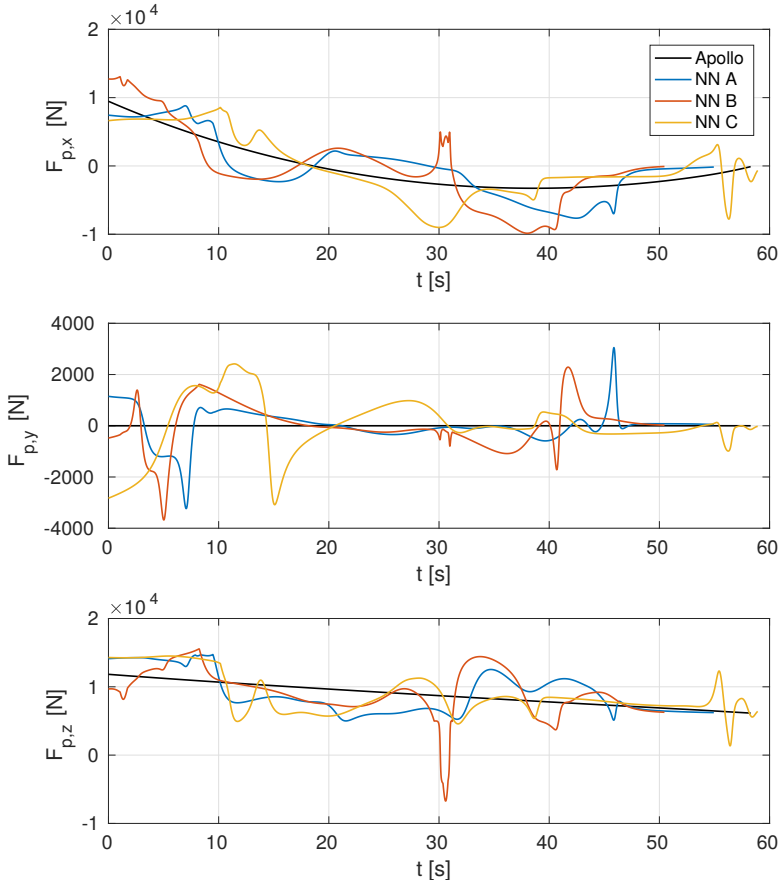


Figure 6.13: Comparison of propulsive force time histories between Apollo guidance and different neural networks.

attitude distribution plots will be given for a number of different network, trained for the extended scenario, in Section 7.2.4, but will not be dealt with further here. Instead, the root cause that leads to the irregular shape of the NN trajectories (that is, commanded acceleration and resulting thrust vectors) will be investigated for three different networks (namely A, B and C in Table 6.1) and compared to the Apollo guidance solution.

Figure 6.13 shows the time series of the propulsive force components for the nominal trajectory (i.e., nominal IC). All environmental and system perturbations have been removed to isolate the NN behavior. As compared to the Apollo guidance policy (black line), the NNs (colored lines) show irregularities in their output, consistently across different networks. The main issue this leads to is that of large attitude rates, which could not be coped with if rotational dynamics and kinematics were accounted for, and ultimately makes these trajectories unfeasible in a real case scenario. Disregarding rotational dynamics altogether, and considering kinematics only, Topcu et al. [2007] provide a reference for an acceptable maximum value for rotational rates during PDG phase, which is of $\sim 3\,\mathrm{deg/s}$.

With this in mind, let us look at Figure 6.14, which compares attitude rates for the Apollo guidance and the three networks shown in Figure 6.13. Rotational rate has been computed here as the rate of change between consecutive commanded acceleration vectors, that is:

$$\omega_k = \frac{\arccos\left(\frac{\mathbf{a}_{cmd,k+1}\cdot\mathbf{a}_{cmd,k}}{\|\mathbf{a}_{cmd,k+1}\|\|\mathbf{a}_{cmd,k}\|}\right)}{t_{k+1}-t_k} \tag{6.7}$$

The downward spike visible in the Apollo guidance profile is due to the stationary point in the downrange force profile (top plot in Figure 6.13), which leaves all attitude changes up to variations in ver-

tical component. This, however, does not affect the attitude very much, as the vehicle is very close to a vertical configuration. Hence, the strong minimum in the attitude rate profile. In the same profile, a small dent is also visible at $t \sim 53\,\text{s}$. This is due to the switch to the open-loop mode, as well as the discrepancy between the guidance logic's accounted-for gravitational acceleration and the real one. In closed-loop, this error is accounted for at every step, whereas this does not happen in open-loop. Hence, the discontinuity in attitude rate. Both because of this, and due to the approaching pitch-up maneuver, attitude rates increase during the final seconds, slightly exceeding the maximum value. However, this excess is very small at about 1 deg/s, and does not represent a problem for landing safety, as was seen in the attitude residual plots of Figure 6.12b.

Besides, what the figure most importantly shows concerns the attitude rates of the NN trajectories. One can see that the large variations in commanded acceleration components result in large attitude rates. At times, instantaneous rotational rates reach hundred or even thousand of degrees per second, which are completely unrealistic. If attitude rates are limited during the simulation, these peaks could possibly be withstood, as their duration is short, and the system might recover and manage to close the loop after a few seconds. However, the numerous long segments of trajectory with lower rates, yet consistently above the maximum value, hopelessly result in unfeasible trajectories if rates are hard-constrained. Besides, especially for NN C, large attitude rates are present very close to TD, which are also problematic, as the time available for recovering the trajectory is even shorter. The latter issue will be solved in Section 7.1.3. The former will be addressed further on in Section 7.2.4.

6.2.5. ON-BOARD CPU TIMES

The computational times of the two algorithms has been compared to verify whether the NN guidance can meet requirement GR-10, that is, run on-board in real-time. The computation of one command has been repeated 100000 times, using a code which has been optimized in Matlab, by vectorizing all possible operations. The test has been run on an Intel Core i5-5200U quadcore CPU at 2.20 GHz. The code has been warmed up so as to remove first-time overhead, and as few tasks as possible were kept running on the system while performing the test. The `tic-toc` functions have been used for the estimation, as adviced by Matlab documentation. Different network sizes have been tested, as well as two versions of the Apollo guidance, that is, an open-loop and a closed-loop version. In fact, if the Apollo is run in closed-loop, it allows for a considerable reduction of the computational load. That is because only the constant vector $\mathbf{c}_0$ of the linear system in Eq. (4.25) is necessary to the computation of the acceleration vector. As such, only the first three rows of the linear system matrix need be computed. The results are shown in Figure 6.15.

One can see that some variance is still present in the estimates, as indicated by the 1σ bounds. However, the standard deviation shows a regular behavior over different network sizes, and it is

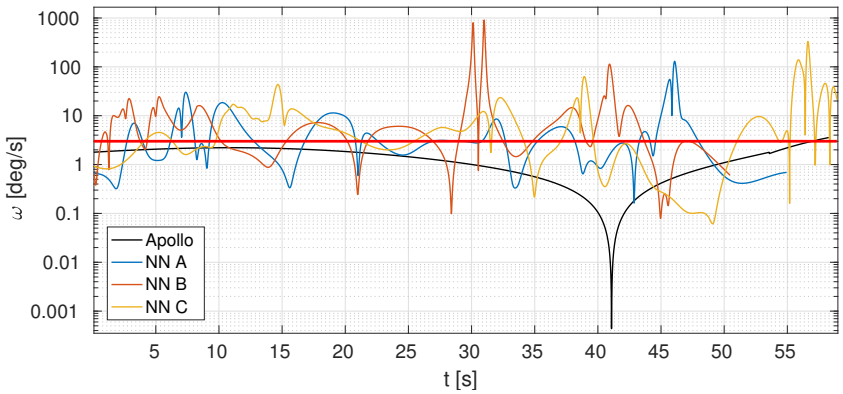


Figure 6.14: Comparison of attitude rate time histories between Apollo guidance and different types of neural networks.

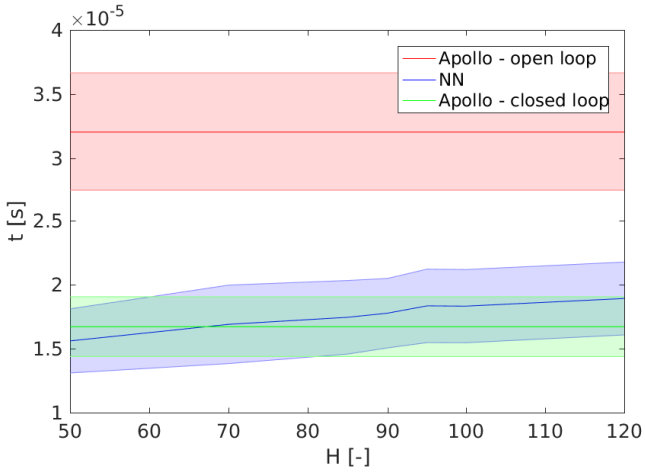


Figure 6.15: Computational load per time step for Apollo and neural networks of different sizes, with 1σ bounds.

comparable between NN and closed-loop Apollo. One can see that the NN performs very well in this test, being comparable to the closed-loop Apollo guidance. Also, the increase is linear with hidden-layer size, and the steepness is low, so larger networks would still perform well. The open-loop Apollo, on the other hand, performs clearly worse. Estimate variances are likely due to system multi-tasking, which explains also why longer computation times in the case of the open-loop Apollo increase its variance as well. Given the Apollo guidance has been flown more than 50 years ago, this test proves that the NN satisfies GR-10, and could run in real-time on on-board hardware.

6.3. KALMAN FILTER TUNING

As already mentioned in Section 4.12, after the introduction of the NN guidance in the simulation loop, it has been noticed that, for certain combinations of the filter's tuning parameters, the LKF started running into stability problems. In particular, a phenomenon of resonance with the NN output was taking place. The critical parameters are found to be a large integrator step-size and low $\mathbf{Q}/\mathbf{R}$ ratios.

Figure 6.16 shows the behavior of true and estimated velocity components (blue and red lines, respectively), estimation errors (yellow line), and commanded acceleration (purple line) over the simulation timespan during which the phenomenon occurs. The oscillations seem not to propagate in a significant way to the position components, which are therefore not reported here. In the top plots, the navigation system is in the loop, whereas in the bottom plots, it is out of the loop.

By comparing the two scenarios, it is quite clear what sets the resonance off. The peaks in commanded acceleration increase the rate of change of the corresponding velocity component to an extent the linear Kalman filter cannot cope with at low frequencies (i.e., 1 Hz). This creates a lag in the estimated state components, which, with navigation in the loop, induces a counter-action by the guidance system. In the best case scenario, this occurrance causes the lander to steer away from a direct TD, resulting in a delay and, therefore, higher propellant consumption. In the worst case scenario, the lander leaves the known region of the state space altogether, which results in a hard landing.

This phenomenon is believed to be caused by various factors. Beside the previously mentioned step-size and $\mathbf{Q}/\mathbf{R}$ ratio, the linearity of the Kalman filter combined with non-smoothnesses in the policy plays a major role in the onset of this behavior. In fact, the latter induce large variations in commanded acceleration for small variations of the input state, especially during the last phase of the trajectory, where the NN seems to be more sensitive to state perturbations. These, in turn, lead to a lag in the estimated state computed by the LKF, which suffers from lack of robustness against

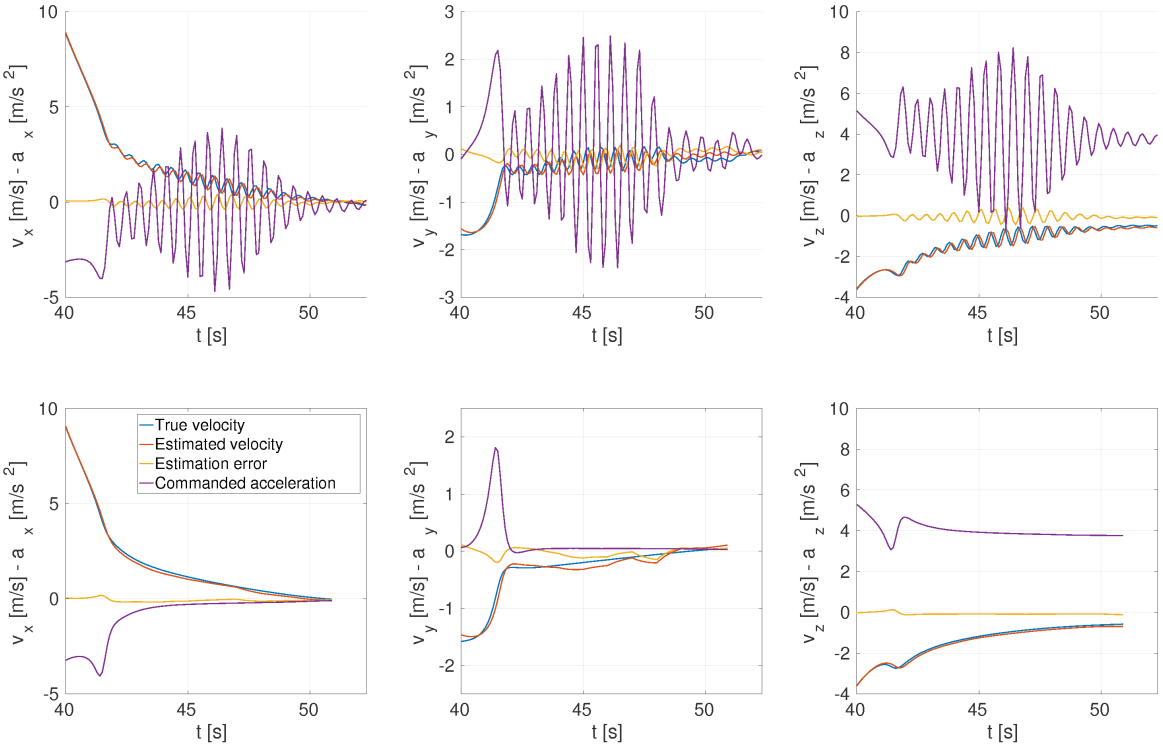


Figure 6.16: Resonance between estimated state and NN policy.

fast dynamics, in particular when its frequency is too low.

As such, this issue has been solved by reducing the integrator step-size (from 100 ms to 20 ms), which allows to increase the Kalman filter frequency as well, and by increasing the **Q**/**R** ratio for the vertical direction, having the filter rely more on the non-inertial measurements. In fact, increasing **Q**/**R** for the vertical components only seems to solve the issue for the other components as well, while allowing for more noise reduction in the horizontal direction. Therefore, the final **Q** and **R** matrices are:

$$\mathbf{Q} = \mathbf{I}_{6\times 6} \tag{6.8}$$

$$\mathbf{R} = \begin{bmatrix} 100 & 0 & 0 & 0 & 0 & 0 \\ 0 & 100 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 50 & 0 & 0 \\ 0 & 0 & 0 & 0 & 50 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.5 \end{bmatrix} \tag{6.9}$$

These values have been used for the simulations presented in this chapter, and they will be kept constant for the remainder of this work. The velocity and acceleration histories after applying these changes are shown in Figure 6.17 for the timespan of interest. One can see that the estimated error behaves now as expected, as it remains close to its steady state value across the timespan during which the resonance effect was previously occuring. Also, small variations in commanded acceleration are present, which are caused by the above-mentioned oversensitivity of the network to state knowledge noise during the final part of the descent. However, these do not cause stability problems, yet they do have the effect of incresing attitude rates, as was seen in Section 6.2.4.

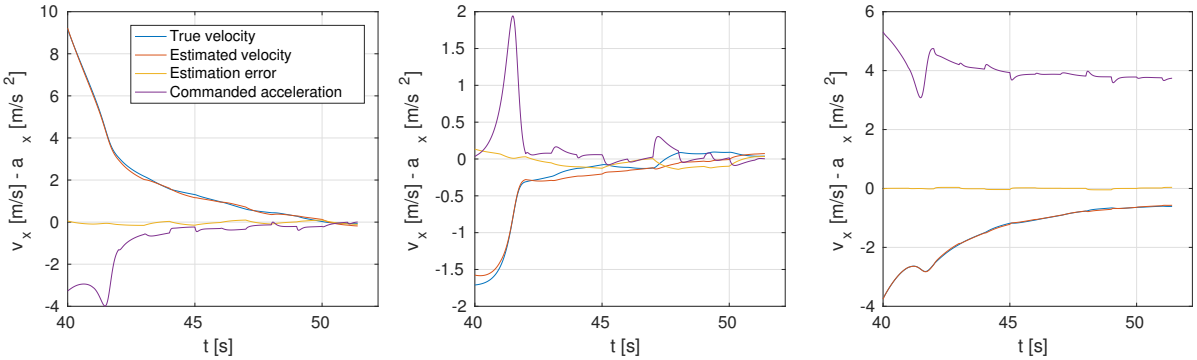


Figure 6.17: Results after retuning of the filter and reduction of the step-size.

To the best of what could be observed, these measures prevented this phenomenon from happening again for any other network than the one tested here. However, before applying these modifications, this phenomenon showed up consistently across different networks, although more severely for some than for others. Given the insight gained into the typical policy behavior of networks obtained with the stochastic search algorithm, this is indeed not surprising, as many networks produce very irregular policies (see Figure 6.13), thus setting off the resonance phenomenon.

6.4. CONCLUSIONS

To summarize the findings of this first part of the work, we will start from the lesson learned from setting up and runnning the optimization algorithm. To this regard, one must say that the random nature of this method makes it lengthy (high number of optimization loops needed for achieving convergence) and, therefore, tedious to tune, as many instances are needed before an insight into the effect of the tuning parameters can be gained. Another issue the random component introduces is that of consistency over instances, as many optimization trials get stuck in far-from-optimal local minima. A possible improvement in this this direction would be the implementation of an adaptive scale factor, which is increased again when convergence slows down. On the other hand, an upside of this method is that networks with difference performance features are obtained, so that the most suitable network can be chosen *a-posteriori*. However, this complicates the verification procedure (GR-11), as each network needs to be thoroughly inspected, and even then a clear definition of its properties remains challenging. For a successful outcome in this sense, MC simulations need to be combined with different inspection methods, which can give insight into the structure of the NN output over the state space. Also, a backup solution if the current state is outside the known domain, which can bring the lander to a safe landing in a less optimal way, is required, as coming from the autonomy requirement GR-09.

On the other hand, optimizing the network in a stochastic environment brings about advantages in terms of robustness. Variations in environmental disturbances do not heavily affect the performance of the NN, which often performs even better in presence of perturbations. As a matter of fact, mean values do usually improve, while variances increase, because of the larger range of encountered conditions. Even so, and in particular as fuel consumption is concerned, the standard deviation value is much lower for the NN than it is for the Apollo guidance. This is very desirable, as it reduces worst case values ($\sim 270\,$kg vs. $\sim 314\,$kg for Apollo), which is ultimately what determines on-board propellant. The NN also performs better in terms of mean value ($\sim 244\,$kg vs. $\sim 258\,$kg), which exceeds the original requirement of being withing 10 % with respect to the optimal one (GR-04), but it can still be considered acceptable, as it is below 20 %.

In terms of accuracy, Apollo and NN perform approximately equal. Navigation system system errors dominate final true state errors, but both algorithms are able to withstand these perturba-

tions and limit true state dispersions at values close to navigated errors. In terms of sheer accuracy, the Apollo guidance performs understandably better when an ideal environment is assumed, as final conditions are fixed in the TPBVP. However, the NN only slightly exceeds requirements GR-01 to GR-03 in terms of worst case values ($< 10\,$cm position exceess and $< 5\,$cm/s velocity excess), while mean values are within bounds by a good margin: its performance can thus be considered acceptable. Final acceleration is also fixed in the Apollo guidance, which explains the better performance of the algorithm in terms of residual attitude at TD, as compared to the NN. The latter, however, is still within the required maximum value of 8^o off-vertical angle (GR-08), which is surprising given that no constraint was included on this variable. Where the NN performs worryingly bad is rotational kinematics. While the Apollo guidance is within the requirement of 3^o/s (GR-07) for most of the time, the NN commanded accelerations show sudden and large variations, which could not be coped with if they were constrained during the simulation. This will be addressed in Section 7.2.4, when additional constraints will be added to the optimization process.

Finally, the NN did great in the CPU performance test, proving to be able to run on on-board CPUs (GR-10), as computing times where in the order of those of the Apollo guidance. On the other hand, the last credit is given to the Apollo guidance, whose robustness against perturbations (e.g. winds and measurement noise) is surprising, given its simplicity and different design focus. It copes well with varying ICs as well, with failure rate of zero for most of the tested IC space. However, this test did reveal its limits in terms of handling control constraints, as it fails for larger negative vertical and downrange velocities, when an efficient allocation of the available thrust is needed. The NN performance, on the other hand, is less clearly defined, as it is suboptimal in regions where Apollo performs excellently, while it is more robust in areas where Apollo starts running into thrust allocation issues. Ultimately, the NN shows the potential of better coping with varying ICs. In this direction, a larger and denser training domain would be beneficial and, given a feasible trajectories (for optimal thrust allocation) and a sufficiently long training time, it could allow for zero failure rates in these areas of the IC space as well.

7

RESULTS - FULL SCENARIO

This chapter presents the work that has been done to adapt the algorithm to the full scenario, and the corresponding results. Specifically, in Section 7.1, the capability of the NN guidance to handle LS redesignations during the landing maneuver will be investigated and the implementation of the Pointing Constraints (PCs) will be presented, as well as the necessary adaptations to the algorithm. Section 7.2 presents the results that have been obtained from the optimization of the NN in the extended scenarios. Finally, in Section 7.3, the sensitivity of these networks with respect to varying ICs, system and environmental parameters will be looked into.

7.1. BACKGROUND AND ALGORITHM MODIFICATIONS

In this section, considerations will be made regarding the introduction of RTs (Section 7.1.1), and the enforcement of PCs (Section 7.1.2). Finally, in Section 7.1.3, an hybrid guidance approach will be presented, that brings about a number of advantages, de facto enabling the optimization process in presence of these additional training elements.

7.1.1. LANDING SITE REDESIGNATIONS

First, the robustness of the NNs resulting from previous optimizations against RTs is investigated, to understand what the current capability is, how much it can be improved, and what needs to be done to do so.

As from guidance requirement GR-06, established in Section 2.2.1, the objective is that of achieving a RT distance of 800 m for the RT, taking place at 1000 m altitude. To compare current performance against this requirement, two different neural networks have been simulated using a grid of 32 worst case initial conditions (the same used for the optimization procedure, see Section 5.4), for each of which 252 different RTs are applied. These RT points are distributed in the x-y horizontal space according to a 18×14 grid, over a [-800, 1292]m interval in the along-track direction, and a [-800, 800]m span in the cross-track direction.

Figure 7.1 shows the failure rate maps, obtained by averaging the single failure rate maps over the different ICs, for two different networks, namely NNs A and B, presented in Table 6.1. In both cases, the NNs already show some robustness against RTs. Even though they were not trained accounting for this, the information provided for the training domain is generalized to areas outside such domain, thus still producing feasible trajectories for moderate RT distances and state combinations. In particular, NN B shows failure rates lower than 0.2 in an area of $1000\,\text{m} \times 500\,\text{m}$ in along- and cross-track directions, respectively. On the other hand, an unexpected local maximum is present within such area at $x \sim 300 - 400\,\text{m}$, which then decreases again for larger downrange distances. This is likely due to the generalization properties enclosed in the NN parameters, which is

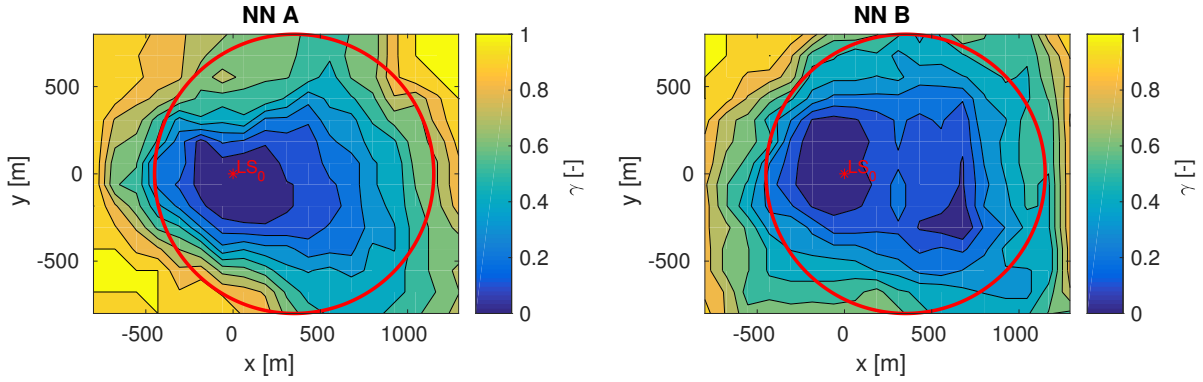


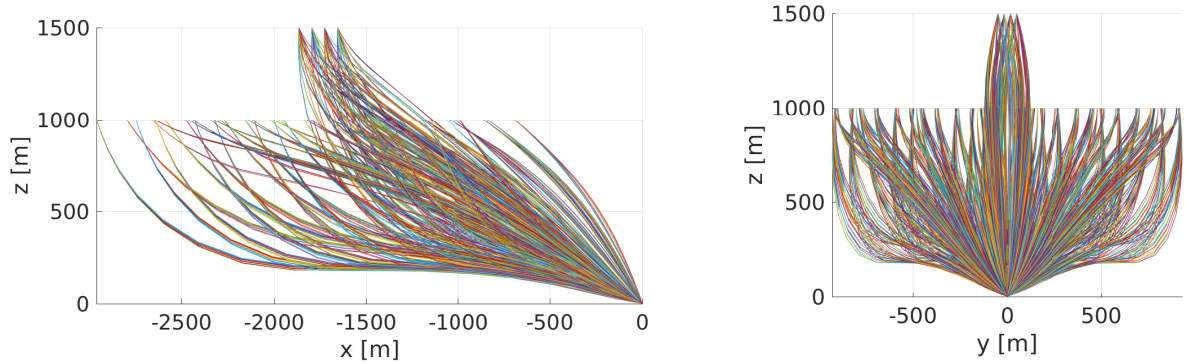
Figure 7.1: Failure rate maps, averaged over a set of worst case trajectories, for two different neural networks.

hard to characterize *a-priori*, and must be done by analysis of simulation results. A smoother failure map is that of NN A, which shows a minimum area of $\gamma = 0$ (no failures) in the area immediately surrounding the nominal LS (red asterisk), which then increases with incresing RT distance.

However, one can see that, for both networks, failure rates increase in an approximately symmetrical fashion with respect to the nominal LS in the y-direction (cross-track), but not in the x-direction (along-track), as there is more tolerance for forward than for backward RTs. In fact, a forward RT means a backward shift in LS-relative position, and as the agent moves forward towards the target, it is more likely to re-enter the state-space domain where it has been trained. On the other hand, a backward RT results in a forward shift in relative position, placing the lander in front of the known state space domain. Moving forward towards the newly designated LS, the vehicle is less likely to re-enter such area, thus remaining in parts of the state space which have not been optimized for. On top of this, backward RTs are inherently more demanding, as they require swifter changes in state, and thus larger changes in commanded acceleration and more propellant usage. This leaves less tolerance for control errors, ultimately resulting in higher failure rates.

As a result of this preliminary analysis, a circular area of 800 m radius is identified, which approximately minimizes failure rate, and is thus expected to ease the optimization of the network in the full scenario. Although being shifted forward with respect to the nominal LS by 350 m, it still provides a RT area of $\sim 2\,\text{km}^2$, which is in line with the one required by GR-06. In fact, although the achievable distance from the nominal LS could also be an important variable in the choice of the hazard avoidance system capabilities, the total reachable area is deemed more important, as it allows to maximize the number of potentially safe LSs the lander can reach. Therefore, the area marked by the red circles in the figures presented above is chosen as the RT area for carrying out the RL optmization.

Now that the RT area has been identified, it is important to check the reachability of such area under optimal conditions, to ensure enough control authority is available. To this end, a set of trajectories has been generated using GPOPS. Starting from the usual grid of 32 ICs, which bound the domain of interest of the state space for the BL scenario (i.e., no RT), 36 RTs have been applied at 1000 m altitude, distributed over a 6×6 grid of target points. The crossrange direction interval goes from -800 m to 800 m, whereas the along-track range of interest is shifted forward by 350 m with respect to the nominal LS, resulting in the interval [-450, 1150]m. This batch of trajectories, together with the original retargeting-free dataset, is shown in Figure 7.2 from different viewing angles. One can see that all trajectories, which are shown in LS-relative coordinates, can reach the designated LS without violating any crash constraints. By looking at the figure, one can clearly tell one database from the other: the trajectories starting at 1500 m altitude represent the old, retargeting-free dataset, while the extended batch, starting at 1000 m altitude, consists of the retargeted trajectories. The coordinates indicate LS-relative position, which is ultimately the input to the guidance system. As



(a) XZ-plane view.

(b) YZ-plane view.

Figure 7.2: Extended set of optimal trajectories.

such, these figures show the relative state space where the NN will be trained.

If one were to follow the approach used for the BL scenario, one would now move on to train a NN by supervised learning, given the acceleration vectors associated with the just presented set of trajectories, before optimizing it in the RL framework. However, in doing so, it was found that the performance of the trained network for the RT scenario was, expectedly, very poor as compared to the fit achieved for the BL scenario dataset. This can be seen in Figure 7.3, where the supervised training RMS errors for different network sizes and input datasets are shown. One can immediately see that the network size has very little effect as compared to that of the dataset used, as increasing the number of hidden neurons up to 150 was still not sufficient to improve the accuracy of the fit by a significant amount. This could be due to an insufficient number of hidden neurons, or even layers. In fact, although theoretically possible to approximate any function with a shallow NN [Cybenko, 1989], a deep NN could allow for smaller numbers of hidden neurons, thus speeding up training times while allowing fitting of a more complex function. Also, since the number of maximum iterations was set to 1000 for computational reasons, it could be that a longer training time was needed for a better fit to be achieved.

By looking at the figure, one can see that three datasets have been tested, namely the BL one (A), the RT one (B), and the one resulting from the combination of these two (A+B). While the drop in performance from the BL to the RT dataset is expected, as a larger state space is involved, more suprising is the fact that the training results for the combined dataset are actually better than for the RT dataset alone. The opposite was expected, as the combined dataset is not only extended in the position state, but also in terms of velocity. In fact, by looking at Figure 7.2, one can see that, in the areas of the position space where the two datasets intersect, large differences in velocity are present, as indicated by the relative incidence of the trajectories from the different batches. The only hypothesis that can be thought of for explaining this phenomenon is that the network achieves a good fit only in the central regions of the training space, when the combined dataset is used. Given the larger number of samples in this region, the errors in the more remote areas of the state space are more than offset, while this cannot happen when only the RT dataset is used, as sample density is more homogeneous, and so must be the fit.

This is, however, of little practical relevance at this point. In fact, performing the RL optimization starting from NNs obtained by supervised training using either of the extended datasets proved unfeasible, as the fit was not good enough for the subsequent stochastic search algorithm to converge in any reasonable amount of time. However, it did prove possible to successfully optimize the network by starting over from a set of weights obtained from the BL training dataset alone, proving once more the remarkable generalization capabilities of this tool. An additional significant improvement

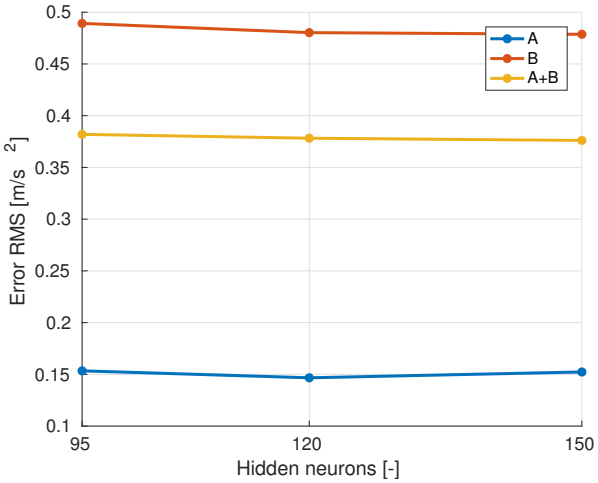
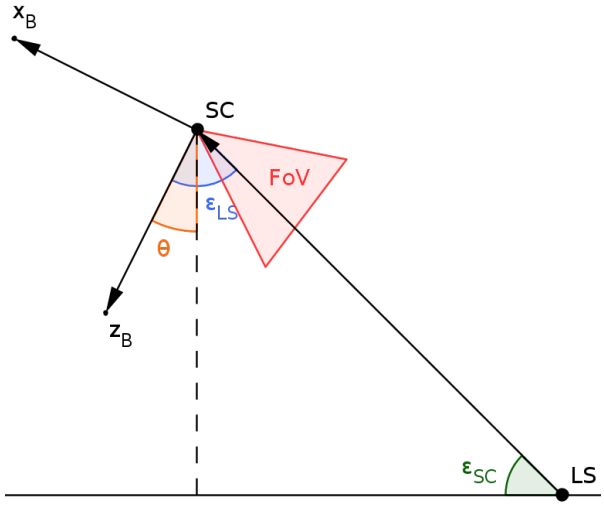
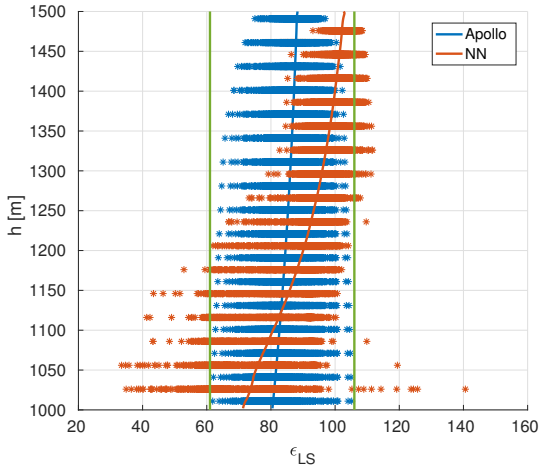


Figure 7.3: Comparison of supervised learning RMS errors for different network sizes, with the BL dataset (A), RT dataset (B), and the two datasets together.



(a) Two-dimensional semplification of the lander-LS relative elevation angles.



(b) Comparison of distributions over altitude of LoS elevation angle with respect to $\mathcal{B}$-frame (ϵ_{LS}) for the Apollo guidance and two different NNs.

Figure 7.4: Pointing geometry for the computation of ϵ_{LS} and distribution profiles of this variable for NN B and the reference Apollo guidance.

has been obtained by using a hybrid guidance approach, as will be introduced in Section 7.1.3. The results obtained from training the network in a RT-only scenario (that is, no PCs) are reported in Section 7.2.1, with the aim of answering research question RQ-2.4. We will now move on to the next section to present the background and implementation of the PCs.

7.1.2. POINTING CONSTRAINTS

As mentioned in Section 2.2.2, the enforcement of PCs is only looked into for altitudes higher than 1000 m, that is, before the RT takes place. If successful, this would allow the HDA system to scan the surface area surrounding the LS, detect any hazards, and command the lander to steer clear from unsafe areas. Besides, it has also been pointed out that a sufficient time would be available for these operations to be performed, as the altitude of 1000 m is usually reached in $\sim 6-7$ s.

Under these conditions, out-of-plane state position components are small as compared to along-

track position and altitude values. As such, a two-dimensional simplification becomes possible for the computation of the LS elevation angle as seen in the lander's $\mathcal{B}$-frame. Figure 7.4a shows the corresponding geometry. An arbitrary camera FOV is shown in red, the pitch angle θ is marked in orange, whereas the relative elevation angles ε_{LS} and ε_{SC} are shown in blue and green, respectively. The variables θ and ε_{SC} are both known functions of the trajectory. The computation of the former has been presented in Section 4.6, while the latter is obtained as $\varepsilon_{SC} = \arctan\left(\frac{\|z\|}{\|x\|}\right)$ (given the above-mentioned two-dimensional simplification). It is then possible to compute the LS elevation angle as seen from the lander as

$$\varepsilon_{LS} = \pi/2 + \theta - \varepsilon_{SC} \tag{7.1}$$

By using this formulation, the distribution of the variable ε_{LS} is computed for two different MC runs with 1000 shots. Figure 7.4b compares such distributions as a function of altitude ($h > 1000\,\text{m}$) for the Apollo guidance and NN B (see Table 6.1). For compliance with requirement GR-05, the LS needs to be visible (that is, within the optical sensor's FOV) at all times in the regarded altitude range. As such, the camera FOV needs to be at least as wide as the difference between minimum and maximum ε_{LS} values over all the MC shots, that is, $\Delta\varepsilon_{LS}$. This maximum span, or variance, can be seen to be $\sim 105^{\circ}$ for the neural network, while it is considerably more contained at $\sim 45^{\circ}$ for the benchmark Apollo guidance, as shown by the vertical green lines which enclose all the Apollo trajectories. On the other hand, one can see that, beside a larger variation in mean value, the NN trajectories also show a number of outliers, which greatly increase the span between minimum and maximum values. This represents a problem for the NN guidance, as larger FOVs require more expensive instrumentation and result in lower image resolutions. While not all the NNs trained so far show such a large $\Delta\varepsilon_{LS}$, it is desirable that this quantity is not left up to the randomness of the optimization procedure, but rather directly constrained.

To optimize the NN with respect to this constraint, an additional term is added to the computation of the reward function. However, this is not computed for each trajectory like the position, velocity and propellant components, but rather at the end of the set of simulations at each loop, and it is added to the cumulative reward directly. This is because, as said above, the quantity to be minimized is the maximum ε_{LS} span over the entire set of trajectories within the desired altitude range. As such, the computation of the reward is presence of pointing constraints can be summed up as follows:

$$r_i(\mathbf{r}_f, \mathbf{v}_f, m_f) = -k_r f(\mathbf{r}_f) - k_v f(\mathbf{v}_f) - k_m f(m_f) \quad \text{for } i = 1, \dots, N \tag{7.2}$$

$$\rightarrow R = \sum_{i}^{N} r_i - k_{PC} f(\Delta\varepsilon_{LS}) \tag{7.3}$$

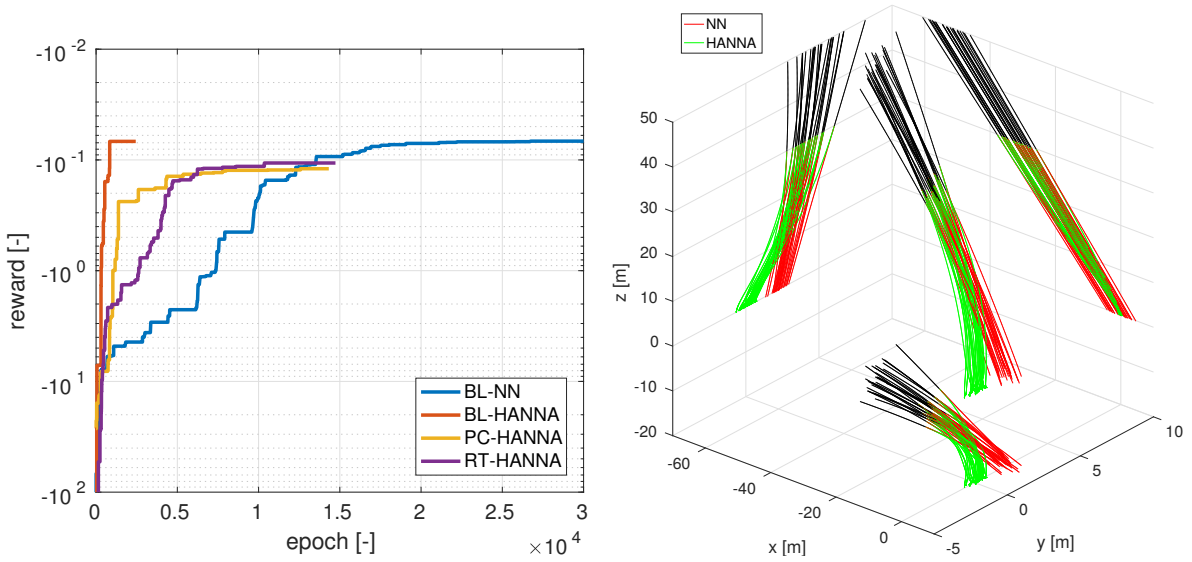
where the total ε_{LS} span $\Delta\varepsilon_{LS} = \varepsilon_{LS}^{max} - \varepsilon_{LS}^{min}$ is scaled as usual according to

$$f(\Delta\varepsilon_{LS}) = \left(\frac{\Delta\varepsilon_{LS} - c_{PC}}{d_{PC}}\right)_{PC}^{m} \tag{7.4}$$

The advantage of expressing the PC this way is that the camera elevation in the $\mathcal{B}$-frame remains free and can be easily chosen *a-posteriori* so as to maximize LS visibility for a given FOV width. On the other hand, the disadvantage is that, if the guidance system is to be optimized on a higher system level, this method provides no framework for the specification of the mounting angle of the optical instrument, which could in reality be constrained by vehicle structure requirements. However, for the purpose of this work, this factor will be disregarded, and the boresight elevation angle with respect to the $\mathcal{B}$-frame will be assumed free.

7.1.3. HYBRID APOLLO GUIDANCE & NN APPROACH (HANNA)

Although it is theoretically possible to optimize the NN as is with respect to RTs and PCs, it proved hard to achieve good convergence in any reasonable amount of time. While some acceptable results



(a) Comparison of convergence speed between pure-NN and HANNA approaches.

(b) Funneling effect of switching over to the Apollo guidance.

Figure 7.5: Convergence speed improvement and funnelling effect of the HANNA method.

were obtained for the RT-only scenario, when including both elements (RTPC, or full, scenario) convergence could not be achieved at all. This is because both the introduction of an additional term in the reward function and the extension of the state space optimization domain have the effect of greatly shrinking the regions of the NN parameter space which yield successful trajectories.

However, by combining the NN policy with the Apollo guidance, it was possible to solve this problem, while achieving a number of other advantages as well. This hybrid approach, referred to as Hybrid Apollo guidance and NN Approach (HANNA) hereinafter, is very simple, and consists in following the NN policy up until an arbitrary altitude h_{switch} above the surface, and then switching over to the Apollo guidance until landing on the surface. Another switch trigger could be used as well. However, this possibility has not been investigated for this study, and a constant value of $h_{switch} = 30\,\text{m}$ proved to work well enough. The rationale behind this choice will be given later in this section.

By looking at Figure 7.5a, one can see the convergence speed improvement that could be achieved with this method. As the figure shows, the number of iterations required to achieve a good convergence is reduced approximately by a factor 15 from a BL-NN to a BL-HANNA optimization. This refers to a scenario with neither PCs nor RTs. However, even when these elements are added to the optimization environment, the number of iterations required to optimize the HANNA policy is still considerably lower than that required to train a pure-NN guidance (that is, the NN method which had been used so far) for the simpler BL scenario. The BL-NN shown in the plot refers to NN B (see Table 6.1), which had been one of the lenghtiest optimization instances. However, even considering more average cases of about 20000 epochs, the HANNA guidance brings about a significant advantage in terms of optimization time.

This improvement in convergence speed is possible because the Apollo guidance acts as a funnel for the NN trajectories, as can be seen in Figure 7.5b, which shows a set of 50 trajectories. The black lines represent the part of trajectory in common between the two methods (pure-NN and HANNA), which result from following the NN policy. However, for certain sets of NN parameters, following this policy all the way to the surface would produce subaccurate trajectories. In fact, as shown in red in the figure, these trajectories show position residuals of a few meters at TD, likely associated to

Table 7.1: Comparison between optimized NN and Apollo guidance under different environmental conditions.

		APOLLO	NN	HANNA
$\|\Delta r\|$ [m]	μ	0.6266	0.5100	0.4561
	σ	0.3164	0.2515	0.2140
	WORST	1.6870	1.5150	1.1030
$\|\Delta v_h\|$ [m/s]	μ	0.09639	0.1052	0.09824
	σ	0.04247	0.05271	0.04271
	WORST	0.2226	0.2926	0.2270
v_z [m/s]	μ	-0.4541	-0.4229	-0.4257
	σ	0.0186	0.04467	0.02009
	WORST	-0.5193	-0.5582	-0.5148
Δm_{prop} [kg]	μ	258.0	243.6	236.3
	σ	9.820	6.481	5.167
	BEST	234.0	226.2	220.6
	WORST	314.1	270.5	258.1

velocity errors, not visible here. Switching over to the Apollo guidance at a low altitude can prevent this, as these are trajectories are channeled to a safe and accurate TD, as shown by the green lines. For this particular network, the channelling effect occurs most notably in the cross-track direction, as can be seen from the projections on the side planes, whereas for other networks the downrange corrections might be the dominant ones. Whatever the case, this considerably increases the regions of the parameter space which result in successful trajectories, thus relaxing the requirements on the stochastic search of such space.

The HANNA approach is therefore able to speed up the optimization by switching to the Apollo guidance during the last part of the descent, while retaining the advantages that the NN offers during the previous, most significant, part of the descent. Besides, the introduction of the HANNA method brings about a number of further benefits. The most important are:

- increased position and velocity accuracy;
- improved propellant optimality;
- improved robustness against navigation noise in the final part of the descent.

The latter will be addressed in Section 7.2.4, when the matter of limiting attitude rates will be discussed. As for position, velocity and fuel performance, Table 7.1 shows a comparison between MC results with 1000 runs each. One can see that the HANNA guidance outperforms both NN and Apollo guidance in terms of position accuracy and propellant consumption, and equals the Apollo guidance in terms of velocity accuracy, doing better than the pure-NN strategy. When it comes to position and velocity, the improvement is achieved because the Apollo guidance, when run in closed-loop, computes its commands by solving the TPBVP at every time step, thus always meeting the desired final conditions, given a feasible initial condition. It therefore seems that the NN here serves the purpose of bringing the lander to a set of interface states, at 30 m altitude, which are more within the feasibility bounds of the Apollo guidance than following a pure-Apollo policy would do. Hence, the improvement in position accuracy as compared to the traditional Apollo, which the NN, by itself, was not able to achieve. Not only does the HANNA improve mean values of position and vertical velocity, but also, most noticeably, standard deviation and worst case value for position. Most of, if not all, the remaining residuals at TD are due to navigation uncertainties and to the few seconds spent in open-loop by the Apollo guidance to avoid the t_{go} singularity, as already discussed in Sections 4.5 and 6.2.2.

Moreover, considerable reductions in propellant consumption are present as well. Again, the HANNA method improves both mean, standard deviation, best and worst case value. Most importantly, the mean is ~ 7 kg lower than the NN, which was already clearly outperforming the Apollo guidance, whereas the worst case value reduction is even larger at about 12 kg. This phenomenon stems from a different reason than the one that caused the improvement in accuracy. In fact, it was observed, for many different trajectories and across a number of NNs, that the quantity

$$\Delta t_{go} = t_{go}^{(apollo)} - t_{go}^{(real)} \tag{7.5}$$

often shows a local minimum during the final part of the descent (and the minimum value is often negative), depending on the network. Here, $t_{go}^{(apollo)}$ is the time-to-go as predicted by the Apollo guidance logic, while $t_{go}^{(real)}$ is the real time-to-go resulting from following the NN policy all the way to the surface. Figure 7.6 shows this graphically for a set of trajectories and two different neural networks, namely NNs A and B, presented in Table 6.1. In the left-hand side plot, the minima occur at higher altitudes (approximately 30 to 80 m) than they do in the right-hand side plot (circa 10 to 20 m). Also, the minimum value changes: it is always negative and between 1 and 6 s for the network on the left, while it is sometimes positive, yet mostly negative and up to 4 s, for the network on the right. This explains why switching over to the Apollo guidance in this altitude range brings about propellant savings, as the remaining time of flight is shorter. It can therefore be said that, for this very specific portion of the descent, the Apollo guidance is more fuel optimal than the RL-optimized NN. This is because of the same specialization vs. generalization dilemma that the RL optimization method faces, which has been mentioned in Section 6.2.1. To optimize the NN to be fuel optimal in a global sense, that is, when considering the descent as a whole, optimality in this last part is more than offset and thus needs to be traded off.

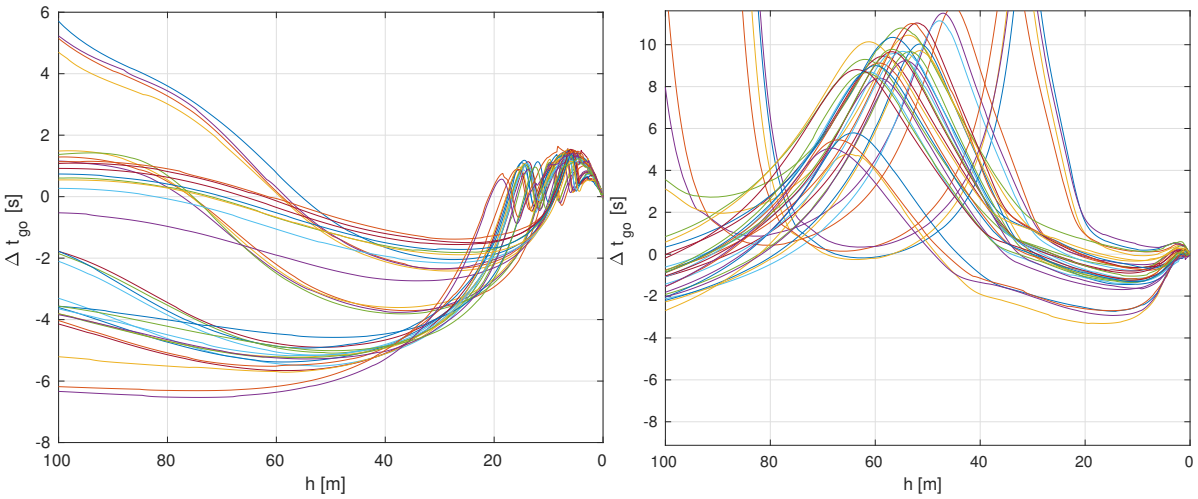
In principle, this minimum would be the optimal trigger for the switch from NN to Apollo guidance, as far as propellant consumption is concerned. However, this minimum is hard to identify in real-time, or even autonomously *a-posteriori*, and its existance is of course not guaranteed. As such, a fixed altitude value of 30 m is chosen that makes for a reasonable compromise between the altitudes at which the minima have been observed. Besides, other criteria would actually underpin the choice of the trigger in a real case scenario, such as interface attitude, so to minimize attitude discontinuities between the pre- and post-switch policies. However, these investigations go beyond the intended aim of the introduction of this method. They are thus not investigated further, but further work in this direction will be recommended in the concluding Chapter 8.

This choice has other motivations as well. In fact, as it will be recalled in Section 7.2.4, the original purpose of the HANNA method was that of limiting attitude rates due to navigation uncertainties during the final part of the descent. It does seem like the gradient of the NN output becomes much larger at lower altitudes, as these policies are easier to find than more fuel optimal ones, while still achieve accurate landings. It was noticed that choosing switch altitude values lower than 30 m did not completely remove peaks in attitude rate cause by navigation noise. This will be further investigated in Section 7.2.4. Ultimately, being so inhomogenous, these NN policies also result in longer times-to-go than the optimal or even the Apollo guidance, thus increasing propellant usage in this portion of the process.

7.1.4. MODIFIED MEASUREMENT MODEL

In Section 5.4.2, a simple measurement model was presented, which consisted in a multiplicative noise applied to all state components. This has been used for the optimizations that led to the results presented in Chapter 6. However, this proved to be non-representative of the environment in which the agent should then operate. As such, a measurement model better fit to match the output of a Kalman filter has been implemented.

First, as part of the attempts made to limit the attitude rate (see Section 7.2.4), the measurement



(a) NN A. (b) NN B.

Figure 7.6: Local mimimum of the difference in computed and real time-to-go.

noise frequency has been reduced to 1 Hz, as this is the frequency at which non-inertial measurement come into the filter during the process. In between these updates, a zero-order hold is applied.

Besides, the parameters of the random sequence, and their evolution over time, have also been adjusted. The measurements are now expressed as follows.

$$\mathbf{x}_{meas} = \mathbf{x}_{real} + N(0, \boldsymbol{\sigma}_0) \left[\left(\frac{h}{h_0} \right)^{h_{decay}} + \frac{\boldsymbol{\sigma}_{SS}}{\boldsymbol{\sigma}_0} \right] \tag{7.6}$$

where $\boldsymbol{\sigma}_0$ are the standard deviation components of the initial dispersion, $\boldsymbol{\sigma}_{SS}$ are the standard deviation components of the expected steady-state state estimation error, h and h_0 are current and initial altitude and h_{decay} is a tuning parameter to adjust convergence speed to the steady-state value.

This model produces an output which better resembles that of the Kalman filter used in the full simulation environment, and, therefore, avoids to overtrain the NN against unrealistically large measurement perturbations, such as those used for the first part of the optimization campaign. Figure 7.7 shows the time series of measurement errors for one sample trajectory. One can see that convergence is achieved after approximately 8-10 s, which is slightly larger than it takes the Kalman filter to do so. This has been done to provide some margin for robustness, and was achieved with a decay rate value of $h_{decay} = 8$. Steady-state noise and initial dispersions have also been amplified with respect to the navigation system values for the same reason. This can be seen by comparing these plots with those of Figure 4.18.

The implementation of this model might have the slight effect of reducing the network's robustness against navigated errors, but has the advantage of reducing navigation noise-induced attitude rate peaks, which have been first introduced in Section 6.2.4 and will be further discussed in Section 7.2.4. Besides, it is also expected to contribute in speeding up the optimization process, as the range of environmental conditions the network must be trained for is now reduced.

7.2. RESULTS

In this section, the optimization and simulation results for different scenarios will be presented. In Section 7.2.1, the network is only optimized with respect to LS redesignations (RTs), and is tested both with and without these divert maneuvers during the actual process. In Section 7.2.2, the networks are trained with no RTs, but PCs are included in the reward function. Again, in Section 7.2.3,

both RTs and PCs are part of the optimization process, and the networks will be subsequently tested both with and without divert maneuvers. Section 7.2.4 analyzes attitude residuals at TD and discusses the work that has been done to limit attitude rates. Finally, 7.2.5 presents an analysis of the converge speed and properties of the algorithm.

7.2.1. RT SCENARIO

As has been done for the BL scenario in Section 6.2.1, propellant optimality will first be looked into. Afterwards, stress cases will be tested to probe the algorithm's robustness against variations in RT points.

PROPELLANT OPTIMALITY

This section will address research question RQ-2.4.2, as propellant optimality and accuracy in the RT scenario will be discussed.

Table 7.2 compares propellant consumption values for a number of different networks. The results obtained by using the pure-NN approach, as obtained prior to the introduction of the HANNA method, are also given as reference, as well as the GPOPS optimal values. The two HANNA networks, on the other hand, are obtained by optimization in a BL environment (BL-HANNA) and in a RT scenario (RT-HANNA), respectively. The rows relative to the latter network, object of this section, are highlighted. The second column refers to whether RTs were included in the simulation environment used for testing the networks, whereas the last column shows the propellant optimality factor with respect to the optimal value. For the tests including RTs, the corresponding optimal value is used to compute the optimality factor, similarly to Table 6.2. All the results reported in the table are obtained from simulations in ideal environments, as this provides the best solution for benchmarking against the GPOPS values. Finally, one can see that the RT-HANNA network, when simulated with RTs, produces some failures (namely, 23 cases or 1.9% of the total), as mentioned in the footnote. It needs to be pointed out that these simulations represent stress cases, featuring worst cases for both ICs and RT points. The robustness of the algorithm against these kind of situations will be better analyzed in the next section while commenting Figure 7.9.

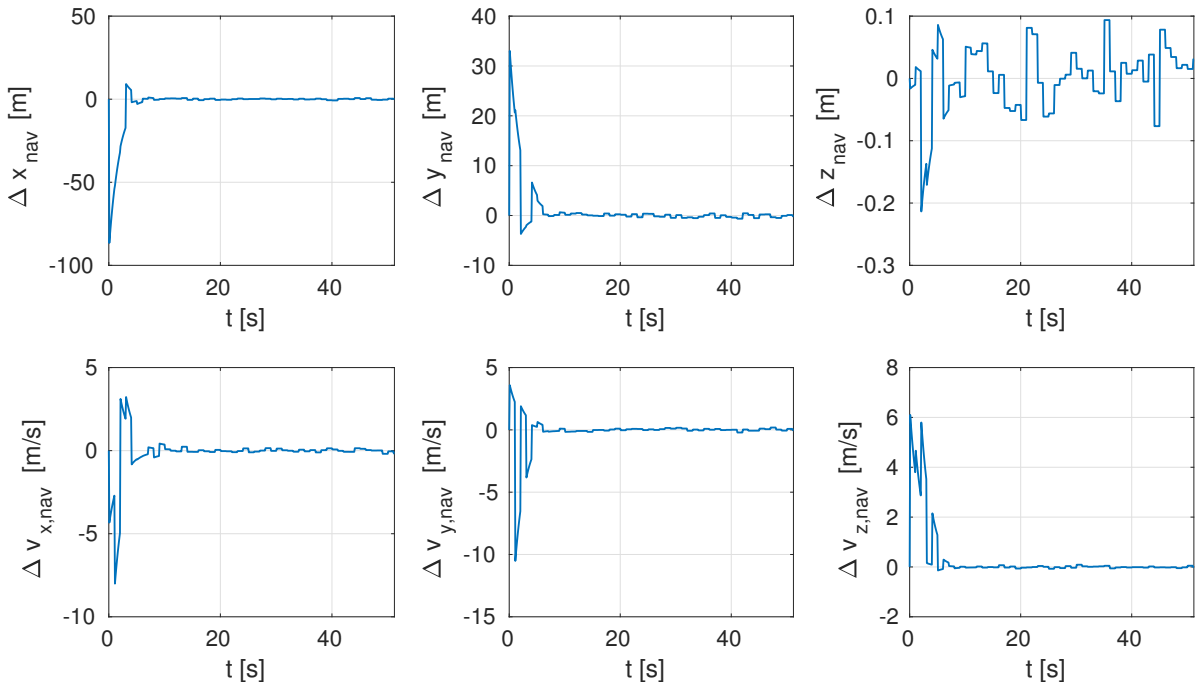


Figure 7.7: Measurement error for a sample trajectory as obtained from the updated measurement model.

Table 7.2: Propellant optimality comparison between BL-NN, BL-HANNA and RT-HANNA.

NETWORK		m[kg]				$\frac{m}{m^*}$
TRAINING	TEST	μ	σ	BEST	WORST	
BL-NN	w/o RT	248.6	13.06	227.1	278.6	1.21
BL-HANNA	w/o RT	241.3	15.85	211.4	327.0	1.17
RT-HANNA	w/o RT	256.4	12.02	231.3	286.2	1.25
	w/ RT[1]	279.0	25.76	222.3	342.4	1.22
GPOPS	w/o RT	205.6	15.02	175.0	240.3	-
	w/ RT	227.9	25.20	169.7	288.6	-

Also, one can see that the mean propellant consumption value is lower for the HANNA network than it is for a pure-NN approach, in agreement with Table 7.1. However, the worst case value is actually higher, while the values shown in Table 7.1 report a lower worst case fuel consumption for the HANNA network, even though the two tables refer to the same networks. That is because, while Table 7.1 shows the results of MC simulations with Gaussian ICs, Table 7.2 reports, as already mentioned, values obtained from worst case ICs. It could therefore be that, since the HANNA network has been optimized for a much shorter time, as shown in Figure 7.5a, it is still suboptimal in specific regions of the state space. On the other hand, the pure-NN policy, despite not making use of the Apollo guidance to increase accuracy in the final meters of descent, has undergone a more extensive optimization, and is therefore robust against a larger set of conditions. To verify whether the newly trained HANNA networks suffer from a lack of robustness because of shorter optimization times, also networks not specifically trained against RTs will be included in the stress cases aimed at verifying sensitivity with respect to RTs and ICs (next section and Section 7.3.1, respectively).

On the other hand, when the HANNA network is optimized accounting for RTs, but simulated without, the mean propellant consumption increases, while the worst case one decreases. Again, this seems to be in agreement with the hypothesis formulated in the previous paragraph, as this network was optimized for a longer time than BL-HANNA (see Figure 7.5a). Even though the training was not performed for these specific conditions (as RTs were included and, therefore, different state space regions were visited), it seems to be beneficial to the inner areas of the state space as well. As usual, a role in this could also be played by the random nature of the optimization, whose effect is hard to identify. On the other hand, the network suffers from large propellant consumption increases, for both mean and standard deviation values, when divert maneuvers are included in the simulation. This is expected, as is the improvement in optimality factor (last column), consistently to what was seen in Sections 6.2.1 and 6.2.2. In fact, the network performs better, in a relative sense, when simulated under conditions more similar to the ones it has been specifically trained for.

Fuel optimality, although still larger than the original requirement GR-04, given values between 1.17 (BL scenario) and 1.25 (RT scenario), can be still considered within reasonable bounds. In fact, the target value of 10 % mass penalty had been set by making an optimistic guess of what could be achieved based on the results obtained by Gaudet and Furfaro [2014]. However, it must be pointed out that, although RTs are included in the reference study, they are only tested for areas of the state space which naturally fall within the BL domain (see central trajectory batch in Figure 7.2), for which fuel penalties between 12 % and 18 % could be achieved. Considering that this work, on top of an extended state space domain, includes an additional dimension and features a lower sampling resolution in the RL environment, the propellant values obtained are deemed more than satisfactory.

Beside bringing about propellant savings, the introduction of the HANNA framework makes it such that position and velocity performance is unlikely to degrade gradually. A trajectory would ei-

[1] Results obtained after removing 23 failures.

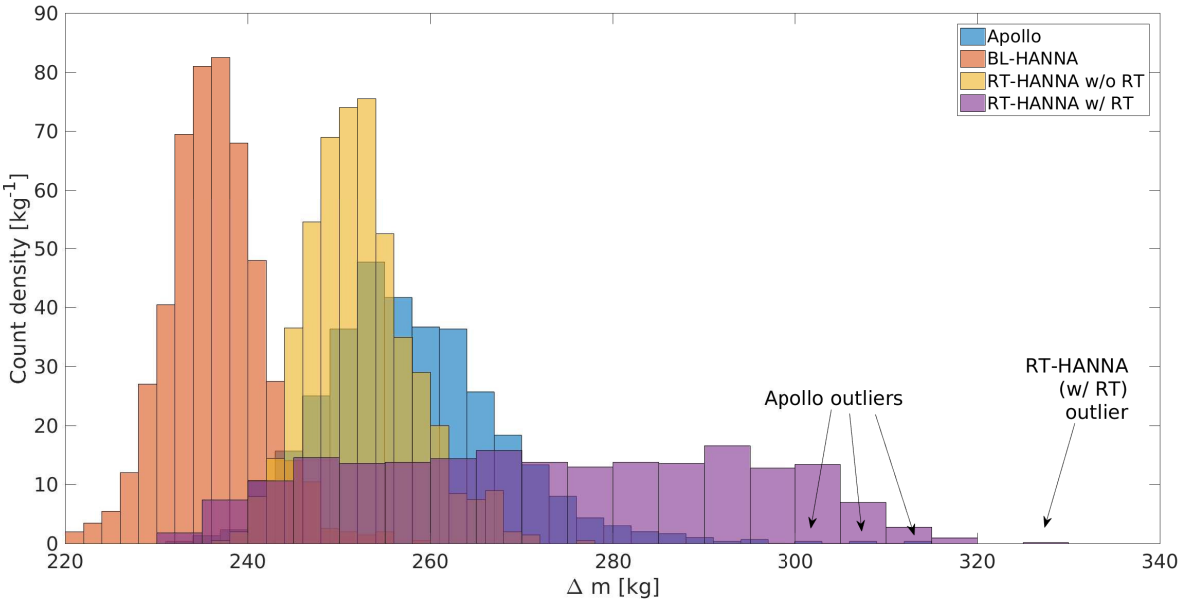


Figure 7.8: Comparison of propellant consumption between Apollo guidance, BL-HANNA and RT-HANNA, with and without RTs.

ther end with a safe and accurate TD, or it would fail altogether. Inaccurate but non-catastrophic landings are still possible, yet unlikely. As such, if a MC simulation does not result in any failure, as is the case when simulating the RT-optimized HANNA network object of this section for a set of Gaussian ICs, the final values will look very much like those presented in Table 7.1. The difference lies in the mass consumption, and attitude residuals at TD, as discussed in Section 7.2.4. As such, accuracy comparisons of MC simulations will be hereinafter omitted, as the focus will be on mass consumption, as well as on the additional metrics made relevant by the new constraints enforced in the next sections (i.e., attitude and attitude rates). In this regard, Figure 7.8 shows the fuel consumption comparison between the HANNA networks reported in Table 7.2 and Apollo. This time, however, the simulations have been carried out for random ICs and RTs. One can see that both HANNA networks, when simulated without RTs, produce distributions with lower standard deviations and mean than those of the Apollo guidance. When RTs are included, no failures are present, opposite to what is shown in Table 7.2, although one outlier in terms of propellant consumption can be seen above 320 kg. This is resposible for raising the worst case value, as is the case for the Apollo guidance distribution. Similarly to what has been already said, the network could be made more robust in terms of mass consumption by means of more extensive optimizations with thicker sampling grids.

RETARGETING FAILURE MAPS

In this section, robustness against varying RT distances and directions will be investigated, with the aim of answering questions RQ-2.4.3 and RQ-2.5.3.

Similarly to Figure 7.1, the plots shown in Figure 7.9 are obtained with the same IC grid used during the optimization process, while performing, for each IC, RTs to a 18×14 grid of 252 points on the surface. The colorbar shows the failure rate γ, where a value of zero means that the landing is successful for all ICs, while a value of one indicates that, regardless of the IC, the trajectory always fails. Nominal landing site and target divert area are marked with the red asterisk and circle, respectively. For reference purposes, the first row shows the same failure maps already presented in Figure 7.1, while the left-hand side plot of the second row shows the failure map for the Apollo guidance. Finally, the bottom right plot presents the failure map for the RT-optimized HANNA network object

of this section, while the remaining two maps refer to HANNA networks optimized for the BL and PC scenario. The bulk of results for the latter will be presented in more detail in Section 7.2.2.

These two network are included here to verify whether the hypothesis formulated in the previous section is valid, that is, whether HANNA networks not specifically optimized to handle RTs suffer from a deterioration in robustness with respect to their pure-NN counterpart, given the shorter search of the parameter space (see Figure 7.5a). As such, these two maps are meant to be compared with the first row plots. The four maps can be immediately identified as the most similar to one another among the six shown in the figure. From a first visual comparison, there does not seem to be a significant loss of robustness against state space domain variations, as the failure maps do not show considerable differences in terms of reachable surface area. The PC-HANNA network differs from the other three in that failure rate increases more rapidly in the positive downrange direction. However, the other maps show the opposite behavior, which seems to be the prevailing trend, as already discussed and motivated in Section 7.1.1. Finally, some of them show asymmetries in the cross-track direction. Once more, this is due to the randomness of the optimization, as has been argued before. Ultimately, given no consistent performance loss against RTs for these network, this random nature must also be the reason behind the failures shown in Table 7.2, from which the hypothesis discussed in this paragraph had first stemmed.

Additionally, one can compare these plots to the Apollo guidance failure map, which differs substantially in gradient and magnitude. First, one can see that only for the largest backward diverts (i.e., negative downrange) is the Apollo guidance able to achieve a failure rate of $\gamma = 0$. This is because, by shortening the downrange distance to be traveled, the necessary downrange thrust decreases, thus allowing for a more efficient allocation of the available thrust in the vertical direction, therefore mitigating the control constraint limitation of Apollo. It is also possible to identify a trend in the Apollo guidance robustness against RT point location, as the failure rate increses with incresing downrange retargeting distance, as well as with crossrange distance. The latter is most noticeable for short RTs than it is for long ones, as the downrange component starts then becoming larger, thus dominating the effect on the value of γ. One can also see that, contrary to the NNs, a failure rate of unit value does never occur. This can be explained by a rationale similar to the one given during the discussion of Figure 6.10. In fact, when the ICs are more benign (i.e., small negative vertical, and large positive downrange velocities), the Apollo guidance has proved to be able to always achieve safe landing at the nominal LS. For off-nominal LS locations, this robustness can be affected positively or negatively, as seen above, yet some ICs always allow for safe landings within the tested surface area.

Finally, the bottom-right plot shows the reachability map of the RT-optimized HANNA guidance. One can see immediately see that the result is positive, as a much larger area than originally intended is reachable from all ICs, consistently showing zero failure rates. This is a consequence of the generalization of the network policy to state space areas outside the training one. On the other hand, some of the LS locations which have been targeted during the optimization procedure (i.e., $(x, y) = (-450, -800)$ m) show failure rates different than zero. This can be explained in the following way. Due to computational reasons, only one RT point was chosen for each of the trajectories simulated during the RL optimization, so as to maintain an equal number of total simulations to perform at each loop. The RT location for each trajectory was chosen so as to bound all the potentially encountered states, for the given RT area. In simpler terms, the largest backward RTs were assigned to the trajectories which result in the largest downrange position at the RT interface, and vice versa. An analogous procedure was followed for the crossrange direction. However, in spite of bounding the worst case state space conditions and limiting computational times, this approach brings about other drawbacks. First, it further decreases the already sparse sampling density (only two bins per state component are used, see Section 5.4) at altitudes lower than the RT one. Consequently, this creates "gaps" in the known domain of the state space, as certain combinations of position and ve-

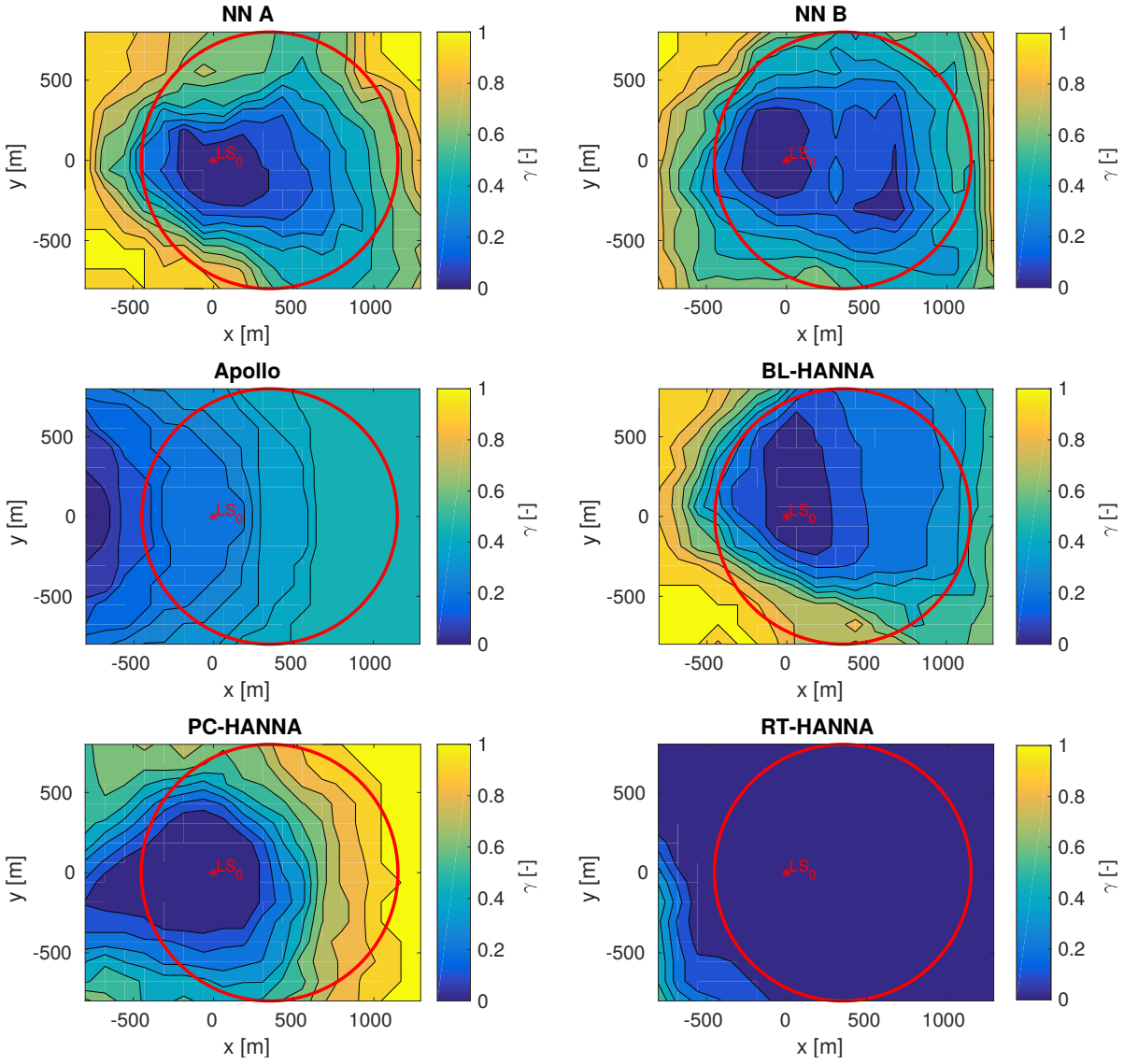


Figure 7.9: Failure maps for different networks and Apollo guidance. Nominal landing site and target divert area are shown in red.

locity, although bounded by sampled trajectories, have never been visited during training. This has the ultimate effect of producing the failures visible in the map during the testing procedure. However, these $\gamma \neq 0$ areas are outside the original target area, marked by the red circle, and can be easily avoided as long as their location is known. Besides, the ultimate reachable area is still much larger than originally aimed for. Therefore, this test shows that the optimization of the network against LS redesignations was very successful.

7.2.2. PC SCENARIO

This section presents the results obtained from the optimization of a HANNA network with respect to PCs, that is, the maximum span of LS elevation angle ε_{LS} (see Section 7.1.2) is here minimized during training. Again, propellant optimality results will be presented first, followed by an analysis of the attitude constraining results.

PROPELLANT OPTIMALITY

This section addresses research question RQ-2.5.2, by assessing propellant optimality of the PC-HANNA network. The results are shown in Table 7.3, where some of the information is included for reference purposes and is redundant to what has been discussed in the previous section. It will be therefore not treated in detail, and the focus will be given to the network object of this section, as highlighted in the table. What these data most importantly show is that the introduction of PCs does not significantly impact propellant consumption. One can see that the mean value is only slightly higher than it is for the BL-HANNA network, with a propellant optimality factor of 1.20, which is in line with what was obtained before for networks trained for the BL scenario. In particular, the PC-HANNA network is not only more fuel efficient, but also faster to converge than a pure-NN optimized for the baseline (see Figure 7.5a). Finally, the standard deviation value is even lower than it is for the BL-HANNA network. This number, however, is not fully explanatory of the fuel consumption distribution, which is better visualized by the histogram plot in Figure 7.10a.

The figure shows the propellant consumption distribution for MC simulations with 1000 shots, for the HANNA networks shown in Table 7.3 and the Apollo guidance. One can immediately see that the PC-HANNA distribution does indeed have a low standard deviation, but also features a non-Gaussian shape. In particular, a second mode can be seen at values of Δm larger than $\sim 249\,\text{kg}$. It has been verified that this is produced by large initial negative vertical velocities, as shown by Figure 7.10b. The blue bars show the amount of vertical velocity samples which yield mass consumption values larger than the selected threshold, while the yellow bars refer to the remaining ones. One can see that most of the trajectories starting from negative vertical velocities larger than $\sim 75\,\text{m/s}$ fall into the first category. As the corresponding remaining IC components are more normally distributed throughout their value ranges, large negative vertical velocities are identified as the cause of the second mode of the distribution shown in Figure 7.10a. One can therefore conclude that, even though the network has been optimized for these IC values, the optimization algorithm relies on average values for improving the reward, and is therefore not able to predict and correct the behavior of the network for specific states or ICs. A possible solution to this issue could be that of maximizing skewness of the fuel consumption distribution obtained at each optimization loop, while still minimizing either mean or worst case value, thus implementing some regularization over suboptimal behaviors of the network under specific conditions. However, this would come at the cost of increased reward function complexity.

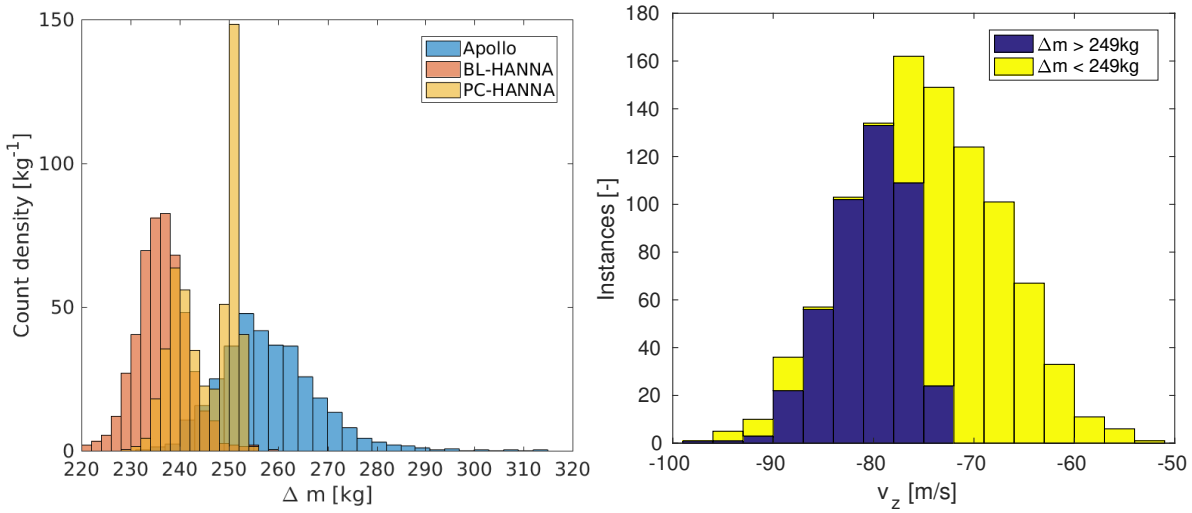
LS ELEVATION VARIANCE AND VISIBILITY

In this section, the outcome of the optimization will be analyzed with respect to the reduction of the attitude span of the vehicle during the part of descent for which PCs have been enforced, that is, above 1000 m altitude (see Section 2.2.2). As such, this section will partly answer research question RQ-2.3.

To enforce pointing constraints, one must first tune the parameters in the additional term of the reward function, as defined in Eq. (7.4). Different values for these have been tested during the optimization phase. However, this will not be discussed here, and a more detailed look into

Table 7.3: Propellant optimality comparison between BL-NN, BL-HANNA and PC-HANNA.

	m[kg]				$\frac{m}{m^*}$
NETWORK	μ	σ	BEST	WORST	
BL-NN	248.6	13.06	227.1	278.6	1.21
BL-HANNA	241.3	15.85	211.4	327.0	1.17
PC-HANNA	246.2	6.888	227.9	262.8	1.20
GPOPS	205.6	15.02	175.0	240.3	-



(a) Comparison of propellant consumption between Apollo guidance, BL-HANNA and PC-HANNA.

(b) Initial vertical velocities associated with the second mode in propellant consumption distribution.

Figure 7.10: Mass consumption anomalies, and their cause, of the PC-HANNA network.

the effect of these variables will be given in Section 7.2.3, as more optimization trials have been run for the full scenario presented in that section. In fact, for the RT and PC scenarios, not many optimizations have been performed, as an acceptable solution could be found within just a few attempts. The following results refer to a network obtained with $c_{PC} = 30^o$, $d_{PC} = 5^o$, $k_{PC} = 0.01$ and $m_{PC} = 4$, as these are the values which yielded the best optimization result in terms of both propellant consumption, discussed in the previous section, and LS elevation angle span.

Table 7.4 reports the minimum and maximum values of ε_{LS}, together with its total span and (approximately) optimal camera mounting angle ε_{FOV}, for different networks and the Apollo guidance. The values are produced from MC runs with 1000 shots and random ICs. As already seen in Figure 7.4b, the Apollo guidance performs well in terms of $\Delta\varepsilon_{LS}$, as its commanded acceleration vector changes slowly along the trajectory. Conversely, the BL- and RT-HANNA networks, not specifically optimized in this sense, show a larger total span, yet not completely unacceptable given commonly used camera FOV widths (e.g., Rogata et al. [2007] assume a camera FOV of 70^o). While it is not uncommon to obtain networks with such values of $\Delta\varepsilon_{LS}$, many instances result in unacceptably large values when PCs are not constrained, such as in the case of NN B, as shown in Figure 7.4b (see Table 6.1 for letter tag reference).

When PCs are enforced, however, results are surprisingly consistent in terms of $\Delta\varepsilon_{LS}$, considering the random nature of the optimization. Not all of these instances resulted in good propellant consumption, but optimization of the network under these conditions can be considered relatively easy, as not much tuning effort had to be made. Out of seven optimizations run, in four cases the maximum elevation span could be limited at values around or below 50^o. In particular, for the network presented here, a value of $\sim 42^o$, corresponding to a reduction of more than 25^o with respect to

Table 7.4: Total span in LS elevation angle as seen from the lander $\mathcal{B}$-frame for PC-HANNA and other reference cases.

	$\varepsilon_{LS}^{min}[^o]$	$\varepsilon_{LS}^{max}[^o]$	$\Delta\varepsilon_{LS}[^o]$	$\varepsilon_{FOV}[^o]$
Apollo	61.76	105.2	43.47	83.48
BL-HANNA	40.62	109.0	68.41	74.82
RT-HANNA	32.73	101.3	68.61	67.03
PC-HANNA	53.97	95.68	41.71	74.83

the non-PC networks, could be achieved at almost no cost in terms of fuel consumption, as shown in the previous section.

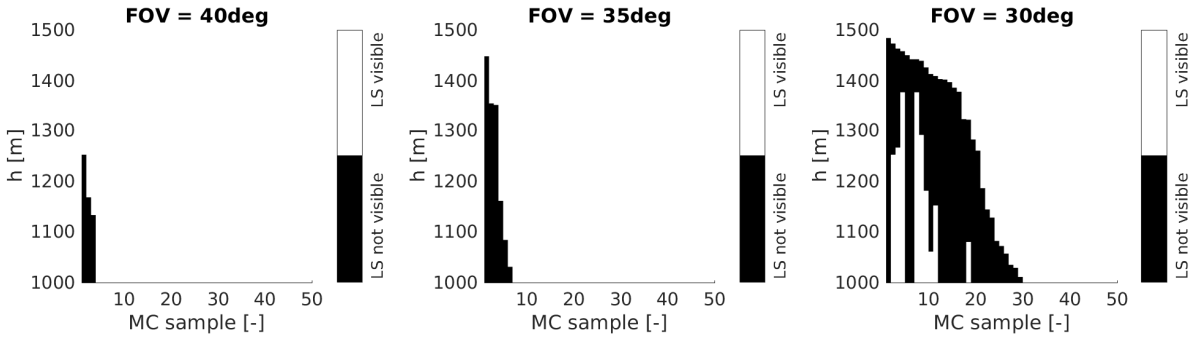
Finally, the last column of the table shows the elevation angle in the $\mathcal{B}$-frame, computed as the mean between the minimum and maximum ε_{LS} values, which would be most convenient as camera mounting angle. Also, when choosing the camera FOV width based on these data, one needs to take into account that the minimum and maximum values are not representative of the underlying distributions over the MC shots and altitude range. Different situations are possible, as the distribution could be compact across the trajectories, which is the case for the Apollo guidance in Figure 7.4b, ot it could be more sparse, with a few outlying trajectories determining the worst case values and, therefore, the value of $\Delta\varepsilon_{LS}$. However, even for a compact distribution such as that of the Apollo guidance, the issue with the plot of Figure 7.4b is that it does not provide insight into how the elevation angle ε_{LS} varies throughout the altitude range traveled by one single trajectory.

For a better visualization of these data, the same information shown in Figure 7.4b (that is, the value of ε_{ls} shown on the x-axis) can be plotted as the third variable in a three-dimensional plot, against altitude and MC samples on the horizontal axes. Then, bounds can be assumed for minimum and maximum values of ε_{LS}, as coming from the choice of camera FOV width (θ_{FOV}) and camera boresight elevation (ε_{FOV}). For instance, given the values in Table 7.4 for PC-HANNA and Apollo, let us assume three different values for θ_{FOV} (e.g., 40^o, 35^o and 30^o), while keeping ε_{FOV} fixed at 75^o for PC-HANNA and 83.5^o for Apollo. One can then check, for each point in the two dimensional space of altitude and MC sample, whether the corresponding ε_{LS} falls within these bounds or not. This produces a binary tabular dataset, whose columns (that is, MC samples) can then be re-sorted to ease readability. The result is shown in Figure 7.11.

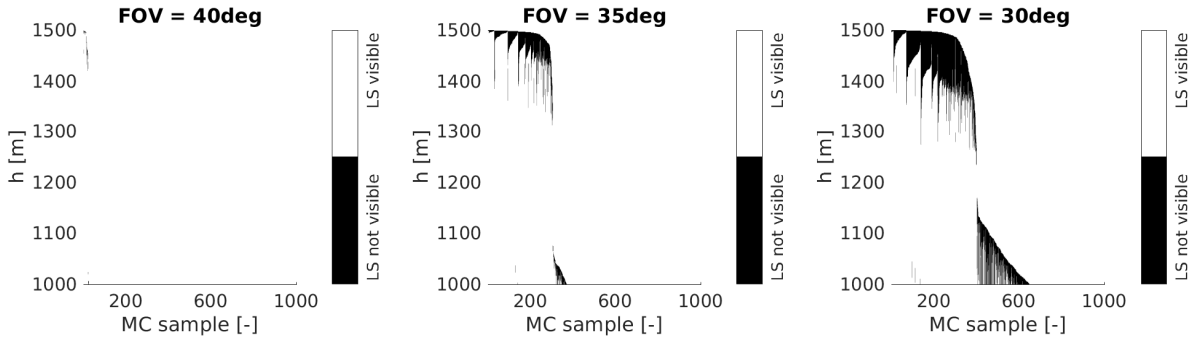
One can see that the y-axis refers to altitude, while the x-axis shows the different MC samples. Both for Apollo and PC-HANNA, a FOV of 45^o does always provide LS visibility. These plots are therefore not shown, as they would not be very informative. In Figure 7.11a, relative to the Apollo guidance, only 50 MC trajectories are taken into account, as most trajectories have a LS visibility consistently equal to 1 (i.e., LS is visible) across the entire altitude range and even for the narrowest FOV width. When $\theta_{FOV} = 40^o$, only a few of the trajectories considered lead the LS outside of the camera FOV for part of the descent. However, this can be considered acceptable, as the LS would still remain visible for 3-4 s, which can be enough to perform HDA. However, when the camera FOV width is decreased further, one can see that the number of trajectories that produce LS non-visibility increases mildly (<10 for $\theta_{FOV} = 35^o$ and about 30 for $\theta_{FOV} = 30^o$), yet the altitude span across which the LS remains not visible increses significantly for some of these trajectories. This represents a problem, as for these trajectories the HDA system would never get the chance to operate, therefore not allowing for the selection of a new LS.

On the other hand, Figure 7.11b shows the same LS visibility maps for the PC-HANNA network. Despite similar values of $\Delta\varepsilon_{LS}$ in Table 7.4, the LS visibility maps are significantly different from those relative to the Apollo guidance. As such, the entire MC sample range is shown. When $\theta_{FOV} = 40^o$, the plot shows very few occultation samples in the two dimentional space (small black features can be seen in the upper-left and lower left), as was the case for Apollo. However, the difference can be seen for wider FOVs. One can see that many more trajectories go through an occultation phase, yet this never lasts more than ~ 250 m even for the narrowest camera FOV. Similarly to the case of Apollo with $\theta_{FOV} = 40^o$, this should allow enough time for performing HDA, but this can now be done at a higher imaging resolution, therefore allowing for the construction of a finer hazard map. The comparison performed here shows the importance of doing a trajectory-specific investigation of the LS visibility, rather than considering its MC distribution.

There is more than this to the plots shown in Figure 7.11, as one should also consider at which point of the descent the occultation occurs. In fact, given the same duration of total LS visibility for a given trajectory, an occultation of a few seconds in the middle of the HDA-dedicated part of



(a) Apollo LS visibility map.



(b) PC-HANNA LS visibility map.

Figure 7.11: LS visibility maps for Apollo and PC-HANNA.

descent is not desirable from an image processing point of view, as it increases the variation in the camera extrinsic parameters (i.e., its attitude and position with respect to the $\mathcal{G}$-frame, in this case) between two consecutive observations of the same point on the surface (e.g., LS or nearby location). This kind of data representation had not been thought of at the moment of the design of the RL reward function. However, the implementation of this kind of metric in the reward is recommeded for future work: for instance, rather than elevation angle span, which proved to not be completely representative of the desired quantity, one could maximize the duration of continuous LS visibility for the worst case trajectory among the simulation batch. A way to efficiently express this metric in the reward function needs to be investigated. Finally, the plots shown here have been obtained by calculating the $\mathcal{B}$-relative LS position by means of the two-dimensional simplification presented in Eq. (7.1). An extension to three-dimensional case is also necessary for enforcing PCs after the first RT, so as to enable a second one, if needed.

7.2.3. FULL SCENARIO

In this section, the training elements included in the two previous sections are combined, as the network is optimized for accommodating both divert maneuvers and attitude constraints.

Differently than for the previous scenarios, the optimization of the network under these conditions proved quite challenging, as the issues of sampling density and optimization time became more prominent under these extended conditions. In fact, the networks often produced failures when the simulations involved RT points or state-space regions different than the ones they had been optimized for. In other cases, failures could be avoided, but only at the cost of extremely high propellant consumption. On top of this, one also needs a good balance in terms of achievable $\Delta\varepsilon_{LS}$ reduction, which greatly reduces the set of suitable results. As such, to pick the best of these networks, a compromise had to be made between presence of failures under stress conditions, mean and worst case propellant consumption values, and $\Delta\varepsilon_{LS}$ values.

Table 7.5: Propellant optimality of RTPC-HANNA. All the previously presented networks are included for comparison.

NETWORK		m[kg]				$\frac{m}{m^*}$
TRAINING	TEST	μ	σ	BEST	WORST	
BL-NN	w/o RT	248.6	13.06	227.1	278.6	1.21
BL-HANNA	w/o RT	241.3	15.85	211.4	327.0	1.17
RT-HANNA	w/o RT	256.4	12.02	231.3	286.2	1.25
	w/ RT[2]	279.0	25.76	222.3	342.4	1.22
PC-HANNA	w/o RT	246.2	6.888	227.9	262.8	1.20
RTPC-HANNA	w/o RT	276.5	14.67	248.4	320.0	1.34
	w/ RT[3]	311.3	37.00	245.3	399.3	1.37
GPOPS	w/o RT	205.6	15.02	175.0	240.3	-
	w/ RT	227.9	25.20	169.7	288.6	-

This section is aimed at answering question RQ-2.6, by investigating the performance of these kinds of networks in terms of propellant consumption, and robustness against IC and RT variations. It will also complete the answer to question RQ-2.3, started in Section in the previous section for the PC scenario, by looking at LS visibility in the full scenario. Finally, the effect of varying parameters for the PC component of the reward function is analyzed, thus addressing RQ-4.2. We will start by analyzing propellant optimality of the selected network in the next section.

PROPELLANT OPTIMALITY

Similar to the previous sections, Table 7.5 shows propellant optimality of the RTPC-HANNA network, highlighted in blue, as compared to all the previous cases, included for comparison. One can see that optimality decreases sharply when both RTs and PCs are included in the optimization. In this case, the values are too far from requirement GR-04, set in Section 2.2.1, to be considered acceptable. In particular, one can see that the standard deviation value increases substantially in presence of RTs in the testing environment, hinting at suboptimal solutions under specific stress conditions, likely due to low optimization sampling density. Also, as said above, networks with better fuel consumption could be obtained, but with non-satisfactory corresponding $\Delta\varepsilon_{LS}$. Therefore, further work is required here to obtain more optimal combinations of these metrics. One can also see that 10 failures occur when simulating the network for worst case ICs in presence of RTs. However, similar to what has been said in Section 7.2.1, this does not necessarily represent a problem, as it will be shown in Figure 7.13. Finally, despite the mediocre performance of the RTPC-network, the worst case fuel consumption value obtained from RT-free simulations is still lower than that of the BL-network, further solidifying the hypothesis of the latter being suboptimal due to short optimization times. Therefore, one should take the necessary measures in terms of mass reward component tuning when optimizing HANNA networks, by increasing the mass penalty strictness, so as to allow for a more extensive search of the parameter space.

LS ELEVATION VARIANCE

Given that more tuning was required to obtain good results for the RTPC optimization, a number of different combinations of the c_{PC} and d_{PC} parameters were tested. Table 7.6 shows the optimized values of total ε_{LS} span for different c_{PC}-d_{PC} combinations, together with the corresponding mass consumption Δm_P. For the first row, eight optimizations were run for each combination, whereas for the second row, six trials were run. Although this amount of samples is not sufficient for an ac-

[2]Results obtained after removing 23 failures.
[3]Results obtained after removing 10 failures.

Table 7.6: Mean $\Delta\varepsilon_{LS}$ and mass consumption Δm_P for c_{PC}-d_{PC} combinations.

		$c_{PC}[^o]$				
		20	30	40	50	
$d_{PC}[^o]$	2	-	105.1	89.06	102.9	$\Delta\varepsilon_{LS}[^o]$
	5	88.03	77.90	88.80	-	
	2	-	427.6	398.7	421.5	Δm_P[kg]
	5	370.4	347.5	353.0	-	

curate statistical analysis, the results are nonetheless indicative of the performance of the different parameter combinations.

The first thing one can notice is that the results are surprisingly well correlated, as parameters that yield a good propellant optimality also produce good pointing accuracy, and vice versa. In general, one can see that $k_{PC} = 5^o$ yields better results than $k_{PC} = 2^o$, in particular when $c_{PC} = 30^o$. In fact, this combination produces the best mean values for both $\Delta\varepsilon_{LS}$ and Δm_P. As such, from what can be seen from these data, there is no real trade-off to be made between the different combinations in terms of mean performance values. This is in contrast with the optimizations carried out for the BL scenario before the introduction of the HANNA method (see Section 6.1.2), where a strict mass constraint did succesfully limit the mass, at the cost of reduced position and velocity accuracy, and vice versa. The latter, however, are not considered here, as the HANNA framework ensures equal accuracy for (almost) all optimization instances[4]. What the pre- and post-HANNA optimization campaigns have in common, however, is that the choice of the best network ultimately comes down to network-specific trade-offs, as each network has different features. Many of the best results were obtained with the parameter combinations marked in green in the table. However, the one which yielded the best compromise between propellant consumption, robustness against ICs and RTs and pointing accuracy was obtained with $c_{PC} = 20^o$ and $d_{PC} = 5^o$. In the remainder of this section, this network will be analyzed with respect to pointing accuracy and LS visibility.

The ε_{LS} values for the chosen network are highlighted in blue in Table 7.7, where the previously obtained networks are included for comparison. One can see that, for the RTPC network, the value is just a few degrees lower than it is for the RT-HANNA network ($\sim 65^o$ vs. $\sim 68^o$), which has not been specifically optimized for pointing accuracy. At the same time, however, propellant consumption has increased significantly, as was shown in Table 7.5. As such, it would seem that, given the extent to which attitude span can be limited, including PCs in the optimization environment is not worth the penalty in terms of fuel usage. As was the case in the PC scenario, however, the values in Table 7.7 do not tell the whole story, and LS visibility maps need to be looked into to draw more specific conclusions regarding the pointing accuracy of these networks. Given the performance gap between the RTPC network and Apollo, the former will be here compared to the RT network, which can serve as a better benchmark for assessing the result of enforcing PCs in this specific case.

Figure 7.12 shows the LS visiblity maps for the RT and RTPC networks. For the latter, some MC samples have been left out, as they resulted in continuous LS visibility throughout the entire altitude range considered, and it was thus meaningless to include them. Similarly to Figure 7.11, these plots show that the performance gap in terms of pointing accuracy is actually much larger than it would seem from the values in Table 7.7. The leftmost plot of Figure 7.12a shows that, for the RT-network, a few trajectories go through occultation portions of a few hundreds of meters already with $\theta_{FOV} = 60^o$. Given the relaxed visibility requirements introduced in the discussion of Figure 7.11, this can still be considered acceptable. On the other hand, the corresponding plot of Figure 7.12b

[4]The optimizations for which accuracy has not been achieved are < 10 % (4 out of 42 trials run). These are excluded from the computation of the metrics given here.

Table 7.7: Total span in LS elevation angle as seen from the lander $\mathcal{B}$-frame for RTPC-HANNA and other reference cases.

	$\varepsilon_{LS}^{min}[^o]$	$\varepsilon_{LS}^{max}[^o]$	$\Delta\varepsilon_{LS}[^o]$	$\varepsilon_{FOV}[^o]$
Apollo	61.76	105.2	43.47	83.48
BL-HANNA	40.62	109.0	68.41	74.82
RT-HANNA	32.73	101.3	68.61	67.03
PC-HANNA	53.97	95.68	41.71	74.83
RTPC-HANNA	43.62	109.16	65.54	76.39

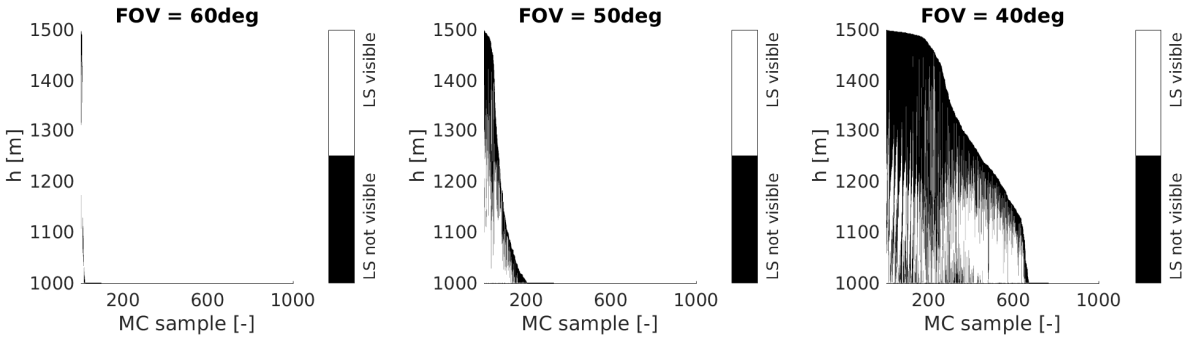
shows occultations only during the final meters before retargeting altitude, and HDA operations can therefore be successfully accommodated. However, for narrower sensor FOVs, performance degrades very differently for the two networks.

For the RT network, a FOV of 50^o produces a few trajectories for which the LS is not visible during most of the altitude range, as was the case for Apollo in Figure 7.11a. This is even more visible for $\theta_{FOV} = 40^o$. On the other hand, some of the RTPC network trajectories do incur into localized occultation periods (marked in red) with narrower cameras, which might lead to continuity problems for the image processing algorihtm, but performance degrades much more gradually for the RTPC-HANNA network. Besides, the loss of LS visibility marked in red in Figure 7.12b is believed to be caused by sudden variations in commanded acceleration, which induce changes in vehicle attitude and, therefore, camera pointing. As such, they can be remedied by solving the issue of large attitude rates (see Sections 6.2.4 and 7.2.4), thus smoothening the attitude profile. Conversely, for the RT network in Figure 7.12a and the Apollo guidance in Figure 7.11a, the issue is more intrisic to the evolution of attitude and state components over the descent, rather than to localized, sudden attitude changes. Therefore, even though the minimum viable camera FOV, from this analysis, is identified as 60^o for both networks, the plots show that the PC-specific optimization was not done in vain, as the RTPC network in indeed more robust to narrower FOVs than the RT network is.

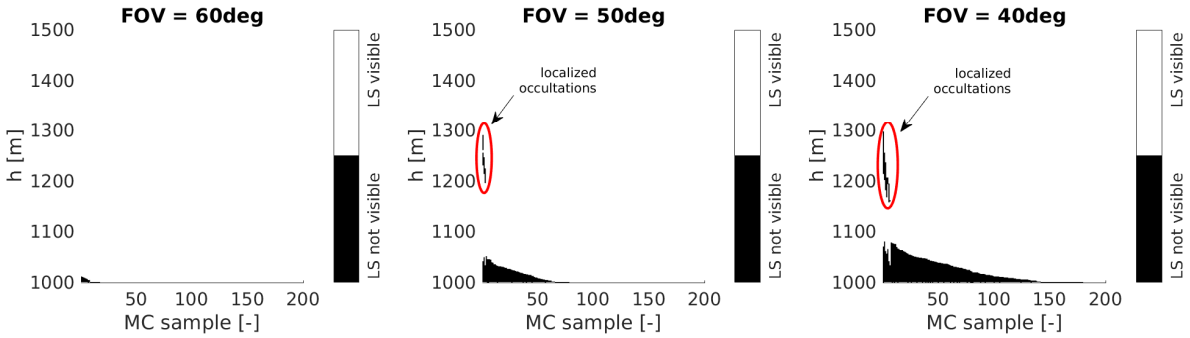
RETARGETING FAILURE MAPS

The results of the RT stress cases for the RTPC-HANNA network are presented in this section. These are only compared to RT-HANNA here. The comparison to different networks and Apollo can be found in the discussion of Figure 7.9. The failure maps for the two networks are shown in Figure 7.13. One can immediately see that the two maps are very similar, with failure rates that reach 0.6 and 0.7 for the RT and RTPC networks, respectively, and only for the worst case RT conditions. These show that the failures mentioned in the footnote in Table 7.5 do not represent a problem, for the same reasons given for the RT scenario in the discussion of Figure 7.9. Summarizing these briefly, failures are localized and can thus be avoided, and the reachable surface area is more than satisfactory, as the maps show zero failure rate (blue areas) for most of the tested surface patch, which is in full compliance with requirement GR-06.

Considering the issues regarding robustness and sampling density which have been mentioned earlier in this report, and particularly at the beginning of Section 7.2.3 with respect to the full scenario, another set of simulations has been carried out to verify whether ICs and, more generally, state-space regions not corresponding to the ones encountered during the optimization would lead to further failures. The simulations are set up with an IC grid of 3 bins per state component, but with a coarser RT grid over a smaller surface area contained within $(-450, 1150)\,\text{m} \times (-800, 800)\,\text{m}$, for the downrange and crossrange directions, respectively. The results are omitted to avoid redundancy, as they are very similar to the plots of Figure 7.13, but it can be concluded that, even for thicker IC grids, only some of the tested RT points, which lie outside the target area marked in red in Figure 7.13, lead to failure rates different than zero. The robustness of the RT and RTPC networks with respect to worst case RT conditions is therefore successfully verified, despite the difficulties



(a) RT-HANNA LS visibility map.



(b) RTPC-HANNA LS visibility map.

Figure 7.12: LS visibility maps for RT-HANNA and RTPC-HANNA.

encountered with this regard during the full scenario optimization campaign.

7.2.4. ATTITUDE AND ATTITUDE RATE

In this section, the off-vertical attitude residuals at TD will be analyzed for the HANNA networks presented in this chapter, and compared to those of the Apollo guidance and NN B (see Table 6.1), already presented in Section 6.2.4. Subsequently, the work that has been done to limit the vehicle's rotational rates will be presented, and recommendations will be given for future work in this direction.

ATTITUDE RESIDUALS

In Section 6.2.4, the off-vertical attitude residuals at TD ($\tilde{\varepsilon}_{TD}$) for the Apollo guidance and NN B have been presented and compared. These are reported here in the top-left plot of Figure 7.14, together with analogous plots for the HANNA networks presented in this section, simulated with and without RTs. The residuals are shown as polar plots, where $\tilde{\varepsilon}_{TD}$ is the magnitude of the residual, and the azimuth angle ψ_{TD} expresses its direction with respect to the $\mathcal{G}$-frame x-axis.

In general, the distributions for the HANNA networks are more compact than that of NN B, but less so than that of the Apollo guidance. This means that switching to the Apollo policy during the last 30 m of the descent is beneficial to the purpose of containing attitude residuals, but that the state dispersion at the HANNA interface does prevent the Apollo guidance from achieving the same level of attitude accuracy attained if the Apollo policy is followed all the way through the descent. In fact, the corrections Apollo needs to make are likely larger when following the network's policy, thus leading to a more dynamic LS approach. Similarly, the direction along which the distribution is spread out depends on interface conditions as well, as cross-track offsets and velocities result in an angled approach with respect to the downrange direction.

One can also see that the presence of RTs does not significantly affect the direction of the attitude

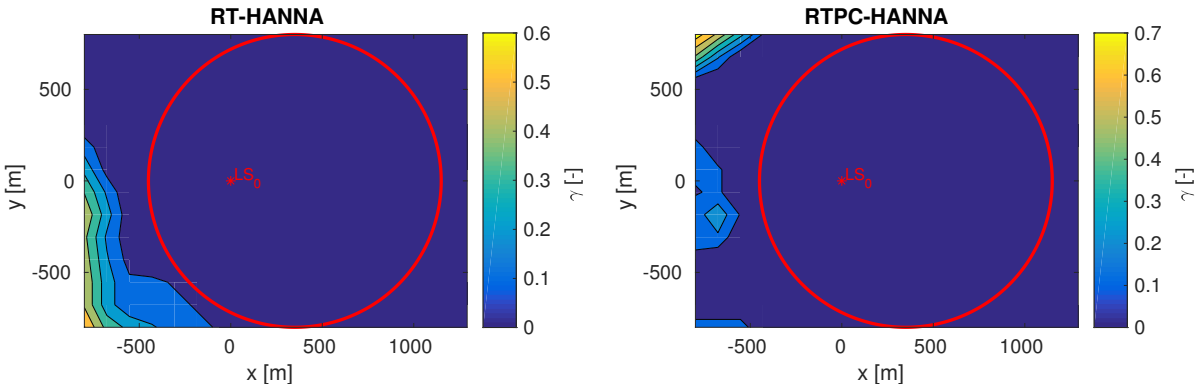


Figure 7.13: Failure maps for the RT-HANNA and RTPC-HANNA networks. Nominal landing site and target divert area are shown in red.

distributions, as the effect of the different networks is the main cause for this. However, it does seem to slightly increase the worst case values, which reach slightly above the required value of 8^o (GR-08) for the RT network. Except this one mild outlier, the distributions are well within bounds for both the RT and RTPC networks, with most of the samples within 6^o. PC-HANNA also shows a couple of outliers at values of almost 10^o, while all the other samples are within the required value. Overall, the performance of the networks with respect to attitude residuals can be considered satisfactory. However, if the presence of outliers were to reveal itself as a common phenomenon from the analysis of additional networks, given the already reasonable distributions achieved without any constraint, it would be possible to further decrease their variance (or worst case value) by the required amount by enforcing an additional penalty term in the reward.

Within the context of the attitude residuals analysis, the most surprising aspect, already mentioned in Section 6.2.4, remains that NN B, which follows the network's policy all the way to the surface and has not been constrained in any way with respect to attitude residuals, still shows a distribution which is within reasonable limits, although more spread out than those of the HANNA networks. This is related to the high trajectory flown by the network (see Figures 6.11 and 6.12a), which leads to a more vertical approach, and by the conservative policy followed during the last part of the descent. The latter is thought to be due to the stochastic nature of the optimization. In fact, the specific parameters leading to both an accurate and very dyanamic landing can likely not be found. As such, the optimization settles at a local minimum which results in a fuel-suboptimal, less dynamic, and more vertical landing.

ATTITUDE RATE

Different strategies have been tested to also limit the attitude rate of the vehicle. In fact, despite both the guidance algorithm and the simulator only focus on CoM dynamics, the limitation of rotational rates is nonetheless very important to obtain trajectories which could be realistic from a rotational dynamics point of view.

Some improvement in this direction was achieved with the implementation of the HANNA approach. In fact, this strategy was initially meant as a way to limit the attitude rate peaks produced by navigation updates during the final part of the descent. As one can see from Figure 7.15 (right-hand plot), the NN is more sensitive to this sort of perturbations than the Apollo guidance. The plot shows the attitude rate profile during the final seconds before TD, after the Apollo guidance is switched over to (red line), as compared to a pure-NN policy (blue line). The peaks, purely caused by measurement updates and thus occuring at a 1 Hz frequency, reach values up to $100^o/s$ ($t = 49\,s$) when a pure-NN is used. On the other hand, with the introduction of the HANNA guidance, maximum attitude rate values can be limited at below $\sim 40^o/s$ during this final phase. This is becuase, as already discussed in Section 6.2.4, the Apollo policy is more homogeneous over the state-space

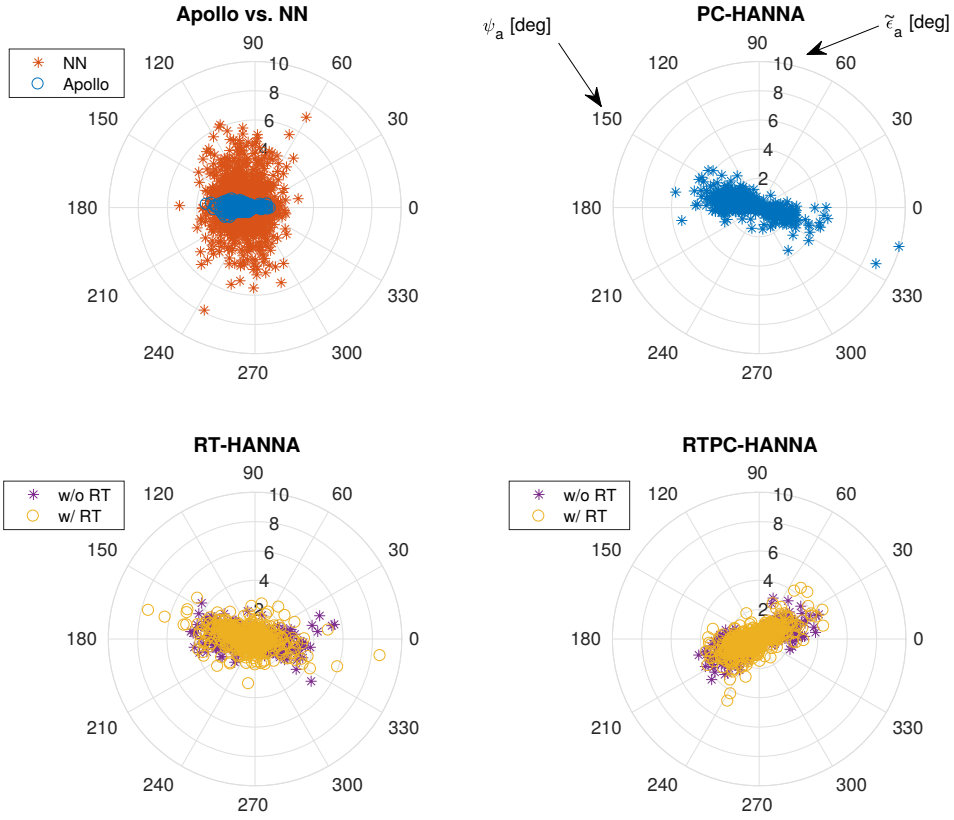


Figure 7.14: Comparison of attitude residuals at TD for the HANNA networks, NN B, and the Apollo guidance.

than that of the NN. As such, the sudden changes in estimated state due to measurement updates produce smaller variations in commanded acceleration, and, therefore, lower attitude rates.

However, in spite of these values being too high for a realistic scenario, they do not represent the main concern when it comes to limiting attitude rates. In fact, these peaks are of very short duration and can usually be saturated without resulting in control loss, that is, attitude rates do not remain at their saturation value for a long time, and actuated acceleration will therefore converge back onto the commanded one after a few seconds. On the other hand, the left-hand plot of Figure 7.15 shows the evolution of the attitude rates throughout the entire descent. The lower but longer excess values (sections above the red line threshold) that occurr at earlier stages of the descent are much harder to constrain by means of a simple limiter. In fact, saturating the rate of change of the commanded acceleration during these sections often leads to a divergence between commanded and actuated controls, eventually leading the state outside the domain within which the NN has been trained and, therefore, to failure. It is also to be noted that, given the challenges encountered for limiting attitude rates within this optimization framework, the target threshold value has been relaxed with respect to the original requirement GR-07 of $\omega_{thr} = 3^o/s$. As such, the red line in Figure 7.15 refers to a value of $\omega_{thr} = 0.1\,\text{rad/s} \approx 6^o/s$.

Efforts were made to limit these more hazardous peaks both by means of an additional penalty term in the reward function, and by running the optimizations with the rate limiter in the loop, as it is done for the thrust magnitude. In the latter case, the lowest value for the rate threshold for which convergence could be achieved is $\omega_{thr} = 0.5\,\text{rad/s} \approx 30^o/s$. For such network, Figure 7.16a shows the attitude rate profile for the nominal trajectory with different ω_{thr} values. The saturation-free profile is included as reference. One can see that the switch from the NN policy to the Apollo guidance expectedly introduces a peak in attitude rate ($t \sim 43\,s$). This gap, however, can be easily bridged when saturation is introduced (red line), as the curve remains for a few seconds at the saturation

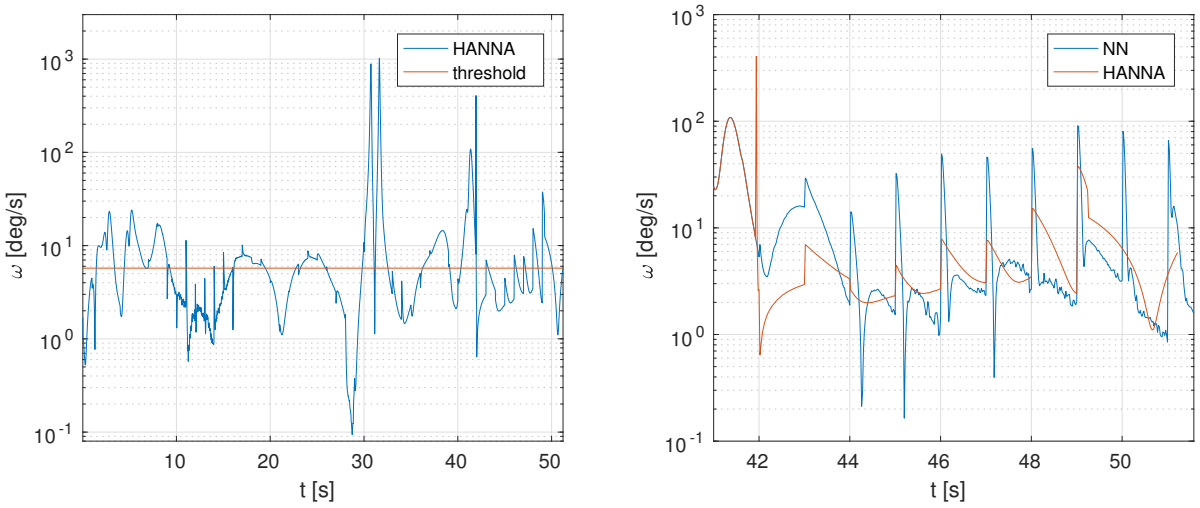
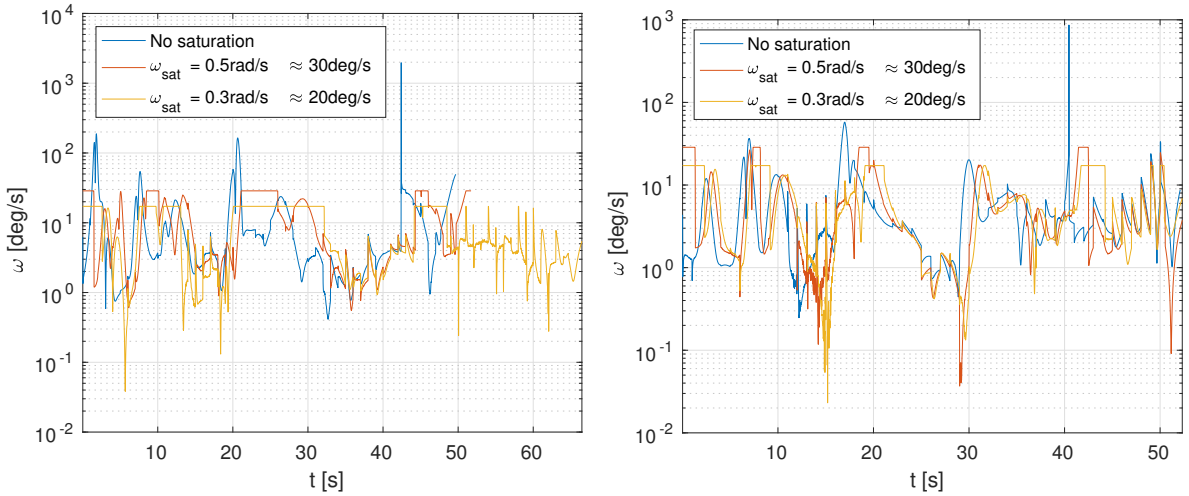


Figure 7.15: Attitude rates of a HANNA network during the entire descent (left), and comparison with a pure-NN policy during the final seconds only (right).

value ($t \sim 45-47\,\mathrm{s}$), before decreasing again. The delay with respect to the saturation-free trajectory is due to the saturation segments occuring earlier in the descent, which result in time lags. Also, the network still achieves a safe landing when ω_{thr} is reduced below the value it has been optimized for (yellow line), although at the expense of increased fuel consumption. This is due to the much longer duration of the process, caused by increased portions where rotational rate is maxed out (e.g., from $t \sim 20\,\mathrm{s}$ to $t \sim 32\,\mathrm{s}$). Finally, for ω_{thr} values lower than $0.3\,\mathrm{rad/s} \approx 18^{o}/\mathrm{s}$, soft landing cannot be achieved.

On the other hand, Figure 7.16b shows corresponding results in the case the attitude rate is limited in the optimization process by means of a penalty in the reward function. Again, the NN with saturated attitude rate is robust against the rate peak induced by the NN-Apollo switch. Also in this case, this is delayed with respect to the saturation-free trajectory, but Apollo is then able to recover the time delay induced by the network during the first part of the descent, and landing is achieved at approximately the same time. Similarly to the previous case, the threshold attitude rate value can be reduced down to $\omega_{thr} = 0.5\,\mathrm{rad/s} \approx 30^{o}/\mathrm{s}$ with no significant consequences in terms of accuracy or propellant consumption. When the threshold is further decreased to $\omega_{thr} = 0.2\,\mathrm{rad/s} \approx 12^{o}/\mathrm{s}$, performance degrades more gradually for velocity residuals ($\|v_{H,f}\| \sim 0.7\,\mathrm{m/s}$), while complete failures occur for further lower values. Ultimately, given these results, it must be said that the aimed-for threshold value of $0.1\,\mathrm{rad/s}$ ($\approx 6^{o}/\mathrm{s}$) could not be achieved with any of the tested methods, and further work is therefore required in order to make this guidance scheme suitable for handling rotational dynamics limitations.

Finally, these results are compared to the Apollo guidance, which performs expectedly better in this sense, given the lower gradient of its output with respect to that of the NN. Figure 7.17 shows the Apollo guidance attitude rates for the entire descent. One can see that maximum values remain consistently below 6^{o}. Also, by further limiting them to the original target value of 3^{o}, set by requirement GR-07 in Section 2.2.1, a safe TD can still be achieved with approximately the same time of flight. The figure confirms that, also in this case, attitude rate peaks during the final seconds can be easily recovered, while, for the remainder of the process, the profile remains intrisically below the threshold value, and hardly any saturation is actually necessary (except for a short amount of time at $t \sim 7\,\mathrm{s}$).



(a) Results for a HANNA network optimized by direct attitude rate limiting and simulated with different saturation threshols values.

(b) Results for a HANNA network optimized by attitude rate penalty in the reward function and simulated with different saturation threshols values.

Figure 7.16: Attitude rate limiting best achievable results.

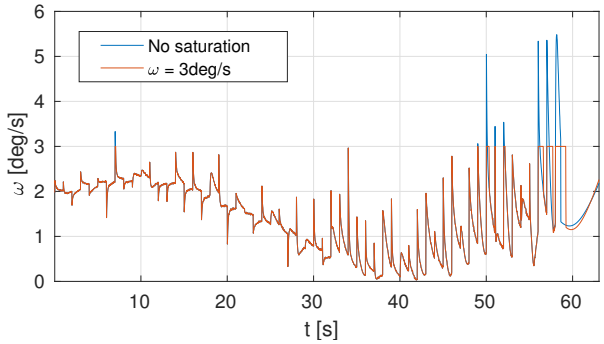


Figure 7.17: Attitude rate behavior of the Apollo guidance during the entire descent, with and without attitude rate saturation.

CONSIDERATIONS AND RECOMMENDATIONS

The attitude rate of the optimal solution, used for supervised training of the networks, was within the target value of 0.1 rad/s. In spite of this, during the optimization process, the randomization of the neural network weights always produced outputs leading to large increases in this quantity. Also, just rotational kinematics, but no dynamics, have been considered here. Given that, even in this simplified scenario, the objective could not be achieved, one must conclude that it would be very hard to achieve a policy complying with all rotational dynamic and kinematic constraints, by using the NN guidance scheme as is.

At the very least, the guidance system needs to be designed with a 6-DoF process in mind. In such scenario, the NN output would consist in a 4-dimensional vector consisting of acceleration magnitude and the three torque (or rotational acceleration) components. It is possible that, if the rotational acceleration is a direct output of the network, it would be easier to constrain it, as it was easier to constrain the thrust magnitude in this work. This is an hypothesis, which needs to be tested. One sure advantage of expressing the network output in this way would be to relax the requirement on the control system, whose task would then be that of reference tracking alone, as the attitude profiling would already be done by the guidance system.

Finally, the implementation of a value-function-based RL method could also be beneficial. In

fact, this would allow for a more selective adjustment of the neural network weights by using eligibility traces. This way, only the network parameters which affect its output at the recently visited states would be tuned, without altering the entire solution. This is believed to be beneficial for other kinds of constraints as well, such as attitude constraints, as it allows for a more tailored shaping of the solution.

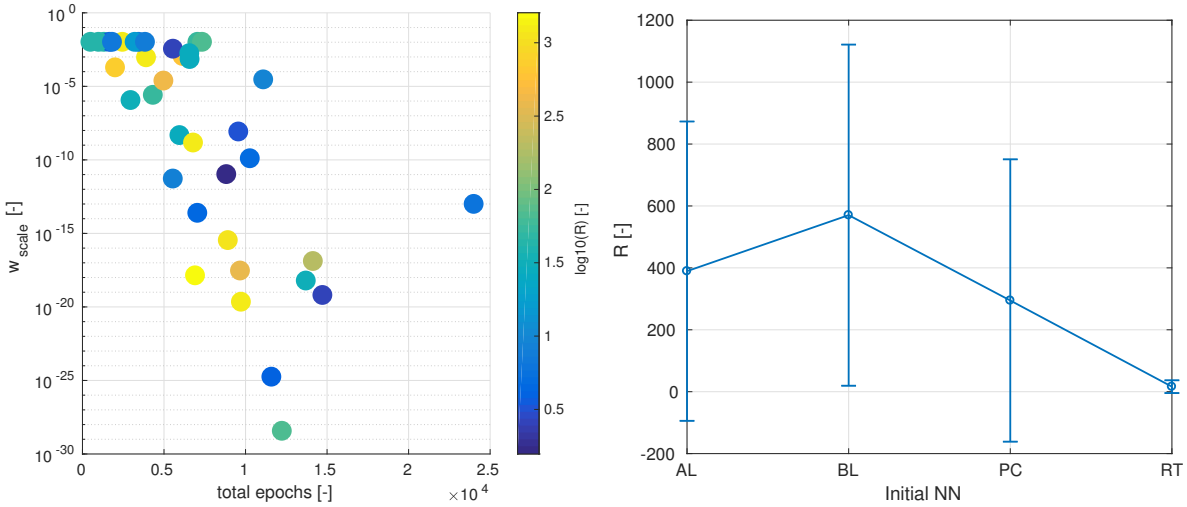
7.2.5. CONVERGENCE ANALYSIS

Similarly to Section 6.1.3, the convergence of the optimization algorithm will be analyzed here for the full scenario, for which a larger number of optimization instances has been run, as compared to the BL scenario. The results are presented after pruning away the optimizations for which convergence of position and velocity could not be achieved (i.e., 4 out of 42, as mentioned already on page 129). In fact, given their low rate of occurance, the nominal performance of the optimization algorithm is better represented by the pruned results.

As already seen in Table 7.6, propellant consumption and pointing accuracy performance are well correlated for the full scenario. As such, looking directly at the total reward is well representative of both these metrics. Figure 7.18a shows, for the full scenario, the same plot presented in Figure 6.5b for the BL scenario. While looking at these data one should keep in mind that not only is the scenario different, but also that a pure-NN policy was being used for the first part of the optimization campaign, while the HANNA approach has been used for the optimizations presented here. One can see that, in both cases, there is a correlation between the final value of w_{scale} and the total number of epochs, but the correlation is more clear in the present figure. Also, although in both cases some of the best instances (darker dots, corresponding to lower negative reward) occur for large values of total epochs, no clear correlation is present in this sense.

On the other hand, some important differences can be seen between the two figures. First, the samples in the full scenario are more spread out in the vertical direction. Second, the steepness of the trend is much larger in the full scenario. Finally, since the first does not offset the second, the total number of epochs is generally lower for this scenario. These differences are due to the introduction of the HANNA method. In fact, as already discussed in Section 7.1.3, the Apollo guidance serves as a funnel for the NN trajectories. This makes it such that fewer attempts (that is, parameter perturbations) are required before an improvement in reward is achieved. On the other hand, the reduction of the scale factor is nonetheless required. Consequently, this will lead to equally low (or slightly lower) values of w_{scale}, but within a lower number of epochs. While for the BL scenario most of the samples are contained within 15000 and 20000 epochs, in the full scenario this range is reduced to 10000-15000 epochs, with a subsequent reduction in optimization time.

The vertical distribution of the samples in the two cases shows that larger numbers of samples are present at both lowest and highest values of w_{scale} in the full scenario. Let us first look at the former case. As one can see in Figure 7.18a, many instances stop converging at $w_{scale} = 0.01$ (that is, no decrease from the initial value). This is not the case in the BL scenario, for which, however, a large sample density can nonetheless be seen for moderately low values of w_{scale} (i.e., $10^{-7} < w_{scale} < 10^{-3}$). One can see that, in both cases, acceptable, yet not optimal, reward values are associated with these samples. With the HANNA policy, this can be achieved within just a few thousands of iterations, given the funneling effect of the Apollo guidance, which more efficiently removes position and velocity residuals. However, if a pure-NN policy is used, a more extensive search is required before an equally acceptable reward value can be obtained. Hence, the lower w_{scale} values of this sample cluster. Conversely, lower minimum values of w_{scale} can also be achieved with HANNA because, once convergence is achieved for position and velocity, these are unlikely to degrade substantially if the scale factor is not incresed (which never happens), therefore reducing the risk of the optimization getting stuck in a local minimum. This, in turn, allows for a more sustained and finer search of the parameter-space, thus keeping the optimization running and, therefore, re-



(a) Correlation between final scale factor and total number of optimization epochs. The color shows the final total reward.

(b) Effect of the initial NN parameter vector on the final total reward.

Figure 7.18: Correlation between final scale factor and total number of optimization epochs and effect of initial NN parameters.

sulting in lower values of w_{scale}.

As far as the correlation of these metrics (w_{scale} and total epochs) with the tuning parameters is concerned, no significant relations were found. However, as more than half of the instances shown were run with $n_{const} = 2000$, it can be said that this did result in longer optimizations than $n_{const} = 1000$, as expected. On the other hand, its effect on the final reward is marginal, as only a small improvement can be noticed from the available samples. However, a more important role in this direction is played by the set of NN parameters used to initialize the algorithm. Figure 7.18b shows the final optimization reward for different initial NN weights. The x-axis ticks refer to apprenticeship learning (AL), a BL optimization with pure-NN policy, and PC-HANNA and RT-HANNA optimizations, respectively. One can see that, contrary to expectations, the AL initial parameters do not yield the worst results among the four cases. However, it must be said that the fewest instances were run with this input, which makes the statistical quantities (the error bars refer to 1σ variations) shown in the plot less accurate. Besides, different reward functions were used to obtain the different samples for each of these cases, which further increases the ambiguity of the metrics shown. Ultimately, the AL case is expected to yield the poorest results if a larger number of instances were available. On the other hand, the trend for the remaining three cases is believed to be more accurate, as the final reward decreases as networks are used for which more elements/constraints had been included in their original optimization environments. In particular, the improvement achievable by using a RT-trained network is considerable, espcially in terms of consistency. This is in agreement with what was found in Sections 7.2.1 and 7.2.2, that is, that the optimization for the PC scenario was more straightforward and less intensive than for the RT scenario. As such, by taking advantage of previous efforts made to achieve a good optimization for the latter scenario, it was possible to considerably improve the optimization performance in the full scenario as well.

The considerations made in this section hint at possible ways forward and further developments to improve optimization efficiency. The first, directly stemming from the last paragraph, is that of carrying out the optimization in a piecewise fashion, that is, by including one element at a time in the optimization environment, and starting from the arrival point when including the next. This method is effective, yet inefficient, as it requires a considerable design effort and it reduces the automation of the optimization process. On the other hand, one could possibly achieve a similar

effect by implementing an adaptive scale factor adjustment. This will help in getting the algorithm unstuck when local minima are reached which cannot be left with the current w_{scale} value. A proper implementation for this technique needs to be researched. Finally, it was shown here in detail how the HANNA approach is beneficial to convergence speed and performance. However, following this approach in a unified fashion reduces the insight that can be gained about interface conditions. As such, a two-phase PDG strategy could be followed, where the NN is optimized separately from the Apollo guidance (or other algorithm used for the final phase). This does increase the complexity of the design, but it also allows for a proper investigation of the NN-Apollo interface, which is required for a smooth and safe transition from one guidance system to the next. The algorithm responsible for the second phase needs to be investigated first, and its envelope of ICs assessed, which can then serve as final conditions for the optimization of the NN in the RL framework. The process will also need to be reiterated with both algorithms in the loop, so as to find the most optimal interface both in terms of propellant optimality and landing accuracy, as well as other metrics of interest (e.g., pointing accuracy).

7.3. SENSITIVITY ANALYSES

On top of the already presented stress cases for the RT points, additional sensitivity analyses have been performed. In Section 7.3.1, robustness against ICs outside the training domain of the state-space is investigated, similarly to what was done in Section 6.2.3 for the BL scenario. In Section 7.3.2, robustness against environmental conditions, such as worst case winds and atmospheric density, is looked into, while Section 7.3.3 discusses sensitivity with respect to system parameters, such as measurement uncertainties and control perturbations.

7.3.1. ROBUSTNESS AGAINST IC VARIATIONS

Figure 7.19 shows failure rates for varying initial velocities for different guidance schemes. The color refers to failure rate γ, whose value is averaged over the initial position values, ranging from 0 (never fails) to 1 (always fails). The Apollo guidance and the pure-NN approach are shown in the first column. These have already been extensively discussed in Figure 6.10, and are included here for reference purposes. The second column shows the BL-HANNA (first row) and the PC-HANNA network (second row). Finally, the last column plots refer to the RT-HANNA (first row) and the RTPC-HANNA network (second row).

By comparing these with the first column plots, it can be seen that the increase in robustness against initial velocities outside the training domain is remarkable when the HANNA guidance is used, even when the networks are not specifically trained against extended state-space regions, as is the case for the plots of the second column. In fact, one can see that both BL-HANNA and PC-HANNA show areas of $\gamma = 0$ which are much more extended than they are for the BL-NN case (bottom-left plot). However, for the bottom plane of the plots ($v_z = -120\,\mathrm{m/s}$) both networks always fail to achieve soft landing. For the other values of vertical velocities, some anomalies (that is, ICs for which the γ value is very different from that of their neighbors, and changes in a non-regular fashion) are still present, although to a lesser extent than for the pure-NN. These anomalies can be identified as lighter-colored points within darker areas, and are due to the common irregularities of the networks' policies over the state-space, as a result of the stochastic perturbation of their parameters during the optimization. The substantial increase in robustness achieved by the HANNA networks when optimized for the baseline state-space domain seems to be due to the funneling effect of the Apollo guidance, as trajectories that would have resulted in final residuals which are inaccurate, yet not excessively so, are successfully brought to an accurate landing.

The increase in robustness is even more significant when the networks are specifically optimized against RTs, as shown by the two rightmost plots. In fact, for the RT and RTPC networks, as long as $v_z > -120\,\mathrm{m/s}$, failures do never occur, regardless of initial position. Some level of accuracy en-

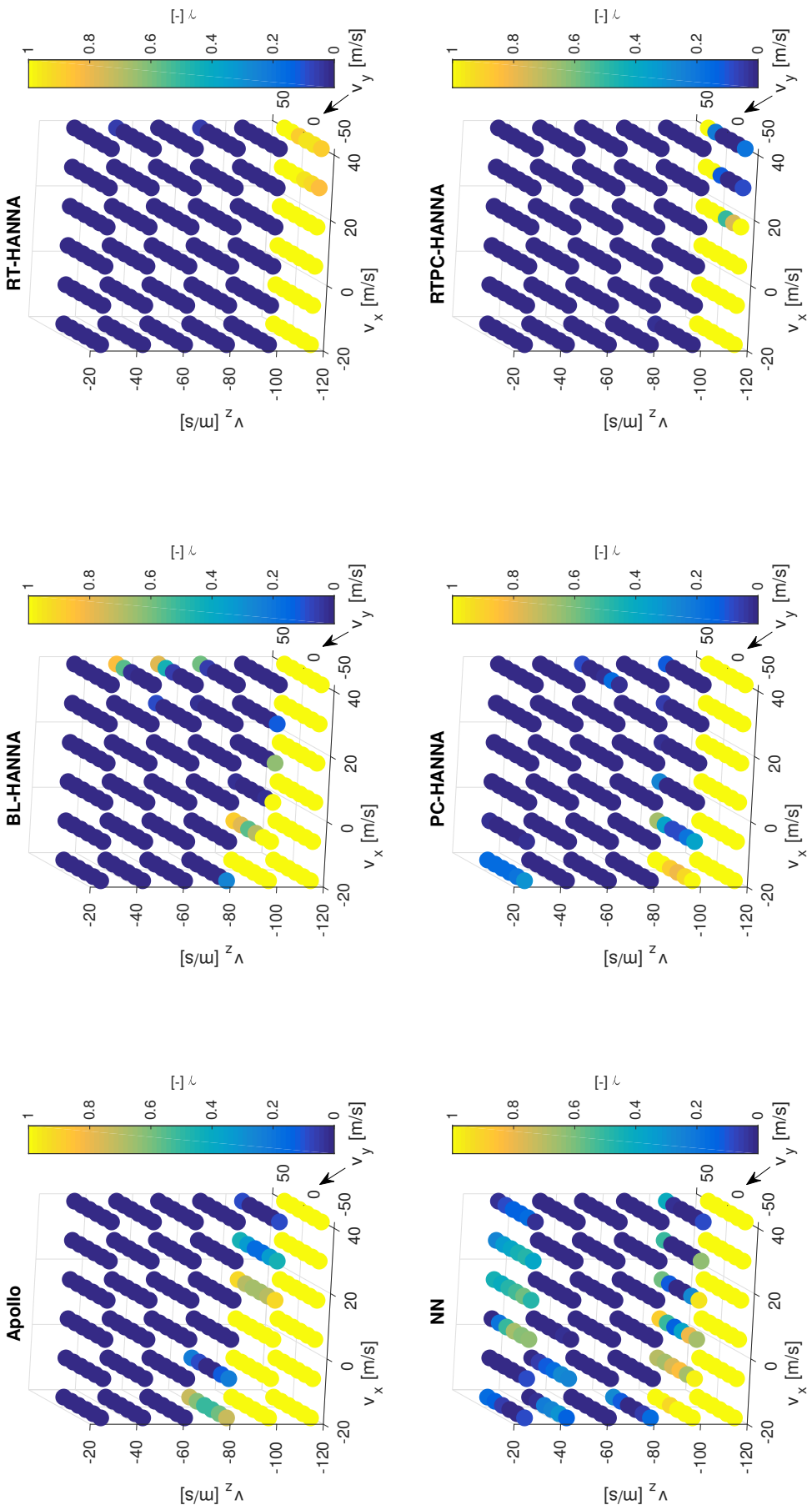


Figure 7.19: Robustness against initial velocities within and outside the training domain.

hancement is also possible at $v_z = -120\,\mathrm{m/s}$, for which all of the other tested networks show failure rates consistently equal to $\gamma = 1$. In particular, as expected, it is for large downrange velocities that this failure reduction is possible, as these combinations produce states which are more forgiving with respect to command delays or inaccuracies. Also, if combined with large negative downrange *positions*, a positive downrange velocity helps the lander to re-enter the known position-space domain, while a negative one works in the opposite direction, thus increasing failure likelihood. Such a large increase in IC robustness for RT-trained networks is not so straightforwardly explained, as RTs are seen as relative shifts in horizontal position, and the behavior with respect to velocity should not be directly affected. As such, the enhanced robustness must be due to the generalization capability of the NN. It seems, in fact, that training the network in an extended position-space, is beneficial to approximating an acceptable solution in extended areas of the velocity-space as well, although actions in these areas might be less optimal than in directly-visited ones.

Similarly to what was found in Section 6.2.3, initial position has a lesser influence on failure rates. In fact, the variations in failure rate for varying position are even smaller than those of Figure 6.9 for Apollo and NN, as compared to their mean values, which are, in turn, mainly determined by the failure rates over the velocity-space, shown in Figure 7.19. As such, the plots have not been included here to avoid redundancy in the discussion.

The test presented in this section stresses once more the excellent robustness and generalization capabilities of NNs, especially when used in combination with the Apollo guidance, as zero failure rates could be achieved for an IC envelope of individual components almost twice as large as the training ones.

7.3.2. ENVIRONMENTAL PARAMETERS

In this section, the sensitivity of the network with respect to varying atmospheric variables is investigated. For computational reasons, from now on, only the network optimized for the full scenario will be tested. Also, as the simulation is stopped at the first negative altitude, a linear interpolation is performed between the last two time steps, to remove errors stemming from the excess simulation time.

ATMOSPHERIC DENSITY

Atmospheric density has been kept deterministic and unvaried throughout the entire set of tests that have been performed so far. The effect of its variations is now investigated. Five trajectories have been simulated. Their accuracy and propellant consumption are reported in Table 7.8. One can see that there are only small differences in position and velocity accuracy, and without a clear trend. On the other hand, there is a clear trend in propellant consumption, which expectedly increases as density decreases, with a maximum difference of $\sim 1\,\mathrm{kg}$ of propellant between best and worst case.

Table 7.8: Accuracy and propellant consumption for varying atmospheric density.

	$x_f[10^{-3}\,\mathrm{m}]$	$y_f[10^{-3}\,\mathrm{m}]$	$v_{x,f}[10^{-3}\,\mathrm{m}]$	$v_{y,f}[10^{-3}\,\mathrm{m/s}]$	$v_{z,f}[\mathrm{m/s}]$	$\Delta m_{prop}[\mathrm{kg}]$
$0.8\rho_0$	-8.4097	-3.2023	-9.3794	-3.6315	-0.41778	271.90
$0.9\rho_0$	-8.4261	-3.2120	-9.4331	-3.6560	-0.41772	271.64
ρ_0	-8.3941	-3.2010	-9.3818	-3.6374	-0.41774	271.43
$1.1\rho_0$	-8.3619	-3.1899	-9.3323	-3.6195	-0.41775	271.21
$1.2\rho_0$	-8.3768	-3.1989	-9.3938	-3.6472	-0.41770	270.95

Table 7.9: Accuracy and propellant consumption for varying aerodynamic coefficient values.

	$x_f[10^{-3}\,\mathrm{m}]$	$y_f[10^{-3}\,\mathrm{m}]$	$v_{x,f}[10^{-3}\,\mathrm{m}]$	$v_{y,f}[10^{-3}\,\mathrm{m/s}]$	$v_{z,f}[\mathrm{m/s}]$	$\Delta m_{prop}[\mathrm{kg}]$
$0.8C_{a,0}$	-8.4097	-3.2023	-9.3794	-3.6315	-0.41778	271.90
$0.9C_{a,0}$	-8.4261	-3.2120	-9.4331	-3.6560	-0.41772	271.64
$C_{a,0}$	-8.3941	-3.2010	-9.3818	-3.6374	-0.41774	271.43
$1.1C_{a,0}$	-8.3619	-3.1899	-9.3323	-3.6195	-0.41775	271.21
$1.2C_{a,0}$	-8.3768	-3.1989	-9.3938	-3.6472	-0.41770	270.95

AXIAL AERODYNAMIC COEFFICIENT

As mentioned in Section 4.9, the axial aerodyamic coefficient of the lander was obtained by adding a 20 % penalty to the Beagle capsule coefficient $C_a = 1.425$. Now, an additional $\pm 20\,\%$ will be applied to investigate the effect of this uncertainty. The results, shown in Table 7.9, look exactly the same as those obtained for varying atmospheric density, when rounded to the fifth significant digit. This makes sense, as lower atmospheric density and smaller aerodynamic coefficient have similar effects on the total aerodyamic force.

GRAVITATIONAL ACCELERATION

The gravitational acceleration, as obtained from GMM-2B, is more accurate than atmospheric density models. As such, it is perturbed between -2 % and 2 %. The results are shown in Table 7.10. One can see that the control policy is considerably more sensitive to gravity variations than it is to atmospheric density or winds. Position varies up to $\sim 1\,\mathrm{cm}$, velocity variations are of $\sim 4\,\mathrm{cm/s}$, and propellant consumption varies up to almost 3 kg. Besides, further perturbing the nominal gravity profile, especially for smaller scaling factors, quickly degrades the performance, leading to very clear failures for gravity profiles of 95 % the nominal one. Performance decreases more gradually for larger-than-unit scaling factors, but it still results in out-of-bound final conditions for variations of 5 %. These cases are thus not included in Table 7.10, as the actual values would not be of great significance when compared to those reported therein. In fact, such variations are unrealistically large, as Mars' gravitational field is known to a better degree of accuracy. However, it is interesting to notice how the algorithm is very sensitive to these variations. The main reason for this is that only a constant gravity acceleration was included in the training environment. It could be worth considering a closer study on gravitational acceleration knowledge accuracy on Mars, and modeling the optimization environment accordingly.

Table 7.10: Accuracy and propellant consumption for varying gravitational acceleration.

	$x_f[10^{-3}\,\mathrm{m}]$	$y_f[10^{-3}\,\mathrm{m}]$	$v_{x,f}[10^{-3}\,\mathrm{m}]$	$v_{y,f}[10^{-3}\,\mathrm{m/s}]$	$v_{z,f}[\mathrm{m/s}]$	$\Delta m_{prop}[\mathrm{kg}]$
$0.98g_{GMM}$	-4.3425	-1.6490	12.537	5.5394	-0.24158	273.95
$0.99g_{GMM}$	-8.3152	-3.2378	-2.9137	-1.0748	-0.34145	272.14
g_{GMM}	-8.3941	-3.2010	-9.3818	-3.6374	-0.41774	271.43
$1.01g_{GMM}$	-8.6532	-3.2489	8.2852	3.4452	-0.48549	271.21
$1.02g_{GMM}$	-12.335	-4.6464	52.987	20.962	-0.54958	271.36

WIND SCENARIO AND DIRECTION

As mentioned in Section 4.8, a mild and a strong wind case scenario have been implemented in the simulation model. The sensitivity with respect to these two scenarios, as well as to the mean wind direction, is investigated here. The results are presented in the form of polar plots in Figure 7.20. The azimuth angle $\psi_{W,\mathcal{G}}$ refers to wind direction with respect to the $\mathcal{G}$-frame x-axis, and increases clockwise. So, $\psi_{W,\mathcal{G}} = 0^{\mathrm{o}}$ refers to a head-wind, while $\psi_{W,\mathcal{G}} = 90^{\mathrm{o}}$ corresponds to a rightward wind.

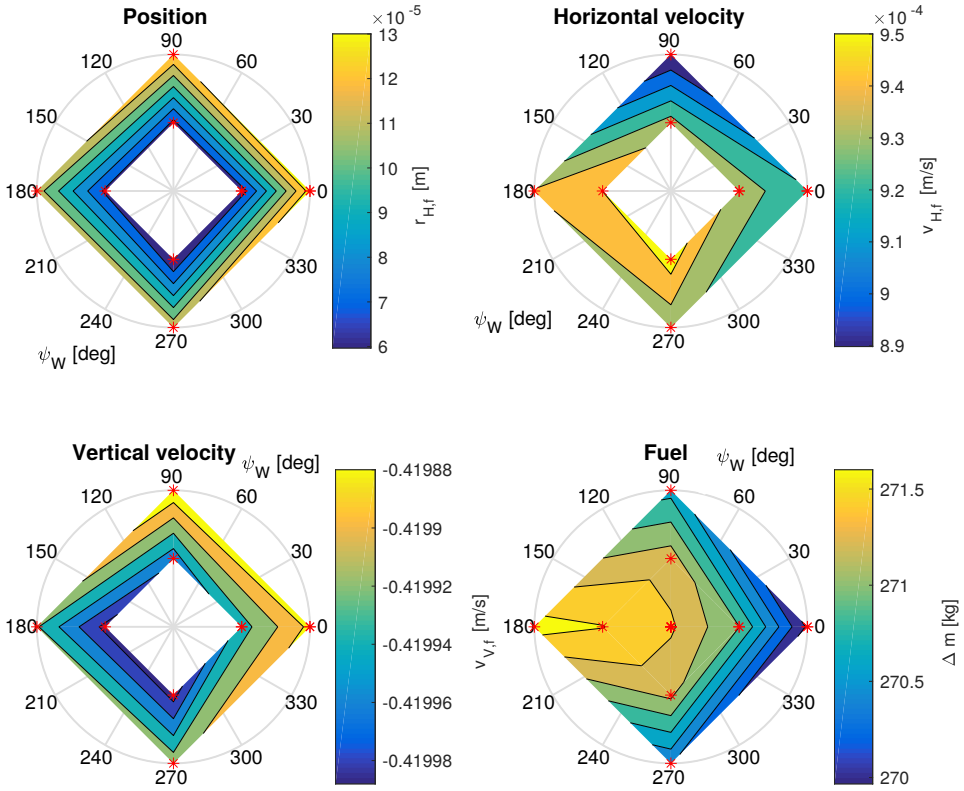


Figure 7.20: Accuracy and propellant consumption for different wind scenarios and directions.

Since wind profiles are generated as an autocorrelated random sequence, a MC run is required for each scenario. However, given this is the only random model for these case studies, a MC simulation with 100 runs is deemed to be sufficient. The plots in Figure 7.20 show the mean values of the performance metrics. Data points are marked with an asterisk, where the inner ones correspond to the weak wind scenario, while the outer ones refer to the strong one. One can see that position and velocity plots are missing the nominal value (that is, the windless scenario). In fact, variations between off-nominal cases are much smaller than their difference with respect to the nominal case, so that plotting the nominal value would make them unreadable. This is because the off-nominal values are mean values and tend to average out, while the nominal one comes from a single deterministic trajectory. This was not the case, however, for fuel usage values, so the nominal one could be included.

The position error increases almost equally in all directions from weak to strong wind scenarios, whereas small asymmetries are there for vertical velocity. Horizontal position error is more unpredictable, but seems to be more robust against head and rightward winds. Propellant consumption expectedly decreases for incresing head winds, similar to what has been found for density and aerodynamic coefficient. For lateral winds, propellant consumption is unexpectedly lower for strong than for weak winds. Both fuel consumption, position and velocity change slightly more than they do for varying density and aerodynamic coefficients, but less than for varying gravity.

7.3.3. SYSTEM PARAMETERS

In this section, the policy sensitivity with respect to variations in system parameters is investigated. These include thrust noise, thrust misalignment, and navigation system parameters.

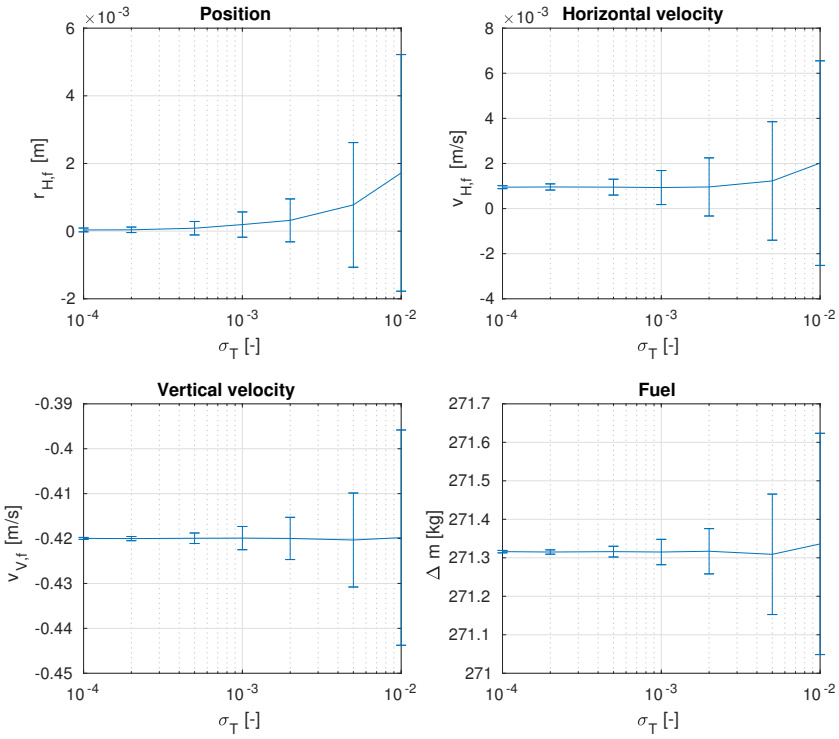


Figure 7.21: Accuracy and propellant consumption for varying thrust noise magnitudes.

THRUST NOISE

Thrust noise is a process noise. As such, a MC is required to investigate the effect of variations of its parameters. Given this is the only random sequence of the model, as other processes are kept deterministic, only 100 iterations have been run during the MC simulations. Given the lack of a clear reference for realistic values of this parameter, it has been varied logarithmically, across two orders of magnitude. As such, the plots of Figure 7.21 feature a logarithmic x-axis scale. However, information retrieved by looking at linear x-axis plots will be given.

Six different values for the standard deviation σ_P of the thrust noise ν_P have been tested. The results are shown in Figure 7.21. The plot shows mean position, velocity and propellant consumption, with corresponding 3σ bounds. With the data at hand, it seems that the confidence bounds increase in all cases approximately linearly with σ_P. It also looks like mean vertical velocity values remain constant, while position and horizontal velocity increase linearly. As for propellant consumption, there is some uncertainty in its trend, likely due to the low number of MC shots, but it the last data point hints at an increase in mass consumption for increasing thrust noise, as would be expectable. However, the nature of this relationship cannot be characterized.

THRUST MISALIGNMENT

Another parameter affecting the thrust vectors is its misalignment with respect to the $\mathcal{B}$-frame z-axis. This is not a process noise, but rather a constant randomized over different trajectory simulations. As such, a MC is not necessary to investigate its effect. Both magnitude and direction of this misalignment have been tested, with maximum misalignments of 0.2°. The results are shown in the polar plots of Figure 7.22. The azimuth angle ψ_P refers to the angle with respect to the negative x-semiaxis, increasing clockwise when looking at the lander body from above. This is also the nominal downrange direction, and corresponds to the pointing direction of the camera FOV, as can be seen in Figure 4.8b.

By looking at the plots, position and horizontal velocity show minima for non-nominal alignments, probably a consequence of the training in the noisy enviornment and the stochastic policy

search. There is, on the other hand, a hint of radial symmetry around the minimum, as one would expect, although the value range is not large enough for this to be clearly seen. It is not clear whether this is the case for vertical velocity as well, or whether the variation here is monotonic in the in-plane direction. Unexpectedly, fuel consumption is most affected by out-of-plane misalignments, with leftward misalignments ($\psi_P = 90^o$) leading to slightly lower fuel usage ($\sim 0.1\,\text{kg}$). This is likely due to asymmetries in the commanded acceleration field. Also, the source of the inhomogeneities which can be seen in the propellant consumption plot is unclear, as they are altogether not visibile in the other plots. They are, however, very small ($\sim 0.02\,\text{kg}$).

The effect of thrust misalignment parameters on position and velocity accuracy is the highest among the parameters tested so far, with the exception of the gravity model. Propellant consumption is affected to a lesser degree, showing smaller changes than those due to varying density, winds and aerodynamic coefficients. However, when only considering propellant consumption variations only due to later winds, one finds that leftward winds ($\psi_{W,\mathcal{G}} = 270^o$) lead to slightly lower propellant consumption ($\sim 0.3\,\text{kg}$) than rightward winds, which matches the effect of lateral thrust misalignments.

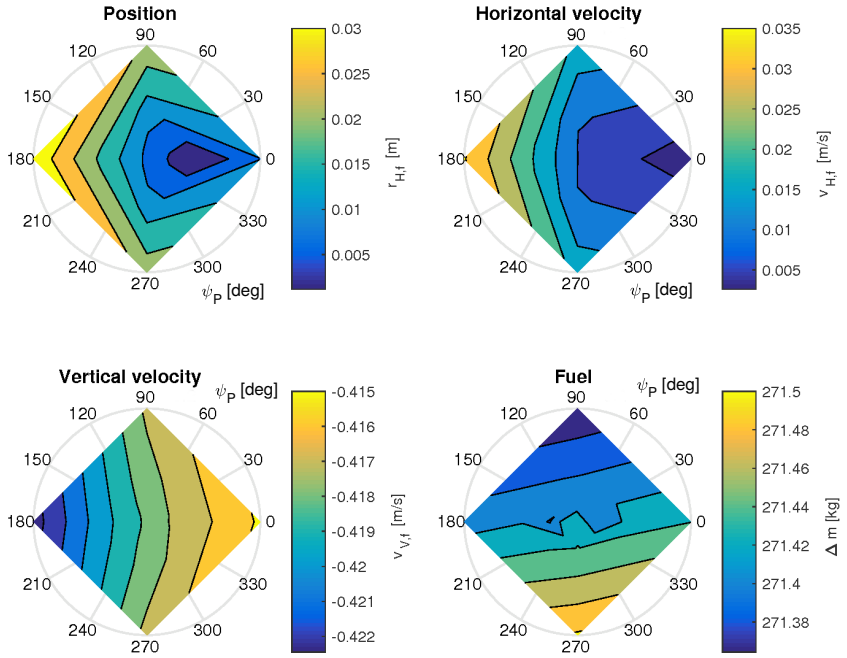


Figure 7.22: Accuracy and propellant consumption for varying magnitudes and azimuth of thrust misalignment.

NAVIGATION SYSTEM

The navigation system incorporates most of the random constants and sequences in the simulator. It features: a randomized constant accelerometer bias, the accelerometer drift's underlying noise, randomized constant map-tie errors, and the non-inertial measurements noise. Besides, these different measurements are blended into the filter, which makes it hard to decouple them. As such, only the ones which are thought to have the biggest impact on system performance are investigated. Accelerometer bias and noise are not very large, and the only real potential problem could be its drift, which could lead the navigated state to diverge. However, as long as accelerometer measurements are blended into the filter, this will not represent a problem, as even inaccurate, yet stable, non-inertial measurement will be able to prevent the estimation from diverging. As such, these measurements' noise is believed to be the key parameter as far as the navigation system performance's effect on the guidance system is concerned. Map-tie errors add directly to non-inertial

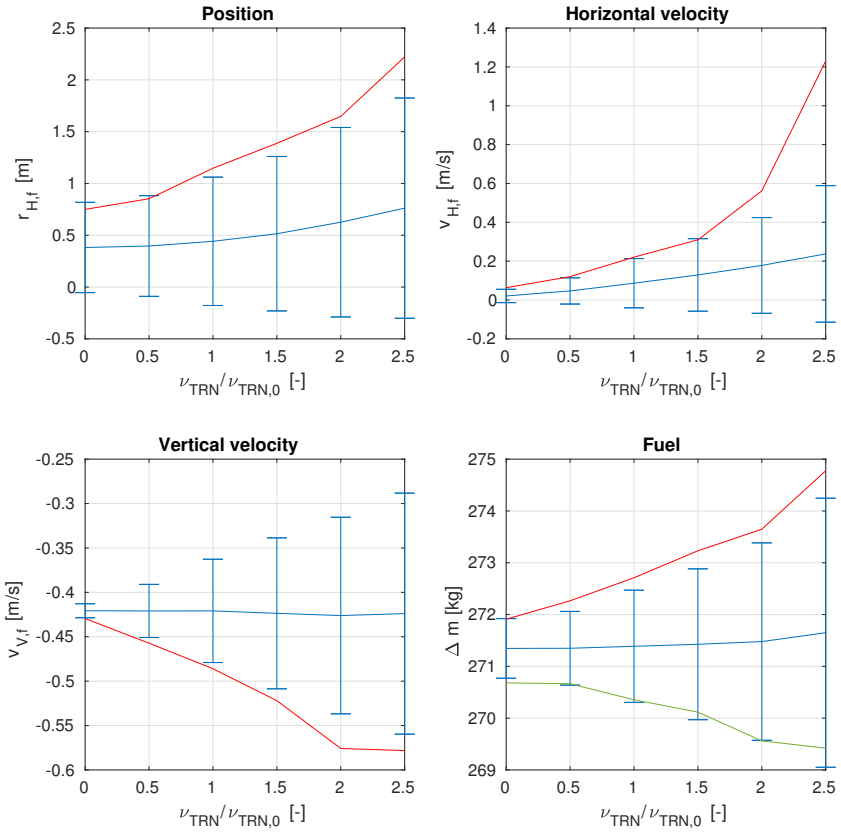


Figure 7.23: Accuracy and propellant consumption for varying non-inertial measurement noise.

horizontal position measurements, and their effect can thus be inferred from that of increasing position measurements noise.

Different scenarios have been tested, ranging from perfect non-inertial measurements to noises 2.5 times the nominal ones presented in Section 4.11. The results are shown in Figure 7.23. All the metrics seem to be increasing more than linearly, as well as their standard deviations. Vertical velocity seems to be the only exception, with a constant mean value and linear standard deviation. Also, propellant consumption increases noticeably at the last datapoint. This is unexpected, as larger residual velocities usually correlate with lower propellant consumption. However, it seems like the fuel surplus caused by state knowledge noise during the descent more than offsets the lower energy removal.

For noise magnitudes twice the nominal ones, worst case horizontal velocity values are already higher than 0.5 m/s, which could be outside the tolerance bounds of some landing systems. This comes at no surprise, as horizontal velocity measurements are modeled as a uniform distribution between -0.4 and 0.4 m/s for this scenario, while the measurement model in the training environment has been fit tightly to the Kalman filter output. This leads to a very small margin. Larger margins would improve robustness by limiting the true state dispersion to values not larger than the navigated state dispersion.

LINEAR KALMAN FILTER: CONCLUSIONS AND CONSIDERATIONS

The tests performed here have been carried out with the linear Kalman filter which has been used throughout the entirety of this work. The linearity of this filter makes it fragile to fast dynamic processes. Besides, its performance degrades for non-Gaussian biased input noises. Both of these are the case here. As such, the sensitivity analysis which has just been carried out bounds the performance of the guidance system against a more robust navigation filter, such as an extended Kalman

filter. On the other hand, the filter tuning parameters, as well as those of the measurement model, have been obtained by a reverse-engineering approach, so as to produce a more realistic set of input navigation errors, given the lack of realistic sensor and TRN system model. As such, the implementation of an estimation filter loses its purpose of assessing the performance of the guidance system under realistic measurement conditions. In particular, the original intention was that of realistically reproducing the larger navigation errors during the filter's pre-convergence phase, as this was believed to be relevant with respect to the robustness of the guidance system. However, the results obtained throughout this work, as well as the just-presented sensitivity analysis, revealed the small offect of such set of errors on the guidance system performance.

On the other hand, rotational dynamics proved to be a most stringent issue to be dealt with, and the filter itself did introduce additional challenges in this sense. Therefore, with hindsight, the time required for the implementation and troubleshooting of the filter would have been better spent towards the development of a guidance algorithm able to comply with rotational dynamics requirements, by, e.g., setting up the problem as a 6-DoF process. Ultimately, as already reiterated throughout this report, future work on the NN guidance presented here should go towards the extension of this method to incorporate attitude profiling, thus enabling a more realistic solution of the landing guidance problem.

8

CONCLUSIONS AND RECOMMENDATIONS

The research question this work set off to answer is the following:

Can a NN guidance system for terminal descent on Mars, trained within a RL framework, allow for a safe and accurate TD, while fulfilling the requirements that enable the operation of the HDA system?

If one were to answer this question in a direct way, one should say that the NN guidance presented here is not at a sufficient level of development to be implemented in a real case scenario. However, given the early stage in the development of artificial intelligence methods for dynamic problems in space, further work is needed to be able to properly assess whether these techniques could be used for such problems or not. Therefore, this question can be better answered indirectly, by singularly addressing the subquestions that have been defined in Section 2.3. This will give a clearer overview of the main findings of this work, the advantages of using NNs and RL, and, finally, the critical aspects which need the most improvement.

8.1. CONCLUSIONS

The first set of conclusions mainly refers to research question RQ-1, which is concerned with the performance of the algorithm in a baseline scenario. These are:

- A satisfactory value of propellant optimality could be achieved, which could be contained at $\sim 20\,\%$ with respect to the optimal value(RQ-1.2). A considerable advantage is present with respect to the Apollo guidance, as a difference of $\sim 45\,\text{kg}$ of propellant mass could be achieved (RQ-1.1). See Tables 6.2 and 6.3.

- On the other hand, the Apollo guidance performs better in terms of position and velocity accuracy, as it solves the TPBVP in closed-loop (see Table 6.3).

- Robustness against environmental and system uncertainties is, in general, more than satisfactory (RQ-1.4), as propellant consumption standard deviations are considerably lower than those of the Apollo guidance (see Figure 6.8).

- The neural network performs well in terms of coarse approximations for a wide range of different conditions, yet it lacks sheer accuracy in more specific situations (see Sections 6.2.1 and 6.2.2). Hence, while the performance of the Apollo guidance degrades when winds and navigations uncertainties are introduced, the NN performs more uniformly accross different environmental conditions (RQ-1.4).

- The robustness to large IC variations is good, yet not outstanding (see Figure 6.10). For specific areas of the IC-space, the Apollo guidance outperforms the network in terms of capability of achieving soft landing. On the other hand, as Apollo cannot handle control constraints, the network performs better for large negative vertical velocities, for which an efficient allocation of the available thrust is necessary (RQ-1.3).
- Once optimized, the NN on-board runtime is extremely limited (see Section 6.2.5). The tests performed show computational times in the order of those of the Apollo guidance (RQ-1.5). As such, a NN guidance algorithm can be run on-board in real-time (GR-10).

As mentioned above, an end-to-end NN guidance lacks the capability of pinpointing the target final conditions with the level of accuracy of the Apollo guidance. However, if a two-phase PDG approach is followed, where the NN delivers the vehicle to a set of interface conditions, from which the Apollo guidance brings it to the LS, multiple advantages can be achieved. Most importantly, these are:

- Increased position and velocity accuracy.
- Reduced propellant consumption:
  - Propellant optimality in the baseline scenario could be further improved down to 17 % penalty with respect to the optimal value (RQ-1.2).
  - Worst case propellant consumption in the baseline scenario could be further decreased by ~ 12 kg (RQ-1.1).
- Increased robustness against off-nominal ICs:
  - Without extending the training domain, robustness against varying IC is now clearly improved, as the network performs better than the Apollo guidance (RQ-1.3).
- Improved convergence speed of the optimization algorithm:
  - The convergence of the stochastic search optimization algorithm could be improved by approximately a factor ten with respect to a simple, end-to-end NN guidance approach (see Section 7.1.3).
  - Position and velocity accuracy do not represent a trade-off element of the optimization process anymore, as an Apollo-level accuracy performance is almost always achieved within the first phase of the optimization. Position and velocity convergence could not be reached for less than 10 % of the optimization instances, which is a low rate of occurrence, considering the random nature of the optimization.

The part of this work relative to the extended scenarios has therefore been carried out by using this Hybrid Apollo guidance and NN Approach (HANNA). The findings are summarized in the following.

- RL did prove to be a suitable framework for the implementation of the PCs, which could be formulated in a straightforward way (RQ-2.1). However, due to the limitations of the indirect policy optimization method used (in particular, its incapability to selectively adjust the NN parameters), a simplified, two-dimensional formulation had to be used for this work.
- When the network is optimized with respect to PCs only (i.e., no RTs), a pointing accuracy is achieved that allows for a 30^{o} FOV, as opposed to the 40^{o} FOV required by the Apollo guidance. Besides, the latter lacks robustness against varying FOV widths, as LS visibility degrades rapidly for narrower FOVs (RQ-3.2). See Figure 7.11.

- When the network is trained for the full scenario, the achievable pointing accuracy decreases considerably. A 60^o FOV is the minimum width which allows for LS visibility in the desired altitude range (RQ-2.3). This is still within realistic limits for typical camera FOVs, but it comes at the cost of a large fuel penalty (see Figure 7.12 and Table 7.5).

- Propellant performance degrades gradually when either PCs or RTs are included in the training environment, with penalty factors with respect to the optimal value of about 20 % and 25 %, respectively. Conversely, when both elements are included (i.e., full scenario), an acceptable propellant consumption value cannot be achieved, as inoptimality increases up to 37 % (RQ-2.4.2, RQ-2.5.2 and RQ-2.6). See Tables 7.2, 7.3 and 7.5.

- The envelope of feasible ICs increases substantially when RTs are included in the training environment (RQ-2.4.1 and RQ-2.6). Stress cases were run with IC variations twice the nominal 3σ values used for the optimization process. The networks can now achieve soft landing from almost all these ICs, except for very large negative vertical velocities ($\sim 100\,\mathrm{m/s}$). This proves the great generalization capabilities of NNs. See Figure 7.19.

- A RT area more than in compliance with the requirement GR-06 could be achieved when the network is specifically trained for this scenario. The tested RT area is approximately $2000 \times 1600\,\mathrm{m}$ in the downrange and crossrange directions, respectively. Almost all points within this area can be reached from all worst case ICs, and further distances, outside the tested area, are deemed attainable in the crossrange and positive downrange directions (RQ-2.4.3 and RQ-2.6). See Figures 7.9 and 7.13.

From the answers to the research questions given in the previous discussion, one can conclude that most of the guidance requirements set in Section 2.2.1 could be met successfully. However, some of them, namely GR-07, GR-09 and GR-11, do require further attention. In particular, these issues are mostly related to the nature of the RL framework and of the optimization method used. The following discussion provides a summary of these aspects.

- The random perturbations which are applied to the NN weights during the stochastic search of the parameter-space produce large gradients in the NN output, which ultimately lead to high rotational rates. Limiting these rates to the required value of $3^o/\mathrm{s}$ (GR-07) was not possible as part of this work. This has been attempted both by means of brute-force rate saturation during the training process, and by means of an additional term in the reward function. None of these approaches yielded satisfactory results, as they only allowed for a minimal level of spin rate reduction. In fact, the rate saturation value closest to the required one for which safe landing could be achieved with a rate limiter in the loop is $\omega_{thr} = 18^o/\mathrm{s}$, which is unrealistically high from a rotational dynamics point of view. The RL method thus needs major improvements in this direction. On the other hand, slower rotational rates are intrinsic to the Apollo guidance policy, which produces feasible trajectories with attitude rate saturation at $3^o/\mathrm{s}$ (RQ-2.3).

- Given the fact that the internal structure of the NN output is not easily visualized, and that its output results from a stochastic process, and therefore very irregular, verifiability of the policy is a major issue of this method (GR-11). Inspection methods different than mere MC simulations need to be investigated if a proper verification of the guidance system is to be performed.

- The stochastic optimiziation method has been found to be slow and, therefore, tedious to tune, as many instances have to be run before an insight into the effect of the chosen parameters can be gained. Also, consistency over instances is an issue, as many optimizations with good parameter combinations get stuck in far-from-optimal local minima.

- A formulation for the mass component of the reward function has been developed, which differs from the one originally proposed by Gaudet and Furfaro [2014]. This allows for a better adjustment of the optimality/accuracy trade-off (RQ-4.1). On the other hand, tuning the PC component of the reward does not seem to introduce any trade-off with fuel optimality, as both metrics vary accordingly with varying PC tuning parameters.
- The optimization algorithm requires a proper set of initial NN parameters for convergence to be achieved. It has been found that initializing it by means of networks obtained from previous RL-optimizations (e.g., in simpler scenarios) brings about significant advantages in terms of optimization time and final achievable reward.
- Another critical drawback is that of autonomy (GR-09). In fact, if the lander finds itself outside the known state-space domain, it will most likely result in failures. For ensuring reliability over larger state-space domains, these extended conditions need to be included in the training environment, at the cost of exponentially larger optimization times and design efforts. This is a major drawback as compared to deterministic methods, which compute their solution online, thus allowing for feasible trajectories from all conditions which are within their capabilities, regardless of the size of the corresponding domain.

In this section, many positive aspects have highlighted of using NNs within a RL optimization framework for the PDG phase. However, as presented above, there are a number of issues which cannot be disregarded, as they represent significant barriers to the implementation of this method for a real case scenario. In particular, these are the high rotational rates encountered during the descent, which could not be limited with the optimization method used here, and the issues of autonomy and verifiability. As such, in the next section, recommendations will be given for future work on this topic, with a particular focus with respect to these aspects.

8.2. RECOMMENDATIONS

As already discussed, the NN proved to be efficient at achieving a generalized approximation of the optimal policy, but it lacks the accuracy required to precisely pinpoint the final conditions. Considering this, from a mission design point of view, the most promising way forward in using NNs for the PDG phase is believed to be that of a two-phase solution, where the NN delivers the lander to a set of interface conditions, from where a more precise, yet possibly less optimal guidance method will bring it to a safe and accurate TD. As such, recommendations in this direction are presented below.

- Currently, a fixed altitude value is used as the trigger for switching from the NN to the Apollo guidance. A better trigger needs to be investigated for the transition between the two phases, so as to minimize total propellant consumption and attitude discontinuities, and maximixe landing accuracy.
- To this end, the two problems are better treated separately. This increases the complexity of the problem, as iterations are required to optimize the system as a whole, but it allows for a better insight on the interface conditions.
- Possible methods to be investigated for the second-phase are higher-order polynomial guidance algorithms, which allow to specify the initial acceleration (non possible in the 2nd-order Apollo guidance), thus constraining attitude changes at the interface, while still providing a computationally light solution.

However, the issue of large rotational rates is possibly the most critical issue that was found in this work. In fact, it does not make sense to consider the method for a real application, if it is not able to comply with realistic rotational dynamics requirements in a simulation environment. Possible approaches in this direction are given here.

- The most straightforward, yet uncertain, solution could be to apply an *a-posteriori* regularization of the NN output after the RL optimization phase, for instance by means on supervised learning techniques. Also, this could possibly ease the verification procedure by simplifying the structure of the NN output.
- A value-function-based policy optimization method is believed to be beneficial to the implementation of more complex constraints (such as pointing and rotational kinematic constraints), as it allows for a more selective adjustment of the NN parameters. A method for temporal-difference learning, which converges with non-linear FAs, has been developed by Maei et al. [2009].
- A more elegant method would be that of implementing a 6-DoF guidance, bringing together translational and rotational dynamics into one unified framework. Since rotational accelerations would then be a direct output of the network, it should be possible to better handle torque magnitude constraints. While this is believed to be beneficial per se, it can also be combined with a value-function-based method, which could possibly yield further advantages.
- The inclusion of pointing and rotational kinematic constraints in the OCP solved by GPOPS (or other OCS), for the generation of the supervised training data (see Section 5.2.2), would be beneficial to the subsequent optimization of these elements in the RL framework.

Finally, in the following are additional recommendations, not directly related, yet possibly beneficial, to the issues discussed above.

- As far as this specific optimization method is concerned, the implementation of an adaptive weight perturbation scale factor is advised, which can allow the optimization to get unstuck from local minima and improving convergence times.
- A PC reward formulation more representative of the desired results in terms of HDA requirements, which maximizes, for instance, the duration of continuous LS visibility (see discussion of Figure 7.11).
- A three-dimensional formulation of PCs, which would enable HDA after the first retargeting has occured, thus allowing for a second divert, if needed.
- The inclusion of a second retargeting at lower altitudes.
- The implementation of glide-slope constraints.

BIBLIOGRAPHY

Abilleira, F., Frauenholz, R., Fujii, K., Wallace, M., and You, T.-H. "2016 Mars Insight Mission Design and Navigation". In *24th AAS/AIAA Space Flight Mechanics Meeting*, Santa Fe, New Mexico, Jan. 2014.

Açıkmeşe, B., Casoliva, J., Carson, J. M., and Blackmore, L. "G-FOLD: A Real-Time Implementable Fuel Optimal Large Divert Guidance Algorithm for Planetary Pinpoint Landing". In *Concepts and Approaches for Mars Exploration*, volume 1679 of *LPI Contributions*, page 4193, Houston, Texas, June 2012.

Adler, M., Wright, M., Campbell, C., Clark, I., Engelund, W., and Rivellini, T. "Entry, Descent, and Landing Roadmap", Apr. 2012. `https://www.nasa.gov/pdf/501326main_TA09-ID_rev5_NRC_wTASR.pdf` [accessed: 2017-07-04].

Arvidson, R., Adams, D., Bonfiglio, G., Christensen, P., Cull, S., Golombek, M., Guinn, J., Guinness, E., Heet, T., Kirk, R., Knudson, A., Malin, M., Mellon, M., McEwen, A., Mushkin, A., Parker, T., Seelos, F., Seelos, K., Smith, P., Spencer, D., Stein, T., and Tamppari, L. "Mars Exploration Program 2007 Phoenix landing site selection and characteristics". *Journal of Geophysical Research: Planets*, 113 (E3), Mar. 2008. ISSN 2156-2202. doi: 10.1029/2007JE003021. E00A03.

Babuska, R. "Knowledge-Based Control Systems: Lecture Notes for the Course SC4080". Delft University of Technology, 2014.

Bayle, O., Lorenzoni, L., Blancquaert, T., Langlois, S., Walloschek, T., Portigliotti, S., and Capuano, M. "Exomars Entry Descent And Landing Demonstrator Mission And Design Overview". *NASA Solar System*, 2016. `https://solarsystem.nasa.gov/docs/Bayle_ExoMars_EDM_Overview-Paper.pdf` [online paper].

Bonfiglio, E. P., Adams, D., Craig, L., Spencer, D. A., Arvidson, R., and Heet, T. "Landing-Site Dispersion Analysis and Statistical Assessment for the Mars Phoenix Lander". *Journal of Spacecraft and Rockets*, 48(5):784–797, 2011. doi: 10.2514/1.48813.

Braun, R. D. and Manning, R. M. "Mars Exploration Entry, Descent, and Landing Challenges". *Journal of Spacecraft and Rockets*, 44(2):310–323, 2007. doi: 10.2514/1.25116.

Buonocore, M., Ospina, J., Canuto, E., Bacchetta, A., Calantropio, F., Martelli, A., and Oddenino, D. "Guided Entry : A Necessary Step from Exomars to Precision Landing on Mars". In *4th European Conference for Aerospace Sciences (EUCASS)*, Saint Petersburg, Russia, July 2011.

Cheng, Y., Goguen, J., Johnson, A., Leger, C., Matthies, L., Martin, M. S., and Willson, R. "The Mars Exploration Rovers Descent Image Motion Estimation System". *IEEE Intelligent Systems*, 19(3): 13–21, May 2004. ISSN 1541-1672. doi: 10.1109/MIS.2004.18.

Choukroun, D. "Spacecraft Attitude Determination: Lecture Notes for the Course AE4S19". Delft University of Technology, 2014.

Cook, R. A. and McNamee, J. B. "Mission Design for the Mars Environmental Survey". In *Astrodynamics 1993: Proceedings of the AAS/AIAA Astrodynamics Conference Held August 16-19, 1993,*

Victoria, British Columbia, Canada, volume 85 of *Advances in astronautical sciences*, pages 967–981. AAS, 1993.

Cooley, C. G. "Viking 75 Project: Viking Lander System Primary Mission Performance Report". NASA-CR-145148, Martin Marietta Corporation, Baltimore, MD, U.S., Apr. 1977.

Cybenko, G. "Approximation by superpositions of a sigmoidal function". *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec. 1989. ISSN 1435-568X. doi: 10.1007/BF02551274.

Desai, P. N., Prince, J. L., Queen, E. M., Schoenenberger, M. M., Cruz, J. R., and Grover, M. R. "Entry, Descent, and Landing Performance of the Mars Phoenix Lander". *Journal of Spacecraft and Rockets*, 48(5):798–808, Sept. 2011. doi: 10.2514/1.48239.

Edquist, K. T., Hollis, B. R., Dyakonov, A. A., Laub, B., Wright, M. J., Rivellini, T. P., Slimko, E. M., and Willcockson, W. H. "Mars Science Laboratory Entry Capsule Aerothermodynamics and Thermal Protection System". In *2007 IEEE Aerospace Conference*, pages 1–13, Big Sky, MT, U.S., March 2007. doi: 10.1109/AERO.2007.352823.

Foessel-Bunting, A. and Whittaker, W. "MMW-Scanning Radar for Descent Guidance and Landing Safeguard". In *Proceedings of the 6th International Symposium on Artificial Intelligence, Robotics, and Automation in Space: i-SAIRAS 2001*, Montreal, Canada, June 2001.

Främling, K. "Replacing eligibility trace for action-value learning with function approximation". In Verleysen, M., editor, *15th European Symposium on Artificial Neural Networks*, pages 313–318, Bruges, Belgium, Apr. 2007.

Gaudet, B. and Furfaro, R. "Adaptive pinpoint and fuel efficient Mars landing using reinforcement learning". *IEEE/CAA Journal of Automatica Sinica*, 1(4):397–411, Oct. 2014. ISSN 2329-9266. doi: 10.1109/JAS.2014.7004667.

Gill, P. E., Murray, W., and Saunders, M. A. "SNOPT: An SQP algorithm for large-scale constrained optimization". *SIAM Review*, 47(1):99–131, 2005. doi: 10.1137/S0036144504446096.

Ingoldby, R. N. "Guidance and Control System Design of the Viking Planetary Lander". *Journal of Guidance, Control, and Dynamics*, 1(3):189–196, May 1978. doi: 10.2514/3.55763.

Johnson, A. E., Huertas, A., Werner, R. A., and Montgomery, J. F. "Analysis of On-Board Hazard Detection and Avoidance for Safe Lunar Landing". In *2008 IEEE Aerospace Conference*, pages 1–9, Big Sky, MT, U.S., March 2008. doi: 10.1109/AERO.2008.4526301.

Johnson, A. E., Cheng, Y., Montgomery, J. F., Trawny, N., Tweddle, B., and Zheng, J. X. "Real-Time Terrain Relative Navigation Test Results from a Relevant Environment for Mars Landing". In *AIAA SciTech Forum*. American Institute of Aeronautics and Astronautics, Kissimmee, FL, U.S., Jan. 2015. doi: 10.2514/6.2015-0851.

Justus, C. and Braun, R. "Atmospheric Environments for Entry, Descent and Landing (EDL)". In *5th International Planetary Probe Workshop*, Bordeaux, France, June 2007.

Klumpp, A. R. "Apollo lunar descent guidance". *Automatica*, 10(2):133–146, 1974. doi: 10.1016/0005-1098(74)90019-3.

Knocke, P. C., Wawrzyniak, G. G., Kennedy, B. M., Desai, P. N., Parker, T. J., Golombek, M. P., Duxbury, T. C., and Kass, D. M. "Mars Exploration Rovers Landing Dispersion Analysis". In *AIAA/AAS Astrodynamics Specialist Conference and Exhibit*, Providence, RI, U.S., 2004. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.2004-5093.

Maei, H. R., Szepesvári, C., Bhatnagar, S., Precup, D., Silver, D., and Sutton, R. S. "Convergent Temporal-Difference Learning with Arbitrary Smooth Function Approximation". In *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, pages 1204–1212, Istanbul, Turkey, Nov. 2009.

Manning, R. and Adler, M. "Landing on Mars". In *AIAA Space Conference*, Long Beach, CA, U.S., Sept. 2005.

Martella, P., Buonocore, M., Canuto, E., Molano-Jimenez, A., Drai, R., and Lorenzoni, L. "Design and Verification of the GNC for the European ExoMars EDL Demonstrator". In *AIAA Guidance, Navigation, and Control Conference*, Portland, OR, U.S., August 2011. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-952-5. doi: 10.2514/6.2011-6341.

Masursky, H. and Crabill, N. L. "Viking site selection and certification". *NASA Special Publication*, 429, 1981.

Mendeck, G. "Mars Science Laboratory Entry Guidance". In *AIAA Atmospheric Flight Mechanics Conference*, Portland, OR, U.S., Aug. 2011. American Institute of Aeronautics and Astronautics.

Millour, E., Forget, F., Spiga, A., Navarro, T., Madeleine, J.-B., Montabone, L., Pottier, A., Lefevre, F., Montmessin, F., Chaufray, J.-Y., Lopez-Valverde, M. A., Gonzalez-Galindo, F., Lewis, S. R., Read, P. L., Huot, J.-P., Desjean, M.-C., and MCD/GCM development Team. "The Mars Climate Database (MCD version 5.2)". In *European Planetary Science Congress 2015*, volume 10, Nantes, France, Oct. 2015.

Mooij, E. *The motion of a vehicle in a planetary atmosphere*. Delft University of Technology, 1994. ISBN 90-5623-003-4.

Moseley, W. C., Graham, R. E., and Hughes, J. E. *Aerodynamic stability characteristics of the Apollo command module*. NASA TN D-4688, Aug. 1968.

Mueller, E., Bilimoria, K., and Frost, C. "Improved Lunar Lander Handling Qualities through Control Response Type and Display Enhancements". In *AIAA Guidance, Navigation, and Control Conference*, Toronto, Canada, Aug. 2010. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.2010-8025.

Paschall, S., Brady, T., and Sostaric, R. "Lunar Landing Trajectory Design for Onboard Hazard Detection and Avoidance". In *32nd Annual AAS Guidance and Control Conference*, Breckenridge, CO, U.S., Jan. 2009. American Astronautical Society.

Portigliotti, S., Dumontel, M., and Capuano, M. "Landing Site Targeting and Constraints for Exomars 2016 Mission". In *7th International Planetary Probe Workshop*, pages 14–18, Barcelona, Spain, June 2010.

Rao, A., Benson, D., Huntington, G., Francolin, C., Darby, C., and Patterson, M. "User's Manual for GPOPS: a MATLAB package for dynamic optimization using the Gauss pseudospectral method". *University of Florida report*, 2008.

Rao, A. V., Benson, D. A., Darby, C., Patterson, M. A., Francolin, C., Sanders, I., and Huntington, G. T. "Algorithm 902: GPOPS, a Matlab software for solving multiple-phase optimal control problems using the gauss pseudospectral method". *ACM Transactions on Mathematical Software*, 37(2), 2010. ISSN 1557-7295 0098-3500.

Rogata, P., Sotto, E. D., Câmara, F., Caramagno, A., ao, J. M. R., Correia, B., Duarte, P., and Mancuso, S. "Design and performance assessment of hazard avoidance techniques for vision-based landing". *Acta Astronautica*, 61(1-6), 2007. doi: 10.1016/j.actaastro.2007.01.030.

Rumelhart, D. E., Hinton, G. E., and Williams, R. J. "Learning Internal Representations by Error Propagation". In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1, pages 318–362. MIT Press, Cambridge, MA, U.S., 1986. ISBN 0-262-68053-X.

San Martin, M. "The Evolution of Guidance, Navigation, and Control in Mars Entry, Descent and Landing", May 2013. `https://mediaex-server.larc.nasa.gov/Academy/Play/a5201882db5441778bd661603dd8c9b41d` [accessed: 2015-12-01].

Soffen, G. A. and Thomas Young, A. "The Viking missions to Mars". *Icarus*, 16(1), Feb. 1972. ISSN 0019-1035. doi: 10.1016/0019-1035(72)90133-9.

Sostaric, R. "Powered descent trajectory guidance and some considerations for human lunar landing". In *2007 AAS Guidance and Control Conference*, Breckenridge, CO, U.S., Feb. 2007. American Astronomical Society.

Starek, J. A., Açıkmeşe, B., Nesnas, I. A., and Pavone, M. *Spacecraft Autonomy Challenges for Next-Generation Space Missions*, pages 1–48. Springer Berlin Heidelberg, 2016. ISBN 978-3-662-47694-9. doi: 10.1007/978-3-662-47694-9_1.

Sutton, R. S. and Barto, A. G. *Introduction to Reinforcement Learning*. MIT Press, Cambridge, MA, U.S., 1st edition, 1998. ISBN 0262193981.

Topcu, U., Casoliva, J., and Mease, K. D. "Minimum-Fuel Powered Descent for Mars Pinpoint Landing". *Journal of Spacecraft and Rockets*, 44(2):324–331, Mar. 2007. ISSN 0022-4650. doi: 10.2514/1.25023.

Uhlig, T., Sellmaier, F., and Schmidhuber, M. *Spacecraft Operations*. Springer Vienna, 2015. ISBN 978-3-7091-1803-0. doi: 10.1007/978-3-7091-1803-0.

Vago, J., Witasse, O., Svedhem, H., Baglioni, P., Haldemann, A., Gianfiglio, G., Blancquaert, T., McCoy, D., and de Groot, R. "ESA ExoMars program: The next step in exploring Mars". *Solar System Research*, 49(7):518–528, Dec. 2015. ISSN 1608-3423. doi: 10.1134/S0038094615070199.

Way, D. W., Powell, R. W., Chen, A., Steltzner, A. D., Martin, A. M. S., Burkhart, P. D., and Mendeck, G. F. "Mars Science Laboratory: Entry, Descent, and Landing System Performance". In *2007 IEEE Aerospace Conference*, pages 1–19, Big Sky, MT, U.S., Mar. 2007. IEEE. ISBN 1-4244-0524-6. doi: 10.1109/AERO.2007.352821.

Way, D. W., Davis, J. L., and Shidner, J. D. "Assessment of the Mars Science Laboratory Entry, Descent, and Landing Simulation". In *23rd AAS/AIAA Space Flight Mechanics Meeting*, Kauai, HI, U.S., Feb. 2013. AAS/AIAA.

Webster, G. "'Lost' 2003 Mars Lander Found by Mars Reconnaissance Orbiter", Jan. 2015. `http://www.nasa.gov/jpl/lost-2003-mars-lander-found-by-mars-reconnaissance-orbiter/#.Vl3ix_mrQU0` [accessed: 2015-12-01].

Wertz, J. R., Meissinger, H. F., Newman, L. K., and Smit, G. N. *Orbit & Constellation Design & Management: Spacecraft Orbit and Attitude Systems*, volume 13 of *Space Technology Library*. Microcosm Press - Springer, 2nd edition, 2009. ISBN 978-1881883074.

Williams, D. R. “Chronology of Mars Exploration”, Aug. 2015. http://nssdc.gsfc.nasa.gov/planetary/chronology_mars.html [accessed: 2015-12-01].