6-DOF Atmospheric Rocket Landing Guidance using Meta-Reinforcement Learning

Jacopo Carradori

Delft University of Technology

TU Delft

DLR

6-DOF Atmospheric Rocket Landing Guidance using Meta-Reinforcement Learning

by

Jacopo Carradori

to obtain the degree of Master of Science in Aerospace Engineering
at the Delft University of Technology,
to be defended publicly on Friday, September 27, 2024, at 9:00.

Student number:	5732042	
Project Duration:	January, 2024 - September, 2024	
Thesis Committee:	Dr. J. Guo,	TU Delft, Chair
	Dr. ir. E. Mooij,	TU Delft, Supervisor
	Dr. ir. M. Sagliano,	DLR, External supervisor
	Dr. ir. E. van Kampen,	TU Delft, Independent examiner

Cover: SpaceX's Falcon Heavy Arabsat 6A landing at Kennedy Space Center on April 12, 2019. U.S. Air Force photo by James Rainier. Available at `https://www.flickr.com/photos/usairforce/48000673302/`.

Style: TU Delft Report Style, with modifications by Daan Zwaneveld.

TU Delft

Preface

This work marks the last deliverable of my Master's degree in Aerospace Engineering, at the Delft University of Technology. It focuses on the design, implementation, and analysis of a 6-Degree-of-Freedom guidance and control algorithm for powered rocket landing on Earth, combining the innovative technique of meta-Reinforcement Learning with Transformers-based Neural Network. It has been challenging, but also extremely fascinating to work on such cutting-edge techniques applied to one of the most interesting space problems.

The thesis closes this chapter of my life that started two years ago when I left, once again, my dear Italy. Since then, there has been no day without learning something new. It has been a wonderful adventure full of amazing people I'd like to thank.

First, I want to express my deep gratitude to my thesis supervisors Dr. Ir. Erwin Mooij and Dr. Ir. Marco Sagliano, who guided me through the journey, offering invaluable advice and helping me grow as an engineer. Erwin, thank you for offering me this incredible opportunity, and most importantly, for your continuous support, guidance, and encouragement to always push beyond my limits. Marco, thank you for the dedication, passion, and commitment you consistently shared with me, stimulating my curiosity. They are a great source of inspiration that I will carry with me into the future.

I would also like to thank the entire Guidance, Navigation, and Control department at DLR in Bremen, who hosted me for the last six months and gave valuable feedback on my work. Francesco, Pietro, Stefano, Tiago, thank you for the conversations and memories shared in Bremen, that helped me unwind and escape the stress of the thesis.

My most heartfelt thank you goes to my family, my number one supporter from day one. I am extremely grateful for all the opportunities you give me to chase my dreams. I would have not achieved anything without you. Mamma, Babbo, and Ginevra, it does not matter how far we are, you are always the closest thing to my heart. This thesis is dedicated to you.

A big thanks goes to Jan, Daniella, Alana, and all the Belgian friends who entered my life seven years ago and never left. Sharing time with you always opens my mind to different perspectives.

A special thanks goes to the "Dei dell'Olimpo" (+ João), my dear friends who always kept me motivated and cheerful during our first year in Delft, as well as when we were spread all over the world. I am flattered to have found you along the way to share this ride, and I wish that many more adventures together will come. You know how tough this journey has been, and your constant support was a key factor in achieving the "final reward".

I can't thank enough all my friends from Pistoia and Milano. You are always present to share laughs and moments of joy when I come back home, but you also never miss an opportunity to visit me wherever I am in my adventures. I am so lucky to have you with me.

A final heartful thanks goes to all the people not explicitly mentioned. Remember that you have been, or you are, part of my life, contributing to making me the person I am today.

I want to thank Delft High Performance Computing Centre for the use of computational resources of the DelftBlue supercomputer (Delft High Performance Computing Centre , DHPC).

Ultimately, a special thanks goes to Fondazione Cassa di Risparmio di Pistoia e Pescia, who provided me with a scholarship to pursue this study program. Your mission to promote the next generations of students is exceptionally valuable.

Jacopo Carradori
Pistoia, December 2024

Summary

Landing a rocket on Earth is a key factor in enabling quicker and more cost-effective access to space. However, it poses significant challenges due to the highly uncertain environment. A robust, reliable, and real-time capable Guidance, Navigation, and Control (GNC) system is essential to guide the vehicle to the landing site while meeting terminal constraints and minimizing fuel consumption. Although the landing problem has been extensively studied using simplified models, this work uses a more complex and uncertain environment, so that the simulations are closer to the operational scenario. A 6-Degrees Of Freedom (DOF) flight simulator is developed to account for the strong coupling between translational and rotational motion. Vehicle and environmental models such as variable Mass, Center of Mass and Inertia (MCI), winds, and detailed aerodynamics are included. Furthermore, not only thrust magnitude and engine deflections are used as controls, but also two sets of aerodynamic fins, as they are present in real rockets. Finally, initial condition dispersions and uncertainties in dynamics, navigation, and controls are included, to assess the robustness of the GNC strategy.

This study applies Meta Reinforcement Learning (meta-RL) to the terminal rocket landing problem. Through repeated interactions between an agent and its environment over multiple episodes, a Neural Network (NN) develops a policy that maps observed states to control actions. Unlike traditional methods, meta-RL leverages both current and past observations to output the optimal action, enhancing the robustness of the policy. Recurrent Neural Networks (RNNs) like Long-Short Term Memory (LSTM), are commonly used in meta-RL for their ability to handle sequential data. However, this thesis investigates also the use of attention-based Gated Transformer XL (GTrXL) networks, which are promising to improve the solution accuracy. Similar attention-based networks are employed in Large Language Models, like ChatGPT, where they have shown excellent performance in learning long-range dependencies in sequences.

This research confirms this hypothesis, demonstrating that the GTrXL-based policy significantly outperforms the LSTM-based one. This policy is also less sensitive to internal hyperparameters, provided that the NN has a sufficient number of weights and biases to capture all the relevant features. The results indicate that incorporating complex vehicle and environmental models, along with dispersed initial conditions, aerodynamic and wind uncertainties, navigation and control errors, and actuator deflection rate constraints into the training process, yields a robust GTrXL-based policy. This guidance policy is able to meet terminal constraints on position, horizontal velocity, vertical angle, and angular rates in 1000/1000 simulations. The only exception is the vertical velocity which is exceeded on average by only 2-3 m/s, depending on the network's hyperparameters chosen. Including a thrust rate constraint mitigates this issue, reducing the average violation to 1 m/s. Finally, developing a terminal patch to increase the thrust magnitude in the last few meters before touchdown completely eliminates the vertical velocity issue, producing a 100% success rate, with all the Monte Carlo runs meeting all the terminal constraints. Moreover, the NN produces a solution in about 6 ms, showing great potential for its real-time use. The results are consistently superior to those of a conventional Guidance and Control (G&C) strategy where a Linear Quadratic Regulator (LQR) controller tracks an optimal trajectory, consuming only 6% more fuel.

Recommendations for future work include improving the reward function to better handle the vertical velocity constraint, and penalizing control effort to have smoother Thrust Vector Control (TVC) and fins control profiles. It is also suggested to expand the simulation scenario, including the unpowered aerodynamic descent.

Contents

List of acronyms

AI	Artificial Intelligence
ANN	Artificial Neural Network
BC	Behavioral Cloning
CNES	Centre National d'Etudes Spatiales
CoM	Center of Mass
CP	Center of Pressure
DCM	Direction Cosine Matrix
DDPG	Deep Deterministic Policy Gradient
DLR	Deutsches Zentrum für Luft und Raumfahrt
DNN	Deep Neural Network
DOF	Degrees Of Freedom
DQN	Deep Q-Network
DRL	Deep Reinforcement Learning
ECEF	Earth Centered Earth Fixed
ECI	Earth Centered Inertial
EDL	Entry, Descent, and Landing
EOMs	Equations Of Motion
ESA	European Space Agency
G-FOLD	Guidance for Fuel-Optimal Large Diverts
GAE	Generalized Advantage Estimator
G&C	Guidance and Control
GNC	Guidance, Navigation, and Control
GNSS	Global Navigation Satellite System
GRAM	Global Reference Atmosphere Model
GTrXL	Gated Transformer XL
HWM14	Horizontal Wind Model 14
IMU	Inertial Measurement Unit
ISS	International Space Station
JAXA	Japanese Aerospace Exploration Agency
LCvx	LossLess convexification
LGR	Legendre-Gauss-Lobatto
LQR	Linear Quadratic Regulator
LSTM	Long-Short Term Memory
LTI	Linear Time Invariant
MAML	Model-Agnostic Meta Learning
MCI	Mass, Center of Mass and Inertia
MDP	Markov Decision Process

meta-RL	Meta Reinforcement Learning
MIMO	Multiple Input Multiple Output
ML	Machine Learning
MLP	Multi Layer Perceptrons
MRP	Modified Rodriguez Parameters
MSL	Mars Science Laboratory
NASA	National Aeronautics and Space Administration
NLP	Non Linear Programming
NN	Neural Network
OCP	Optimal Control Problem
PDF	Probability Distribution Function
PID	Proportional, Integral, Derivative
POMDP	Partially Observable Markov Decision Process
PPO	Proximity Policy Optimization
RCS	Reaction Control System
RK4	Runge Kutta 4th order
RL	Reinforcement Learning
RLV	Reusable Launch Vehicle
RNN	Recurrent Neural Network
SAC	Soft-Actor Critic
SCvx	Sequential Convexification
SGD	Stochastic Gradient Descent
SL	Supervised Learning
SPCP	Sequential Pseudospectral Convex Programming
TD3	Twin Delayed Deep Deterministic Policy Gradient
TPBVP	Two Point Boundary Value Problem
TRN	Terrain Relative Navigation
TRPO	Trust Region Policy Optimization
TVC	Thrust Vector Control
US76	U.S. Standard Atmosphere 1976
USA	United States of America
USSR	Union of Soviet Socialist Republics
VTVL	Vertical Takeoff Vertical Landing
ZEM	Zero-Error-Miss
ZEV	Zero-Error-Velocity

List of Symbols

Greek symbols

Symbol	Description	Units
α	Angle of attack	rad
α_T	Total angle of attack	rad
β	Sideslip angle	rad
$\beta_{fin,pitch,i}$	Deflection angles i-th pitch fin	rad
$\beta_{fin,yaw,i}$	Deflection angles i-th yaw fin	rad
β^*	Auxiliary sideslip angle	rad
δ	Vertical angle	rad
ϵ_T	TVC deflection angle around $Y_{\mathcal{P}}$-axis	rad
$\boldsymbol{\theta}$	Neural network's parameters (weights and biases)	-
θ	Pitch angle	rad
μ	Mean value	Various
$\mu_\oplus$	Earth gravitational parameter	m^3/s^2
$\boldsymbol{\mu}$	Langrange multipliers of path constraints	Various
$\boldsymbol{\lambda}$	Costates (or adjoints)	Various
π	Neural network policy	-
ρ	Atmospheric density	kg/m^3
$\boldsymbol{\sigma}$	Neural network activation function	-
σ	Standard deviation uncertainty	Various
ϕ	Roll angle	rad
ψ	Yaw angle	rad
ψ_T	TVC deflection angle around $Z_{\mathcal{P}}$-axis	rad
$\boldsymbol{\omega}$	Rotation rate vector	rad/s
ω_E	Earth rotation rate	rad/s
ω_x	Roll rate around $X_{\mathcal{B}}$-axis	rad/s
ω_x	Pitch rate around $Y_{\mathcal{B}}$-axis	rad/s
ω_x	Yaw rate around $Z_{\mathcal{B}}$-axis	rad/s

Latin symbols

Symbol	Description	Units
A_π	Advantage function	-
$\mathbf{a}$	Acceleration vector	m/s^2
$\mathbf{a}$	Actions vector	Various
b_{ref}	Wingspace	m
c_{ref}	Aerodynamic chord	m
C_A	Axial force coefficient	-
C_D	Drag coefficient	-
C_{fin}	Maximum fins aerodynamic normal coefficient	-
C_L	Lift coefficient	-
C_N	Normal force coefficient	-
C_X	Aerodynamic coefficient along $X_{\mathcal{B}}$-axis	-
C_Y	Aerodynamic coefficient along $Y_{\mathcal{B}}$-axis	-

C_Z	Aerodynamic coefficient along $Z_{\mathcal{B}}$-axis	-
$\mathbf{F}$	Force vector	N
$\mathbf{f}$	Equations of motion	Various
$\mathbf{g}$	Path constraints	Various
$\mathcal{H}$	Hamiltonian	kg
$\mathbf{K}$	LQR gain matrix	Various
$\mathbf{K}$	Transformer keys matrix	-
$\mathbf{J}$	Inertia tensor	kg/m^2
I_{sp}	Engine specific impulse	s
$\mathcal{L}$	Lagrangian	kg
$\mathbf{M}$	Moment vector	Nm
M	Mach number	-
$\mathbf{Q}$	LQR state error weight matrix	Various
$\mathbf{Q}$	Transformer queries matrix	-
Q_π	State-value function	-
$\mathbf{q}$	Quaternion vector	-
$\bar{q}$	Dynamic pressure	Pa
q_c	Heat flux	W/m^2
$\bar{q}\alpha$	Dynamic loads	KPa·deg
$\mathbf{R}$	LQR control effort weight matrix	Various
$R_\oplus$	Earth equatorial radius	m
R	Reward function	-
R_N	Rocket curvature radius	m
$\mathbf{r}$	Position vector	m
r_x	Position with respect to X_{UEN}-axis	m
r_y	Position with respect to Y_{UEN}-axis	m
r_z	Position with respect to Z_{UEN}-axis	m
S_{ref}	Reference area	m^2
S_{fin}	Fin reference area	m^2
$\mathbf{s}$	Observations vector	Various
m	Mass	kg
T	Thrust	N
$\mathbf{u}$	Control vector	Various
$\mathbf{V}$	Transformer values matrix	-
V_π	Value function	-
V	Candidate Lyapunov function	-
$\mathbf{v}$	Velocity vector	m/s
v_x	Velocity with respect to X_{UEN}-axis	m/s
v_y	Velocity with respect to Y_{UEN}-axis	m/s
v_z	Velocity with respect to Z_{UEN}-axis	m/s
$\mathbf{x}$	State vector	Various
X_{CP}	Longitudinal component center of pressure with respect to rocket bottom	m
X_{fins}	Longitudinal fin position with respect to rocket bottom	m
$\mathbf{x}_{CP}$	Center of pressure location	m

1 Introduction

1.1. Background Information

Early landing missions aimed to achieve a soft touchdown without destroying the vehicle. After succeeding in that, the focus shifted to improving landing precision and optimizing fuel consumption. Engineers developed various techniques to reduce landing dispersion and enhance fuel efficiency across different mission environments, while also addressing the challenges posed by environmental uncertainties, such as atmospheric conditions and wind effects.

This thesis focuses on the terminal atmospheric powered landing, in an Earth-like environment. It can be the last stage of the Entry, Descent, and Landing (EDL) sequence. For instance, for booster recoveries like the Falcon 9, the flight involves a suborbital atmospheric re-entry, an aerodynamic descent using fins, and a final engine burn for a controlled precise touchdown. The terminal powered landing starts in the last few kilometers above the landing site and it terminates at touchdown. Thrust magnitude, engine deflections, and fins are the control means used to correctly guide the rocket. Landing in an atmosphere is much more complex than on an airless body due to non-linear aerodynamic forces, requiring perfect coordination of thrust, fins, position, velocity, and attitude for an accurate touchdown (Sagliano et al., 2021a). The GNC system must be able to compensate for errors and drive the vehicle to the predetermined landing conditions. It is fundamental to account for all the interactions between the state variables because, at such low velocity, the translational and rotational motion are highly coupled. Hence, it is essential to correctly model the 6-DOF flight dynamics.

To achieve a precise pinpoint landing, the vehicle must have an extremely high terminal accuracy, while minimizing fuel consumption to avoid lowering payload mass. A Reusable Launch Vehicle (RLV) landing on Earth, should reduce errors on position (and velocity) within a few meters (and meters per second) while keeping the attitude (and angular velocities) within prescribed ranges to avoid tip-over right after touchdown. Achieving these goals in a dynamic and uncertain environment presents significant challenges to landing accuracy. Factors, like dispersed initial conditions, atmospheric characteristics, and navigation and control errors, are inherently uncertain and not precisely known in advance. The guidance system must be robust enough to handle these uncertainties.

Various methods are developed to enhance the robustness of landing guidance algorithms against uncertainties. One approach involves tracking an offline-generated optimal trajectory with a closed-loop controller (Wang, 2024), but this method struggles with large uncertainties and requires recalculating the trajectory if deviations occur. Real-time closed-loop guidance effectively addresses disturbances. Convex optimization methods enable real-time trajectory generation within the control loop, regularly updating guidance commands during flight (Szmuk et al., 2020). However, maintaining high-frequency updates usually requires assumptions and simplifications, potentially reducing accuracy.

Over the last decade, the use of Machine Learning (ML) has experienced a rapid surge in many scientific fields, including the space sector. NNs are particularly effective as function approximators for complex, nonlinear functions and they are capable of handling vast amounts of data. They have been successfully applied in space guidance algorithms, to discover patterns and generate optimal actions, generalizing and adapting to different, yet similar, situations. From Supervised Learning (SL) (Sánchez-Sánchez and Izzo, 2018) to Reinforcement Learning (RL) (Gaudet et al., 2020b; Federici and Furfaro, 2024; Rosa et al., 2023), many methods were employed in different uncertain scenarios, showing performance comparable to traditional guidance algorithms and offering increased robustness to disturbances and uncertainties. Furthermore, during their real-time deployment, they are computationally light, enabling a much higher update frequency, without the need for any simplification and assumption in the problem. These Artificial Intelligence (AI) approaches are very promising, thanks to rapid advancements in ML algorithms and models. This thesis addresses the strengths and weaknesses of different methods to identify the most suitable technique for increasing robustness to uncertainties.

However, ML applications in rocket landing often use low-fidelity models that lack realism, typically employing only thrust magnitude and engine deflections as controls, without considering rate constraints or 6-DOF dynamics. Many studies simplify (or neglect) aerodynamics and often exclude factors like winds or variable mass, resulting in overly simplistic vehicle models, far from a realistic scenario. This research aims to apply innovative ML techniques to a complex 6-DOF rocket landing problem on Earth, incorporating thrust, TVC, aerodynamic fins, angular rate constraints, detailed aerodynamics, randomized wind profiles, and variable mass models for greater realism, to reduce the simulation-reality gap.

The results obtained with these innovative techniques are compared with a conventional method that consists of an optimal trajectory generated with direct optimization methods and tracked with an LQR controller. This is a typical implicit guidance method, with a trajectory pre-generated offline considering a nominal unperturbed scenario. Then, the controller is synthesized to cope with dispersed initial conditions and perturbed dynamics.

Finally, different Monte Carlo campaigns are run, comparing the RL and the conventional G&C strategies, to understand how the two perform in such a complex, uncertain scenario.

1.2. Problem Statement

Landing on Earth presents significant challenges due to complex dynamics and atmospheric uncertainties that affect the vehicle's motion. Conventional guidance algorithms that have addressed these disturbances often rely on simplifications due to limited computational power. This research aims to develop a robust, adaptable guidance system for powered landings that can operate in real-time, handling environmental uncertainties, dispersed initial conditions, and achieving pinpoint landings, while minimizing fuel consumption. Given their success in data-intensive tasks, ML algorithms are promising candidates for this purpose. Their intrinsic ability to generalize policies may be a cornerstone to increase the robustness of powered landing guidance systems.

This thesis explores and tests ML techniques in a realistic 6-DOF scenario with high-fidelity models and uncertainties to evaluate their performance in conditions similar to an operational rocket landing scenario on Earth.

1.2.1. Research question

Based on the discussed challenges, goals, and problem statement, the aspects on which this research focuses are defined. Thus, the main research question is:

How can Machine Learning improve the robustness of a guidance system with real-time capability, for the powered landing phase in an uncertain (atmospheric) environment?

The main research question is then broken down into multiple subquestions that are addressed during the thesis, in a, more or less, chronological order.

- **[RQ 1] What is the best Machine Learning method to be used in a guidance system for an atmospheric powered landing mission scenario?**
- **[RQ 2] How can the training process of a Machine Learning algorithm be improved for a rocket landing guidance system?**
- **[RQ 3] How accurate is the performance of the developed guidance system in terms of mass consumption and terminal pose?**
- **[RQ 4] How does the Machine Learning-based guidance system perform compared to the state-of-the-art solutions in a strongly perturbed environment?**

1.3. Report Overview

The work developed in this report is spread across multiple chapters. Chapter 2 gives an overview of past missions and methodologies, focusing also on the definition of the mission, vehicle, and requirements of this thesis. Chapter 3 introduces the flight dynamics and environment models used in this work, while Chapter 4 explains the theory behind RL and NNs. Chapter 5 presents the verification of the 6-DOF flight simulator and the models' choices. The results of the conventional feedback guidance and control are outlined in Chapter 6, where the optimal trajectory generation is explained, as well as the LQR controller synthesis and tuning. The chapter also illustrates the results from two closed-loop Monte Carlo campaigns, with two different controller architectures. Chapter 7 describes the development and implementation of the meta-RL-based guidance and control strategy. Several policies with different hyperparameters, constraints, or formulations are explained and presented. The results are analyzed, comparing the performances between the different cases, to understand the strengths and weaknesses of such an approach. Finally, conclusions are drawn in Chapter 8, with the lessons learned from the work performed and recommendations for future possible extensions.

2

Mission Heritage

Section 2.1 gives an overview of methodologies and previously flown missions that attempted powered landing on Mars and Earth. Then, in Section 2.2 the main challenges related to precision landing in atmospheric environments are presented and explained. Section 2.3 gives an overview of the progress of guidance algorithms, from the first analytical implementations to the current state-of-the-art and future perspectives. In Sections 2.4 and 2.5 the reference vehicle and the mission scenario used in this manuscript are presented, respectively. Finally, Section 2.6 concludes the chapter with the introduction of the system and mission requirements that have to be fulfilled by the G&C strategy.

2.1. Powered Descent and Landing History

Landing on the surface of planets or moons has always been one of the primary goals of space exploration of the Solar System, to get additional measurements and insight. Moreover, with the start of the new space race in the 21st century, the paradigm shift to reusability has made a big change in the launch vehicles market. New designs include rockets able to return and land on Earth, before being launched again, reducing the cost of access to space.

So far, man-made objects reached the surface of a few planetary bodies, namely Earth, Moon, Mars, Venus, and Titan. The first successful powered landing took place on the Moon on February 3, 1966, with the Soviet mission Luna 9. On July 20, 1969, as part of the Apollo 11 mission, the Lunar Module "Eagle" landed on the surface of the Moon, achieving the first crewed landing. On Titan, reached by the European probe Huygens in 2005, and Venus, where the Russian Venera 7 set foot in 1970, no powered descent was executed, but parachutes and drag plates were used, given their high-density atmospheres. Furthermore, a spacecraft landed on a comet during the European Space Agency (ESA) Rosetta mission, and some "touch and go" have been performed on the surface of some asteroids. Samples have been collected from Itokawa, Ryugu, and Bennu, and brought back to Earth, in the Japanese missions Hayabusa, Hayabusa2, and the American Osiris-REx.

Among the future planned landing missions, there are Artemis on the Moon, ExoMars on Mars, MMX on its moon Phobos, DAVINCI on Venus, and Dragonfly on Titan.

This work focuses on atmospheric Earth-like landing, but the achievement of landing a rocket derives also from missions done in similar environments, such as the Martian one.

Mars

Mars exploration started during the first Space Race. After a few failures, on December 2, 1971, Union of Soviet Socialist Republics (USSR) became the first country to land on the Red Planet, with the mission Mars 3. In 1976, the United States of America (USA) landed the missions Viking 1 and 2. For the following 45 years, the Americans were the only ones reaching

Mars' surface with multiple rovers, such as Mars Pathfinder (1997), Spirit and Opportunity (2004), Phoenix (2008), Curiosity (2012), InSight (2018), and Perseverance (2021).

These missions present a strong EDL heritage from Viking missions, such as the use of an ablative heat shield and a disk-gap-band parachute, with only slight modifications from the original ones used in 1976. The main differences are related to the type of entry trajectory, which can be lifting or ballistic, depending on whether the vehicle creates lift flying at an angle of attack. In guided entries, a guidance law modulates the bank angle to rotate the lift vector around the velocity vector, to steer the vehicle and increase landing precision. Moreover, terminal landing systems have evolved from propulsive gravity turns to airbags and sky cranes to increase the payload mass and to make the landing mechanism robust to inclined, rocky surfaces, and lateral winds. Mars 2020, after a guided entry, featured the use of Terrain Relative Navigation (TRN) algorithms, to divert in case of medium or large hazards (Nelessen et al., 2019), introducing the possibility of accessing more interesting, but risky, landing sites. The combination of guided entries and more advanced terminal landing guidance systems has increased the precision, reducing the landing dispersion ellipse.

Several failed missions, such as Mars 96 (Russia), Mars Polar Lander (National Aeronautics and Space Administration (NASA)), and Schiaparelli (ESA), proved the difficulties of descending into an uncertain atmosphere and landing on hazardous terrain. Many missions are planned for the next decades, such as the fascinating NASA-ESA Mars Sample Return, which includes a precise landing, before ascending again from Mars' surface, bringing soil samples back to Earth, to study the possibility of present or past form of life on the Red Planet.

Earth

Initial reusability concepts were conceived for horizontal landing on a runway, such as the US-made Space Shuttle, the first reusable orbital spaceplane. It achieved its first orbital flight and landing on April 12 and 14, 1981. It has become one of the most famous, but controversial, icons of the American spaceflight, achieving 133 successful flights and two disastrous failures. USSR developed the Buran, a similar reusable spaceplane, which only flew and landed once.

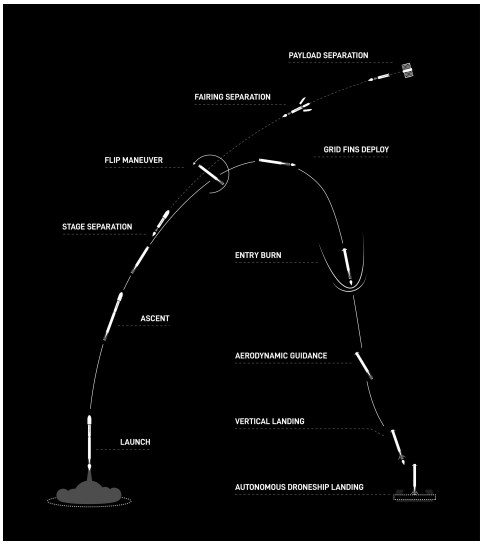
The idea of Vertical Takeoff Vertical Landing (VTVL) launch vehicle has found much interest in the rocket scientists community. Landing rockets, to reuse them again, allows faster and cheaper access to space, enhancing space exploration.

The first prototype that came to light was the **McDonnell Douglas DC-X**. On August 18, 1993, it executed a 100 m hop, achieving the first vertical Earth landing. Unfortunately, in 1996, NASA abandoned funding the project after a single failed test.

In 2006 Blue Origin started the development of VTVL vehicles, which ultimately led to the creation of **New Shepard**, a reusable suborbital booster that first flew on November 23, 2015. It flies above the Karman Line, before landing on a pad a few km far from the launch site. It is also human-rated, carrying a capsule that is released at around the apogee. So far it has completed 24 landings. Concurrently, NASA funded studies for landing guidance algorithms, carried out on the Masten Xombie vehicle and the Morpheus Lander (Scharf et al., 2017).

From 2012 onward, SpaceX started developing a test vehicle to create powered descent and landing technologies. This demonstrator, called Grasshopper, reached a peak altitude of 744 m during test flights. Eventually, this technology enabled **Falcon 9** first stage to achieve the first successful vertical landing on December 21, 2015, becoming the world's first orbital class reusable rocket. So far, more than 350 successful landings of Falcon 9 first stages have been performed, with about 300 reflights. This rocket has an impressive record of reliability, allowing up to 23 reuses of a single booster, so far. Reused stages have also been used to fly humans to the International Space Station (ISS). The Falcon 9 first stage, after the ascent phase and stage separation, performs a precise landing on an autonomous drone ship or at the original launchpad (Fig. 2.1(b)). It reignites its engines twice: for the entry burn and for the

(a) Past, current and future landing vehicles: Space Shuttle (Tony Landis, NASA, 2007), Falcon 9 (SpaceX, 2017), CALLISTO (DLR), Starship (SpaceX, 2023).



(b) Falcon 9 mission trajectory (SpaceX, b).

Figure 2.1: Example of Earth landing vehicles and rocket landing mission.

landing burn at a couple of km of altitude. It is equipped with hypersonic titanium grid fins, which control the aerodynamic unpowered descent phase to enable precise landing.

SpaceX is also developing **Starship**, a heavy-lift two-stage fully reusable rocket, with both stages capable of vertical landing. Test flights in 2023 and 2024 made it the world's most powerful rocket and demonstrated its ability to (partially) successfully land both stages. Similarly, Blue Origin is developing New Glenn, a reusable heavy-lift rocket.

Outside of the US, interest for RLVs grew both in Europe and Asia. Already in 1999, the Japanese Aerospace Exploration Agency (JAXA) started the Reusable Vehicle Testing campaign, based on a model of the DC-XA (JAXA, 2007). In Europe, Deutsches Zentrum für Luft und Raumfahrt (DLR) and Centre National d'Etudes Spatiales (CNES), developed small demonstrators to test GNC algorithms for autonomous vertical landing, called EAGLE (Dumke et al., 2020) and FROG (Ferlin, M., 2022), respectively. More recently, JAXA, DLR, and CNES have joined forces to develop **CALLISTO**, a reusable first stage, that will be launched from the Guiana Space Center (Dumont et al., 2021). The goal is to demonstrate the ascent flight and the aerodynamic and powered landing of a 40-kN class booster. (Sagliano et al., 2019, 2023). Concurrently, ESA funded the **Themis** program, to develop a reusable booster using the 1000-kN Prometheus engine, to create and validate the technologies for a future reusable Ariane rocket. The EU funded the investigation of launchers' reusability for RETALT 1 and 2: a first-stage booster and a single stage to orbit fully reusable vehicle (Marwege et al., 2022).

To summarize, the rocket industry is shifting towards the paradigm of reusability, which is the key for future launchers development, allowing for cheaper, faster, and more flexible missions. Some past, current and future vehicles are illustrated in Fig. 2.1(a).

2.2. Precision atmospheric landing challenges

Successfully landing a rocket is a complex challenge that requires coordination across multiple phases. Understanding the key variables is essential for addressing precision landing issues.

When a vehicle returns from orbit, it initially dissipates a large amount of energy through aerodynamic drag, inducing extreme thermo-mechanical loads, necessitating ablative heat shields (Launius and Jenkins, 2012). Such challenging conditions are typical for capsules

like Apollo and Dragon, but much less for suborbital boosters like Falcon 9. In the terminal landing phase, at lower velocities, the focus is shifted towards different aspects, such as:

- Challenging environment
- Small margins
- Navigation uncertainties
- Vehicle modeling inaccuracy
- Control actuation
- Initial uncertainty distribution
- Automated G&C

Bodies with atmospheres present **environmental challenges** due to their variable properties. On Mars, the thin atmosphere has minimal impact on aerodynamic forces, while on Earth, they are significant. However, the atmosphere is not perfectly known, leading to density uncertainties that can affect flight success. Turbulence and gusts add unpredictability, with Earth's winds at 5 km reaching 40 km/h (McDonough, 1970). The vehicle's low velocity makes wind influence considerable, inducing parasitic moments (Blackmore, 2016). Given all these uncertainties, it is impossible to exactly replicate online a pre-generated nominal trajectory, but they have to be compensated in real-time, with a proper GNC algorithm.

In space missions, propellant mass is minimized to maximize payload, leaving very **small margins**. Hence, the first landing attempt must be a success, because there is no fuel for a second chance. Hovering is also avoided due to its large propellant consumption and the high inefficiency of low throttling levels of rocket engines (Blackmore, 2016). The GNC system must ensure an exact engine cutoff at touchdown because an early shutdown would make the vehicle free fall, while a late one would induce a too large residual landing velocity.

The **navigation** is crucial for any mission, combining multiple sensors to estimate the states of the spacecraft, needed to compute its trajectory. Unlike other planetary bodies, on Earth, the navigation precision is strongly improved by the use of Global Navigation Satellite System (GNSS) receivers (Kruger et al., 2021). For instance, in landing missions, such as CALLISTO (Schwarz et al., 2019), a combination of Inertial Measurement Units (IMUs), precise GNSS receivers and radar altimeters is used to enhance the estimation. However, the estimated state is not the real one, and the GNC system should be able to cope with it.

Between the nominal and the real-world scenario there is a certain gap, due to the assumptions and **accuracy in the modeling** of the vehicle and environment. For example, simplified models are typically used for aerodynamics, with errors on extrapolated coefficients even above 15% (Sagliano et al., 2022). This leads to slightly different acceleration sensed by the real vehicle with respect to simulations. All these aspects have to be considered during mission design, and the robustness of the GNC system to these assumptions has to be assessed. For example, Monte Carlo campaigns can be used to check the mission success a posteriori.

Another typical challenge in space missions is to ensure that the GNC system performs adequately, even when the actual controls executed by the actuators are slightly different from the commanded ones, due to **hardware imperfection** or unmodeled effects.

The landing phase is the terminal part of the flight. However, since during previous phases the uncertainties accumulate, the initial states of the landing stage can vary inside a hypercube bounded by some a priori expected values. Accordingly, the GNC system should be able to land the vehicle from any of the possible **uncertain initial states**.

A rocket must be able to adjust its trajectory using a real-time **automated GNC** algorithm, periodically correcting for all the uncertainties and disturbances. The computational time must be small enough to run on the onboard computer, updating the trajectory with a

relatively high frequency. For example, guidance systems typically run at 10 Hz (Simplício et al., 2020), implying that the algorithm should execute in a few milliseconds.

From the previous points, it stands out that the GNC system is one of the most critical and stressed components, requiring robust design to handle various uncertainties and initial conditions within a limited time frame. This work focuses on Guidance and Control, which generates optimal trajectories and commands control actions. Since the G&C system operates in a closed-loop with navigation, any errors in navigation estimates can propagate through the system, potentially leading to mission failure. Furthermore, the G&C system must be robust to uncertainties in both the vehicle and environment to prevent error accumulation (Steinfeldt et al., 2010). The mission's success depends on meeting constraints on terminal state variables such as position, velocity, attitude, and angular rates.

The 6-DOF formulation is essential for accurately capturing the complex dynamics of the landing phase. The combination of high thrust, frequent maneuvers, and low velocity cause coupling between translational and rotational motion (Simplício et al., 2020). This coupling necessitates a 6-DOF approach to ensure realistic interaction of lateral forces generated by attitude control moments, and vice versa. Additionally, attitude and angular rates constraints require full 6-DOF modeling. The 3-DOF approach, which assumes perfect attitude control, overestimates system performance and ignores inertial effects. Hence, a 6-DOF framework simulates a more accurate and realistic scenario, correctly modeling large, rapid attitude maneuvers where interactions between guidance and control are considerable.

2.3. Precision Landing Guidance methods

The GNC system shall be able to cope with dispersed initial conditions and uncertainties to drive the vehicle to the landing target, in an optimal way. Concurrently, it must satisfy path and terminal constraints, in a robust way (Song et al., 2020).

Guidance algorithms for pinpoint landing have become a very important research focus in the last decades, with innovative solutions driven by mathematical development and computational advances. Two types of guidance algorithms can be identified:

- **Open-loop guidance**, also known as **implicit guidance** (Simplício et al., 2018), relies on a pre-computed reference trajectory and controls profile generated beforehand, based on mission analysis considerations and stored in lookup tables. Generating these solutions is slow, hence, they are created offline due to restricted onboard computational capability.
- **Closed-loop guidance**, also known as **explicit guidance**, refers to the methods where reference trajectory and thrust profile are computed in real-time, based on navigation onboard measurements, to correct the trajectory in a closed-loop way.

2.3.1. Analytical Guidance methods

The first guidance algorithm, developed for the Apollo Lunar Module, is an open loop polynomial solution to a boundary value problem. Given the terminal states, the thrust acceleration is computed, based on a quadratic function of time (Klumpp, 1974). It has many simplifications to derive an analytical expression, such as constant gravity and fixed end-of-mission time. This algorithm is unable to account for constraints and does not provide an optimal solution, as it does not minimize any cost function. It was adopted due to its simplicity and limited computational power. Similarly, the entry guidance of Mars Science Laboratory (MSL) uses an analytical formulation to correct for down and crossrange errors using bank reversals, based on the Apollo Command Module guidance (Mendeck and Craig, 2011).

In parallel to real-world missions, a significant advancement in the field was the use of Optimal Control strategies to minimize energy, fuel consumption, or time of flight, aiming for more efficient solutions. One early solution was the analytic, unconstrained energy-optimal one

identified by D'Souza (1997), where the weighted sum of final time and integrated quadratic acceleration is minimized. However, it involves only translational motion, path constraints cannot be enforced, and simplifications, such as constant gravity, are introduced.

Similarly, another analytical guidance law, without imposing any path constraints, is the Constrained Terminal Velocity Guidance (CTVG), where a closed-form solution is obtained, by assuming a constant gravity field (Guo et al., 2012). This solution is a function of time to go and of Zero-Error-Miss (ZEM) and Zero-Error-Velocity (ZEV), which represent the position and velocity error at the end-of-mission, if no corrective maneuvers are made after time t.

2.3.2. Computational and AI-based Guidance methods

The challenge of implementing advanced guidance methods has historically been the computational burden of solving optimal problems with state and control constraints in real time. However, recent mathematical developments combined with advances in computational power have led to the rise of *Computational Guidance and Control* (Tsiotras and Mesbah, 2017). This field focuses on developing methodologies for on-board, closed-loop solutions. These advances enabled the successful landings of reusable rockets like Falcon 9 and New Shepard. The increasing use of AI and ML has further driven studies on using neural networks to develop robust G&C policies. In the following paragraphs, recent studies and advances in the field of computational and AI-based guidance are presented.

Indirect and Direct methods

Optimal trajectories are typically generated by solving the Optimal Control Problem (OCP), which involves minimizing a cost function, such as fuel consumption, time, energy, or control effort while respecting the dynamics equations of motion and a set of constraints on states, controls, and boundary conditions. These problems are often complex due to the differential equations governing the dynamics and the numerous equality and inequality constraints involved. As a result, except for simple cases, rocket landing OCPs generally require numerical methods for their solutions. These methods fall into two main categories (Rao, 2010):

- **Indirect methods**: These methods are applications of *calculus of variation* and *Pontryagin's Minimum Principle* to determine the set of differential equations that represent the necessary optimality conditions of the OCP (Von Stryk and Bulirsch, 1992). These conditions are usually derived using the Hamiltonian ($\mathcal{H}$), which is the sum of the Lagrangian ($\mathcal{L}$) of the system, the costates ($\boldsymbol{\lambda}$) multiplied by the dynamics ($\mathbf{f}$) and the product of the path constraints ($\mathbf{g}$) and their Lagrange multipliers ($\boldsymbol{\mu}$) :

$$\mathcal{H}(\mathbf{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}, \mathbf{u}, t) = \mathcal{L} + \boldsymbol{\lambda}^T \mathbf{f} - \boldsymbol{\mu}^T \mathbf{g} \tag{2.1}$$

  where $\mathbf{x}$, $\mathbf{u}$, t are the states, control variables and time. Finally, the necessary conditions of optimality on initial and final costates are called *transverality conditions*. Indirect shootings or indirect collocation methods belong to this group. Hence, indirect methods convert the OCP to a Two Point Boundary Value Problem (TPBVP), based on the initial and final conditions, and subject to a set of path constraints. While the necessary optimality conditions guarantee a highly accurate solution, their derivation can be complex, and, above all, the convergence strongly depends on the selection of the initial guess.
- **Direct methods**: These methods consist in the **transcription** of an infinite-dimensional OCP to a finite dimensional Non Linear Programming (NLP) problem (Von Stryk and Bulirsch, 1992). They minimize a scalar function, dependent on a finite number of parameters while respecting a set of nonlinear constraints. The states and controls are discretized over time to turn the problem into a parametric optimization problem. The advantage is to avoid the complex derivation of necessary conditions (Betts, 2010). They

present a straightforward implementation and good convergence, but the computational time scales with the number of discretization variables, making them unable to meet the computational requirements of real-time trajectory updates. Some of the direct methods are the direct collocation methods or pseudospectral methods. The latter ones provide spectral (quasi-exponential) convergence of the NLP solution to the OCP solution when the number of nodes employed is increased (Sagliano, 2018).

- **Hybrid methods**: Their objective is to maximize the strength of both methods, such as using a direct method, less prone to the initial guess sensitivity, to feed an indirect method, and get a very accurate solution (Bulirsch et al. (1993); Spada et al. (2023)).

Implicit Computational Guidance methods

Implicit guidance methods encompass both indirect and direct formulations. Indirect methods are harder to converge but lead to an optimal solution, while direct ones offer a discretized version of the continuous-time problem, but with a better convergence.

Among the direct methods SPARTAN is a pseudospectral NLP tool developed at DLR that transcribe multi-phase OCP to efficient finite-dimensional representation using pseudospectral methods (Sagliano, 2014; Huneker et al., 2015; Sagliano, 2016; Sagliano et al., 2021b).

GPOPS, GPOPS-II, and CGPOPS are direct algorithms developed at the University of Florida, presenting increasing accuracy and complexity. They include a Legendre-Gauss-Radau quadrature collocation method and hp-adaptive mesh refinement methods to transcribe the continuous OCP to large sparse NLP Patterson and Rao (2014).

ICLOCS2 is an optimal control software for direct methods, developed within the Imperial College of London, equipped with state-of-the-art transcription methods to transform the OCP into large NLP with sparsity patterns, solved using direct collocation methods (Nie et al., 2018).

The resulting NLP can be solved with an off-the-shelf solver, such as IPOPT or SNOPT.

Explicit Computational Guidance methods

Explicit guidance methods optimize trajectories online, in a closed-loop manner, given the navigation states. The onboard guidance algorithm should quickly converge to an accurate solution, within strict computational limits, as it recalculates the solution every tens of milliseconds.

Inspired by the indirect guidance system for launch vehicle ascent (Lu et al., 2012), a propellant-optimal powered descent guidance has been developed by Lu (2018). It uses an indirect method to compute the fuel optimal trajectory for a Mars landing, finding out that a bang-bang thrust profile with no more than two switches between upper and lower boundaries is optimal. This method provides the optimal burn time, but it does not enforce inequality constraints. Furthermore, it does not theoretically guarantee convergence, and some assumptions are made, such as the constant gravity field and the absence of aerodynamics.

The advent of Convex Optimization marked a paradigm shift, introducing a class of methods that enables the computation of real-time optimal solutions.

Convex optimization problems minimize a convex objective function, handling convex constraints. While no analytical solution exists, efficient numerical methods such as the Interior-Point methods are used to solve them (Boyd and Vandenberghe, 2004). Unlike NLPs, where iterations grow indefinitely, convex problems have a deterministic bound on computational power (Malyuta et al., 2022). Additionally, they are less dependent on the initial guess, as convexity ensures the convergence of all solutions to the minimum of the convex set. Unfortunately, many aerospace problems, including powered landing, are non-convex and cannot be directly solved with convex optimization. Nevertheless, different techniques have been developed to convert some of the non-convex states and constraints into convex formulations.

The first one introduced is the so-called **LossLess convexification (LCvx)** that relaxes certain non-convex constraints into a convex form, ensuring the relaxed problem's optimal solution matches the original problem, using Pontryagin's Minimum Principle (Açıkmeşe and

Blackmore, 2011). It was first introduced by Acikmese and Ploen (2007) and Blackmore et al. (2012), applying it to the 3-DOF fuel optimal pinpoint powered landing on Mars. The term lossless refers to the fact that no part of the feasible space of the original problem is removed in the convexification process (Blackmore et al., 2012). However, LCvx applies only to some specific problems since not all the sources of non-convexity can be convexified.

This algorithm is at the core of the Guidance for Fuel-Optimal Large Diverts (G-FOLD) software, which was the first guidance algorithm based on convex optimization to be used in test flights on the VTVL Xombie vehicle (Acikmese et al.). It computed real-time fuel-optimal trajectory that satisfies descent constraints and enforces terminal location (Scharf et al., 2017).

Since only a narrow range of problems can be solved with the LCvx, a solution is to apply Sequential Convexification (SCvx). It iteratively solves convex optimization subproblems, starting from a possibly infeasible initial guess and using local linearization around a reference trajectory to remove the non-convexities, until the solution is satisfactory.

The SCvx method was initially applied to a 3-DOF formulation (Szmuk et al., 2016). Then, a 6-DOF solution was formulated to account for attitude constraints and interactions between translational and rotational motion (Szmuk et al., 2017). On top of this, Szmuk and Acikmese (2018) introduced free final time variable. Ultimately, Szmuk et al. (2020) included also atmospheric drag, angular rates, and state-triggered constraints. While the complexity of the problem is increased, the solution accuracy and runtime constraints are still met. The algorithm was also tested onboard a New Shepard flight (Bieniawski et al., 2022).

Although yet undisclosed by SpaceX, convex optimization methods, either lossless or sequential, are used onboard Falcon 9 for the powered landing guidance (Malyuta et al., 2022).

Combining convex optimization and pseudospectral optimal control, Sagliano (2018) has proposed a novel Sequential Pseudospectral Convex Programming (SPCP) method. It offers accurate solutions through pseudospectral operators, and real-time capabilities, thanks to the convex optimization convergence properties. Sagliano et al. (2024a) presents five formulations of the powered landing problem, with increasing levels of complexity, from only the thrust vector as control input to an aero-propulsive landing, closer to the CALLISTO operational scenario.

All the mentioned works consider only the thrust vector as guidance output. Nevertheless, for descent and landing within Earth's atmosphere, aerodynamics is one of the predominant forces, and aerodynamic surfaces, such as fins, are typically used to control the vehicle. Since the accuracy of the powered landing phase depends on the dispersions of the previous aerodynamic descent, it can be formulated as a multi-phase problem.

Sagliano et al. (2021a) separated the EDL phases by using SPCP, based on the explicit separation of convex and non-convex terms in the computation of the solution. The aerodynamic descent is controlled only by the aerodynamic fins, while during the powered landing the TVC manages the translational motion, with the fins trimming the vehicle. In 6-DOF landing (Sagliano et al., 2024c) and entry scenarios (Sagliano et al., 2024b), SPCP-techniques are combined with augmented convex-concave decomposition to deal with nonlinear equality constraints and discrete decision-making without using dedicated solvers.

In Lee and Lee (2022), SCvx and direct optimization are used to successfully tackle a multi-phase optimization problem, considering the guidance of aerodynamic descent and powered landing simultaneously. The problem is formulated as 6-DOF so that aerodynamic fins and thrust vector can alter both the translational and rotational motion of the vehicle.

In Simplício et al. (2020) and De Oliveira and Lavagna (2023), only the thrust vector is considered in the guidance strategy, using CTVG and SCvx, respectively. Afterward, in the control loop, the control authority is allocated between TVC and fins, first considering a single effector and, using a second one only if some additional control moment is needed.

AI-driven Guidance methods

With the recent growing interest in AI and ML for space applications, their use in powered landing guidance is being explored. These methods are attractive because the NNs can learn policies able to generalize across multiple tasks. Compared to the lengthy implicit methods, NNs enable real-time guidance with rapid computations, taking just a few milliseconds for matrix operations. They are also faster than convex optimization methods because NNs do not solve an optimization problem online, but they represent a function approximating the mapping between states and controls. Compared to convex optimization where simplifications are introduced to reduce the computational time to meet real-time requirements, RL can use very accurate and complex models to replicate reality. The computational burden due to more sophisticated models is only in the training, not affecting the real-time performance. Several surveys have explored recent advances in the field (Tipaldi et al., 2022; Shirobokov et al., 2021; Silvestrini and Lavagna, 2022; Izzo et al., 2019).

In the first place, **Supervised Machine Learning**, in the form of imitation learning, has been applied by Sánchez-Sánchez and Izzo (2018), where NNs are successfully trained to represent the solution to the Hamilton-Jacobi-Bellmann equation for different pinpoint landing scenarios. Therefore, the ESA Advanced Concept Team coined the term "G&CNETs", referring to the application of ML techniques in Guidance and Control. In imitation learning, the network learns to replicate a set of optimal control trajectories generated offline.

Imitation learning is also used to learn an approximate states-controls mapping policy to warm start optimization methods, increasing their convergence speed. Cheng et al. (2020) use a fully connected network to approximate the costates to supply a good initial guess, achieving a high success rate of the indirect shooting method, applied to asteroid landings.

SL efficiently classifies data but it is inherently limited by its training set, preventing the learned policy from generalizing to unforeseen scenarios. Since it learns to imitate pre-generated trajectories, the guidance policy is tied to the accuracy of the underlying dynamic models. This dependency can undermine the policy's effectiveness in real-world situations if any incorrect assumptions, uncertainties, or unmodeled dynamics are present in the training. Finally, offline generation of trajectories with Optimal Control solvers is very time-consuming.

To enhance the robustness and adaptability of the guidance system, **Reinforcement Learning** is employed due to its ability to generalize. Instead of learning from data, the guidance strategy is learned from experiences, which consist of simulated interactions between an agent and its environment, receiving rewards or penalties based on the goodness of its action. By generating new data while interacting, the exploration of the state-action space is enabled.

One of the first landing applications is Gaudet and Furfaro (2014), where Deep Reinforcement Learning (DRL) is employed, in combination with GPOPS trajectories pre-training, to find the landing guidance strategy in a 3-DOF Mars environment. Despite the accuracy of the solution being decent, this work was published in 2014, years before some of the most revolutionary ML algorithms were discovered. They employed a custom stochastic search algorithm, using only a delayed reward based on terminal position, velocity, and fuel consumption. Later, the same authors applied the state-of-the-art Proximity Policy Optimization (PPO) algorithm and a new reward function, giving bonuses at each time step as the vehicle approached terminal conditions, to a 6-DOF Mars landing (Gaudet et al., 2020b). The solution was strongly ameliorated, showing submeter and cm/s precision levels on terminal position and velocity, with only 10-15% more fuel consumption than GPOPS trajectories, due to a longer flight time caused by a suboptimal reward. This guidance algorithm is capable of handling a larger range of initial conditions and uncertainties, being computationally very fast.

Meta-RL, combined with RNNs like LSTM, is a method where the agent leverages a memory of past information, to make optimal decisions. The agent is trained in uncertain

environments, making its policy robust to several perturbations so that it can adapt to new scenarios, by retaining past knowledge about temporal variations of the observations. This method has been applied to a multitude of space applications, showing large improvements in unforeseen situations and uncertain environments.

One of the most notable works using RNNs is Gaudet et al. (2020a), where two guidance laws were developed for 3-DOF Mars and asteroid landing. In the former scenario, the agent is trained with PPO including random engine failures, large mass variations, and navigation errors, while in the latter it is trained with unknown dynamics. In both cases, the guidance laws showed extremely good performances, consistently better than standard fully connected networks and conventional energy-optimal closed-loop guidance. In the same work, integrated guidance and navigation was successfully developed, using only Doppler radar and LIDAR altimeter readings, as observation.

Another example of integrated guidance and navigation is the meta-RL based autonomous lunar landing (Scorsoglio et al., 2022). They developed a guidance strategy that maps from lander images to thrust outputs, using convolutional layers. Similarly, Federici et al. (2022) applied image-based meta-RL to guide an impactor in a binary asteroid problem, successfully handling complex dynamics and large uncertainties, with a 98% success rate in Monte Carlo campaigns. In this case, convolutional layers are combined with LSTM, trained with PPO.

Federici and Zavoli (2024) developed a robust closed-loop guidance algorithm for low-thrust Earth-Mars transfers, using PPO and LSTM-based meta-RL. This approach, embedding dynamics, navigation, and control uncertainties in the training, improved performance, increasing terminal accuracy by 20% compared to their previous work on the same topic, but without recurrent networks (Zavoli and Federici, 2021). They also introduced an "ϵ-constraint" technique to gradually enforce terminal constraints, enhancing initial exploration during training.

Fereoli et al. (2024) applied meta-RL to proximity operations in cislunar space between Lunar Gateway and Orion spacecraft, including approach cone constraints. This LSTM PPO-based approach demonstrated better performance than a GPOPS trajectory.

Meta-RL has also been used in defense applications, such as 3-DOF hypersonic glider approach guidance (Gaudet et al., 2022), where it outperformed an optimal GPOPS trajectory tracked by an LQR controller in adapting to aerodynamic uncertainties and actuator failures.

Finally, the last development in the field of ML is the use of **Transformer** neural networks to capture even more distant dependencies in a sequence. They are based on the concept of attention and they are largely employed in Large Language Models and Generative AI.

Using SL, Briden et al. (2024) employed them to learn the set of tight constraints of the 3-DOF fuel-optimal powered descent guidance problem. This set is fed as an initial guess to the direct optimization problem. The offline learning is done using pre-generated optimal trajectories, and the set of tight constraints is returned given only the initial conditions of each simulation. Compared to LCvx, the computational time for a solution is reduced by 78%.

Combining RL and Transformers for the first time in space guidance, Federici and Furfaro (2024) showed that in 3-DOF, atmosphere-free, landing problem the Transformer-based policy performs better compared to a standard fully connected NN, demonstrating adaptability to new scenarios. The control output of the guidance law is the thrust vector to drive the vehicle to a fuel-optimal soft landing. The results of Monte Carlo simulations show improvements by about 50% and 20% in mean terminal position and velocity errors, respectively.

These two papers have shown that Transformers, even if still immature in RL, are very promising for creating robust guidance systems able to adapt to many scenarios.

All the aforementioned works developed guidance algorithms for landing on planetary bodies without atmosphere (or with a thin negligible one), avoiding the challenge of the highly non-linear aerodynamic accelerations. Applying RL techniques to RLV landings on Earth

remains largely unexplored, except for a few papers that use simplified models.

Chen and Ma (2019) trained an agent for a 3-DOF Earth landing with constant C_D using PPO achieving sub-meter accuracy in terminal position and velocity. Only the nominal case is analyzed, suggesting that performance would likely degrade in uncertain scenarios.

To enhance fuel efficiency, Su et al. (2022) introduced multi-stage reward functions in a 3-DOF Earth reusable rocket landing scenario that includes drag, combining a fully connected NN based on PPO with pre-training through imitation learning. Xue et al. (2023) developed a 6-DOF guidance system for Earth landings using RNNs, pre-trained with Behavioral Cloning (BC) using an expert dataset of optimal trajectories. The results show that combining supervised learning with DRL improves the accuracy of terminal position and velocity.

Rosa et al. (2023) is the only rocket landing guidance work with a higher modeling complexity. A 6-DOF high fidelity simulator is used, including TVC, aerodynamics, and winds. Given the increased complexity, the formulation of the reward function is also more articulated. However, they use Deep Deterministic Policy Gradient (DDPG) and no RNN to learn the policy. They show 97% of success rate in a Monte Carlo campaign including limited dispersed initial conditions on position, velocity, and mass only, and allowing a very large terminal vertical velocity (10 m/s).

Most of these works comprise only the thrust vector components as control means, and only one of them uses Transformer-based RL. Furthermore, a level of problem complexity, simultaneously including large uncertainties, fins, winds, aerodynamics, control rates constraints, and variable mass properties is never addressed using any RL method. This leaves an opportunity to close the gap for Earth landing in realistic scenarios using innovative techniques.

2.4. Reference Vehicle

Since the goal of this work is to study the landing of a rocket on Earth, the vehicle adopted represents the first stage of RLV. Although real-world current and future missions, such as Falcon 9 and CALLISTO, vary in size, they all use a throttleable gimballed engine and are equipped with aerodynamic fins (Sagliano et al., 2021a, 2024a). Most of the works on this kind of problem, such as Simplício et al. (2020), De Oliveira and Lavagna (2023), and Lee and Lee (2022), employ reference vehicles with very similar parameters. The vehicle from Lee and Lee (2022) is taken as the reference one because the scenario described in their work is chosen as the benchmark. It is the only publication using TVC and fin deflections as control means in a feedback guidance policy. Since the RL framework employed in this thesis optimizes a feedback policy with TVC and fins as controls, the aforementioned paper is considered the most similar option to compare with. Hence, most of the vehicle parameters are taken from Lee and Lee (2022). Since their vehicle is inspired by the one in Simplício et al. (2020), all the missing values in the former are taken from the latter. As a final remark, the reference vehicle described in this paragraph is close to CALLISTO, which is the main focus of the GNC department at DLR Bremen that hosted the author of the thesis.

The reference vehicle has an axis-symmetric shape, a diameter of 1.5 m, and a total length of 15 m. It is equipped with a single throttleable engine, gimballed around two axes thanks to a TVC system. The characteristics of the rocket engine are summarized in Table 2.1. The vehicle is also equipped with two pairs of fins, one to control the motion around the pitch axis and one around the yaw axis. They are useful in alleviating the efforts of the TVC, reducing fuel consumption required to alter translational and rotational motion Lee and Lee (2022). The fins are mounted in a diametrically opposite configuration, on the top of the stage, to improve stability during descent and landing. The reference vehicle parameters that are not dependent on the model used, are summarized in Table 2.1. Figure 2.2 is an illustration of Callisto, a vehicle similar to the one used in this thesis.

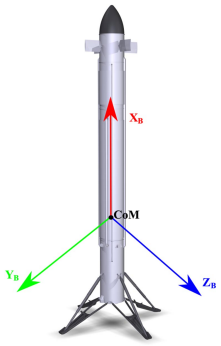


Figure 2.2: Example of the reference vehicle, with body reference frame. Taken from Sagliano et al. (2023).

Table 2.1: Vehicle parameters. Mostly taken from Lee and Lee (2022); Simplício et al. (2020).

Description	Symbol	Value	Unit
Initial wet mass	m_0	4000	kg
Dry mass	m_{dry}	2750	kg
Minimum thrust	T_{min}	40	kN
Maximum thrust	T_{max}	100	kN
Specific impulse	I_{sp}	282	s
Maximum TVC deflection angle	$\epsilon_{T_{max}}, \psi_{T_{max}}$	10	deg
Maximum fin deflection angle	$\beta_{fin,i_{max}}$	25	deg
Maximum TVC deflection rate	$\dot{\epsilon}_{T_{max}}, \dot{\psi}_{T_{max}}$	15	deg/s
Maximum fin deflection rate	$\dot{\beta}_{fin,i_{max}}$	200	deg/s
Fin area	S_{fin}	0.54	m^2
Longitudinal fin position wrt bottom stage	X_{fins}	11.1	m
Engine gimbal position wrt bottom stage	$\mathbf{x}_T$	(0.3, 0, 0)	m

2.5. Mission Scenario

The mission scenario in this thesis is the terminal phase of a RLV re-entering Earth's atmosphere, focusing on the powered landing phase after aerodynamic descent. This phase is the final stage in both Down-Range and Return-To-Launch-Site landings, and it is largely independent of the mission profile. In Down-Range missions, the rocket ascends, the first stage separates, and then it undergoes exo-atmospheric ballistic flight before the aerodynamic descent and powered landing on a downrange platform. In the Return-To-Launch-Site scenario, an additional boost-back maneuver redirects the stage back to the launch site before descent and landing. The ascent is typically flown open-loop with predetermined attitude angles interpolated with respect to altitude. In this phase, dispersions and uncertainties increase, while in the exo-atmospheric flight, where the only relevant force acting on the vehicle is gravity, they remain approximately constant. Only during atmospheric descent, and especially powered landing, they are reduced and driven to (almost) zero.

Two different scenarios are used, and shown in Table 2.2. The first one, called "Benchmark Scenario", replicates Lee and Lee (2022) and it is used to design and verify the flight simulator and the optimal control framework. The second, defined "Complete Scenario", is a more realistic scenario, to represent real-world conditions. Here, initial conditions are enlarged with respect to the Benchmark Scenario, uncertainties in the dynamics are included and more accurate environment and vehicle models are used. In this case, dispersions on the initial states are also introduced, simulating uncertainties accumulated in previous phases.

The initial altitude is set at 2000 m, while the horizontal initial position is chosen to be

Table 2.2: Initial conditions in UEN landing site reference frame, and uncertainties. Benchmark data is taken from Lee and Lee (2022), while Nominal mainly from Federici and Furfaro (2024); Sagliano et al. (2024c).

Description	**Symbol**	**Unit**	**Nominal "Benchmark Scenario"**	**Nominal "Complete Scenario"**	**Dispersion "Complete Scenario"**
Elevation position	x_0	m	600	2000	[1900,2100]
East position	y_0	m	50	200	[180,220]
North position	z_0	m	-50	-200	[-220,-180]
Vertical velocity	v_{x_0}	$\frac{\text{m}}{\text{s}}$	-60	-100	[-110,-90]
Horizontal velocity (East)	v_{y_0}	$\frac{\text{m}}{\text{s}}$	-10	-25	[-30,-20]
Horizontal velocity (North)	v_{z_0}	$\frac{\text{m}}{\text{s}}$	10	25	[20,30]
Roll	ϕ_0	deg	0	0	[-5,5]
Pitch	θ_0	deg	20	20	[15,25]
Yaw	ψ_0	deg	20	20	[15,25]
Roll rate	$\omega_{x,0}$	$\frac{\text{deg}}{\text{s}}$	0	0	[0,0]
Pitch rate	$\omega_{y,0}$	$\frac{\text{deg}}{\text{s}}$	-10	-5	[-7.5,-2.5]
Yaw rate	$\omega_{z,0}$	$\frac{\text{deg}}{\text{s}}$	-10	-5	[-7.5,-2.5]
Mass	m_0	kg	4000	4000	[3900, 4100]
Density uncertainty	ρ	kg/m^3	-	-	$\mathcal{U}(0,10\%)$
Aerodynamic uncertainty	C_X	-	-	-	$\mathcal{U}(0,15\%)$
Aerodynamic uncertainty	C_Y	-	-	-	$\mathcal{U}(0,15\%)$
Aerodynamic uncertainty	C_Z	-	-	-	$\mathcal{U}(0,15\%)$

10% of the initial altitude, following the approach used in CALLISTO studies (Sagliano et al., 2024a). Many landing videos of Falcon 9 show the start of the propulsive phase at 2 km. The initial vertical velocity is 100 m/s, scaled to be 20 times smaller than the initial altitude, as done in Federici and Furfaro (2024) and Sagliano et al. (2024a). Initial horizontal velocities are chosen by reasonably increasing from the benchmark, while initial pitch and yaw angles are kept the same because they already represent realistic values. Angular rates are slightly reduced, to reflect real flight conditions. The nominal mass value is the same used by Lee and Lee (2022) and Simplício et al. (2020).

Table 2.2 presents the initial dispersions, that are sampled from uniform distributions, centered on the nominal values, at the start of each simulation. Regarding the position states, 10% uncertainty is chosen for the horizontal axes, while only 5% for the initial altitude, since it is known with more precision because it is the variable that triggers the beginning of the propulsive landing. Dispersion on vertical velocity is 10%, similar to what is done by Federici and Furfaro (2024), while it is 20% on horizontal components, due to lower absolute value. Dispersions on rotational states are chosen to be a few degrees (or degrees per second) since these variables are not very large in absolute terms. Mass dispersion is 2.5%, close to the value used by De Oliveira and Lavagna (2023).

The most uncertain dynamic variables are those related to atmospheric and aerodynamic properties. At such low altitudes, gravity is almost constant, while density and aerodynamic coefficients can have large local variations due to changes in weather conditions and imperfect knowledge. The uncertainties presented in Table 2.2 are those used in the RL problem. At the start of each simulation, values are sampled from the distributions. They are kept constant for the entire simulation and applied as multiplicative factors to the nominal value. They are similar to the ones used in Sagliano et al. (2022) and they are sampled from a uniform distribution.

2.6. System and Mission Requirements

To develop the G&C system with quantifiable numbers and specific objectives, some requirements on the mission (**MR-XX**) and on the G&C system (**GR-XX**) are determined.

MR-01 to **MR-05** represent the initial conditions of the mission scenario, defined as the interface between aerodynamic descent and powered landing phases. Given the uncertainties accumulated during previous phases, the initial conditions are variable within a given range. They are derived from similar missions and studies, as explained in Section 2.5.

Requirements **MR-06** to **MR-08** focus on the aero-thermal loads the vehicle must sustain. However, during the landing phase, structural and thermal loads are not critical due to the low velocities and altitudes involved. Simplício et al. (2020) highlight that the ascent phase is far more challenging in terms of aero-thermal loads (dynamic pressure and stagnation point heat flux), with the powered landing phase having only a marginal impact. It is expected that as velocity decreases much more than density increases, aero-thermal effects diminish, making these requirements less crucial.

The dynamic loads ($\bar{q}\alpha$) are the product between dynamic pressure $\bar{q}$ and angle of attack α and they are an indicator for lateral loads and bending moments on the vehicle, giving a limit on its structural integrity. Even if the dynamic pressure is not large in the powered landing phase, the angle of attack can significantly increase due to wind gusts, increasing the loads. Therefore, it can have a relevant impact on the mission. The maximum admissible is a standard value in the order of magnitude of typical launchers.

Axial loads are significant during terminal landing when thrust is high and mass is minimal, leading to considerable axial accelerations. The maximum value is chosen as the limit for Falcon 9 (Blackmore, 2016).

The stagnation point heat flux depends on atmospheric density and vehicle velocity. Assuming a cold-wall model, and a turbulent flow given the engine flame, it is calculated with the Chapman equation (Mooij, 2017):

$$q_c = c\frac{\rho(h)^{0.8}}{R_N^{0.2}}||\mathbf{V}||^3 \tag{2.2}$$

where $c = 1.74 \times 10^{-4}$ on Earth and R_N is the curvature radius of the vehicle. The value of this requirement is driven by the entry and aerodynamic descent phases, where both the vehicle's velocity and atmospheric density are substantial.

- **MR-01**: The vehicle shall be able to successfully land from a range of initial altitudes between 1900 and 2100 m with respect to the landing site.
- **MR-02**: The vehicle shall be able to successfully land from a range of initial wet masses between 3900 and 4100 kg.
- **MR-03**: The vehicle shall be able to successfully land, given a 10% initial dispersion on each axis of the horizontal position, with respect to the landing site.
- **MR-04**: The vehicle shall be able to successfully land from a range of initial downward vertical velocity between -90 m/s and -110 m/s.
- **MR-05**: The vehicle shall be able to successfully land from a range of initial attitude between 15 and 25 degrees on the pitch and yaw axis.
- **MR-06**: The vehicle shall not exceed dynamic bending loads ($\bar{q}\alpha$) of 250 KPa deg.
- **MR-07**: The vehicle shall not exceed an axial acceleration (g-load) of 6 g.
- **MR-08**: The vehicle shall not exceed a heat flux of 200 kW/m^2.

Requirements are also determined for the G&C system, to create the correct algorithm for the powered landing mission goals. It has to operate in closed-loop with the navigation to

output the control actions (**GR-01**), giving frequent updates (**GR-02**). The engine shutoff can happen not above 1 m from the landing site because, otherwise, the vehicle would free-fall, and the legs could not structurally sustain the impact (**GR-03**). Similarly, the requirement on the vertical velocity is driven by the same reason (**GR-05**). It is significantly smaller, but more realistic, compared to the one set by Rosa et al. (2023). Constrained by the width of the landing site, either on land or on a barge, the requirement on the horizontal position error is set (**GR-04**). Both the requirements on the horizontal velocity (**GR-06**), residual attitude angles (**GR-07**), and residual angular rates (**GR-09**) are driven by the same idea of enabling a simultaneous landing of all the legs and avoiding tipping over after touchdown. These values are chosen by taking as reference real-world results from Falcon 9 and from similar research (Federici and Furfaro, 2024; Rosa et al., 2023). The angular rates of the vehicle during the trajectory (**GR-08**) are limited by the effectiveness of the control means: TVC and fins. The remaining requirements (**GR-10** to **GR-15**) are based on expected uncertainties of different elements of the vehicle and environment, mainly based on what is found in the literature for similar problems. For example, uncertainties on atmospheric density and aerodynamic coefficients are the same as those used in Sagliano et al. (2022), while the control errors are based on values applied to similar rocket landing problems.

- **GR-01**: The G&C system shall output thrust magnitude, TVC and fin deflections, given as input estimate position, velocity, attitude, angular rates, and mass states.
- **GR-02**: The G&C system shall execute in less than 50 ms on a standard computer, to enhance the potential of running onboard the vehicle to implement real-time trajectory computations.
- **GR-03**: The G&C system shall be able to command engine shutoff during landing within 1 m of vertical terminal position error in the nominal scenario.
- **GR-04**: The G&C system shall enable landing within 10 m of horizontal position error.
- **GR-05**: The G&C system shall enable landing within 2 m/s of vertical velocity error.
- **GR-06**: The G&C system shall enable landing within 1.5 m/s of horizontal velocity error on each axis.
- **GR-07**: The G&C system shall enable landing within 3 deg of vertical attitude residual with respect to the vertical axis of the landing pad.
- **GR-08**: The G&C system shall generate trajectories that do not exceed rotational rates larger than 15 deg/s during the flight, on each axis.
- **GR-09**: The G&C system shall be able to ensure that the vehicle has less than 3 deg/s residual rotational rate at touchdown, on each axis.
- **GR-10**: The G&C system shall be robust up to $\pm 15\%$ aerodynamic coefficients uncertainty.
- **GR-11**: The G&C system shall be robust up to $\pm 10\%$ atmospheric density uncertainty.
- **GR-12**: The G&C system shall handle wind profiles magnitude up to 15 m/s.
- **GR-13**: The G&C system shall be robust to uncertain wind gust.
- **GR-14**: The G&C system shall be robust to 0.25 deg/s errors in TVC and fins rates.
- **GR-15**: The G&C system shall be robust to 1% bias and 1% error in thrust magnitude.

3

Flight dynamics

Flight dynamics describes the motion and orientation of a vehicle subject to forces and moments in a three-dimensional space. The reference frames are described in Section 3.1 and Appendix B.1, while the transformations in Appendix B.2. State variables are defined in Section 3.2. Section 3.3 defines all the forces and moments present in the Equations Of Motion (EOMs) for the landing problem considered. Finally, the models used for the environment are presented in Section 3.4.

3.1. Reference Frames

A reference frame represents a coordinate system with a given origin and orientation, with respect to which the kinematics and dynamics are described. Multiple reference frames are used in the description of the motion of a vehicle because it is more understandable to formulate some external forces in a specific reference frame compared to others. As categorized by Mooij (1994), they can be divided into two different classes: the planet-centered reference frame and the vehicle-centered ones. The inertial frame and the one where the EOMs are represented are described in this section, while all the other frames needed to describe quantities involved in the problem are presented in Appendix B.1. All the following reference frames are expressed in Cartesian coordinate systems and are considered to be right-handed.

Earth centered inertial reference frame $\mathcal{I}$

The inertial frames are those where Newton's first law is valid. The Earth Centered Inertial (ECI) reference frame is an example of the aforementioned type of frames [1], where the origin coincides with the Earth center of mass, the $X_{\mathcal{I}} - Y_{\mathcal{I}}$ plane corresponds to the terrestrial mean equatorial plane at J2000 and the $Z_{\mathcal{I}}$ axis is defined by the direction of the Earth mean rotation pole at J2000. More specifically, $X_{\mathcal{I}}$ axis points in the direction of the mean Vernal Equinox at J2000 and $Y_{\mathcal{I}}$ completes the right-handed system (Subirana et al., 2011). J2000 is used to refer to the epoch corresponding to 12:00 Terrestrial Time on January 1, 2000.

Landing site Up-East-North rotating reference frame UEN

The Up-East-North landing site-centered reference frame belongs to the category of planetary ones. It has its origin on the landing site, it has the $X_{\mathbf{UEN}}$ axis perpendicular to the local horizon, the $Y_{\mathbf{UEN}}$ axis aligned with the true east direction, and $Z_{\mathbf{UEN}}$ axis aligned with the true north direction.

[1] Formally, it is defined as pseudo-inertial due to Earth motion around the Sun, and thus it is subjected to a certain acceleration but can be thought as inertial over short periods of time

3.2. State Variable Definition

A set of variables defined as states is used to describe the evolution of a dynamic system over time. Variables related to both translational motion and rotational motion are explained in Sections 3.2.1 and 3.2.2, respectively.

3.2.1. Position and Velocity

The position and velocity of a vehicle flying in an atmosphere are commonly described using a Cartesian or spherical representation. Here, only Cartesian ones are described, because it is the representation used throughout this work. Since the flight takes place at a low altitude (below 2 km), flat Earth can be assumed, and it is convenient to refer to position or velocity to a surface relative reference frame, such as the landing site one. Hence, to simulate the motion of the vehicle, the Cartesian representation is used rather than the spherical one. Moreover, the visualization is also done using landing-site relative Cartesian coordinates, which are easier to understand for low-altitude flights, compared to the harder-to-interpret small variations of spherical coordinates. For instance, a 1 km distance on Earth's surface, along a meridian, corresponds to roughly 0.01 degrees in latitude, which is not intuitive to interpret.

Cartesian components

Cartesian components express position and velocity with respect to either an inertial or rotating reference frame. In this thesis, they are both used depending on the scenario.

$$\begin{cases} \text{Position:} & \mathbf{r} = r_x, r_y, r_z \\ \text{Velocity:} & \mathbf{v} = v_x, v_y, v_z \end{cases} \tag{3.1}$$

3.2.2. Attitude and Angular Rates

The attitude of a vehicle is defined as the orientation of a body-fixed reference frame relative to an external one. It can be expressed through various representations, including aerodynamic angles, attitude Euler angles, Modified Rodriguez Parameters (MRP), and quaternions. While aerodynamic and Euler angles are easy to visualize in 3D space, quaternions lack intuitive interpretation but are singularity-free. Although singularities in aerodynamic, Euler angles, and MRP can be managed by switching between multiple sets of angles, quaternions are used to simulate the attitude due to their straightforward implementation, computational efficiency, and widespread use. However, vehicle attitude is visualized using Euler angles, which intuitively represent roll, pitch, and yaw, making the vehicle's behavior easier to interpret. Since many relative orientations, such as TVC and fin deflections with respect to the body frame, are commonly available a sets of Euler angles, these are used directly for frame transformations.

The angular rate of a body is the rotational velocity of the body frame with respect to the landing site **UEN** frame, expressed in body axes components. Therefore, the rotation vector ω is composed by the roll rate ω_x, pitch rate ω_y and yaw rate ω_z.

Classical attitude Euler angles

Classical attitude angles are represented by the following set of Euler angles: roll ϕ, pitch θ and yaw ψ. They define the orientation of the body frame with respect to the landing site **UEN** reference frame. They are used for visualization and interpretation purposes.

Quaternions

Quaternions are a mathematical formulation used to represent the three dimensional relative orientation of two reference frames. In this work, they define the attitude of the body reference frame with respect to the landing site **UEN** reference frame, and they are used to simulate the vehicle's attitude in the equations of motion.

$$\mathbf{q} = (q_1, q_2, q_3, q_4)^T = \begin{pmatrix} \mathbf{q}_{1:3} \\ q_4 \end{pmatrix} \tag{3.2}$$

A quaternion is a 4-dimensional hyper-complex number, with a real component and three imaginary parts. In this work, the fourth element of the quaternion (q_4) is considered the scalar (real) part, using the notation from Markley and Crassidis (2014). In attitude representation, the four parameters of a unit quaternion are not independent of each other, but they are constrained to satisfy the norm of the quaternion equals to one:

$$q_1^2 + q_2^2 + q_3^2 + q_4^2 = 1 \tag{3.3}$$

3.3. Equations of Motion

This section lays out the 6-DOF EOMs used to describe the motion of the vehicle in an atmospheric environment. In Section 3.3.2 the translational motion of the Center of Mass (CoM) of the vehicle is described, due to the forces acting on it, as illustrated in Section 3.3.1. Additionally, in Section 3.3.4, the motion around the CoM is described, due to the moments acting on it, as illustrated in Section 3.3.3.

Due to the small vehicle velocity and mass compared to the speed of light and to the mass of the stars, the special and general relativity are not considered and the EOMs are expressed according to Classical Mechanics. As additional assumptions, the space vehicle is considered a rigid body with variable mass, neglecting effects such as sloshing and bending. The time dependency (t) has been dropped in all the following equations, for easiness of notation.

3.3.1. External Forces

The external forces acting on the rocket during the landing phase are coming from three different sources: gravity, propulsion, and aerodynamics.

Gravitational forces

The gravitational force is the force exerted by the central body on the vehicle. Expressed in Cartesian coordinates in the landing site frame (**UEN**), it is:

$$\mathbf{F}_{\mathbf{G}}^{\mathbf{UEN}} = m\mathbf{g}^{\mathbf{UEN}} = m \begin{pmatrix} -g \\ 0 \\ 0 \end{pmatrix} \tag{3.4}$$

m is the vehicle mass and $\mathbf{g}^{\mathbf{UEN}}$ is the gravitational acceleration vector in **UEN** frame.

Propulsion forces

The propulsion frame $\mathcal{P}$ is defined, where the thrust force vector is expressed as:

$$\mathbf{F}_{\mathbf{T}}^{\mathcal{P}} = \mathbf{T}^{\mathcal{P}} = \begin{pmatrix} T \\ 0 \\ 0 \end{pmatrix} \tag{3.5}$$

The thrust magnitude T is computed as:

$$T = \dot{m} I_{sp} g_0 \tag{3.6}$$

g_0 is the gravitational acceleration constant at sea level (Table 3.1).

To correctly define the propulsion forces and moments in the body-centered reference frame $\mathcal{B}$, the following parameters have to be known:

- T_i, magnitude of the thrust force generated by the engine,
- ϵ_T, ψ_T, thrust force angles, due to TVC actuators, with respect to the body frame.

According to the transformation $\mathbf{C}_{\mathcal{B},\mathcal{P}}$ as outline in Table B.1, the thrust force in the body frame becomes (Simplício et al., 2020):

$$\mathbf{F}_{\mathbf{T}}^{\mathcal{B}} = \begin{pmatrix} T\cos\epsilon_T \cos\psi_T \\ T\cos\epsilon_T \sin\psi_T \\ -T\sin\epsilon_T \end{pmatrix} \tag{3.7}$$

Aerodynamic body forces

The aerodynamic forces are significant forces during the landing problem, due to the dense Earth's atmosphere, strongly affecting the motion of the vehicle. They depend on the vehicle shape, on its orientation with respect to the velocity vector (aerodynamic angles α and β), and on the dynamic pressure. The dynamic pressure is calculated as:

$$\bar{q} = \frac{1}{2}\rho||\mathbf{V}_A||^2 \tag{3.8}$$

The aerodynamic forces can be expressed in the (airspeed-based) aerodynamic frame $\mathcal{AA}$ or in the body frame $\mathcal{B}$ directly. In the first case, C_D, C_S, and C_L are the coefficients that are needed to calculate drag (D), side (S), and lift (L) forces, orientated as the axes of the aerodynamic frame. With the second formulation, the aerodynamic coefficients (C_X, C_Y and C_Z) and forces (X, Y and Z,) are already orientated as the body frame axes. It is remarked that these two formulations are totally equivalent. The only difference is whether the rotation between aerodynamic and body frames is already embedded in the aerodynamic coefficients (when they are provided in $\mathcal{B}$ frame) or shall be performed after the computation of the aerodynamic force (when the coefficients are provided in $\mathcal{AA}$ frame).

In this thesis, the coefficients are stored in lookup tables, already in the body frame. They are provided as axial (C_A) and normal coefficients (C_N), avoiding the rotation from aerodynamic to body frame. The body frame identifying C_A and C_N has the same axes as the body frame ($\mathcal{B}$) of this thesis, but it has opposite positive directions. Hence, the only operation is to transform C_A and C_N to C_X, C_Y, and C_Z, correcting the direction of axes of the reference frame in which the coefficients are provided. Therefore, the aerodynamic forces, in the body reference frame are expressed as:

$$\mathbf{F}^{\mathcal{B}}_{aero,body} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \bar{q}S_{ref} \begin{pmatrix} C_X \\ C_Y \\ C_Z \end{pmatrix} \tag{3.9}$$

where X, Y, and Z are the aerodynamic forces, orientated as the axes of the body frame. In case there is no wind, the groundspeed and airspeed are coincident, otherwise, the following steps have to be taken to compute the aerodynamic forces. Groundspeed and wind velocity are given in **UEN** frame, they are subtracted to get the airspeed, which is first rotated in body $\mathcal{B}$ frame and then used to compute the aerodynamic angles (α, β). Using them, from the lookups, the aerodynamic coefficients (C_X, C_Y, C_Z) are extracted, and forces are calculated in body $\mathcal{B}$ frame. Eventually, they are rotated back in **UEN**, to be used in the EOMs.

Aerodynamic fins forces

In addition to the aerodynamic forces generated by the body of the rocket, the forces generated by the fins are also included in the problem and described here.

The force generated by each fin is calculated similarly to what is done in Eq. (3.9) for the aerodynamic body forces. For each fin, they are calculated in the corresponding fin frame:

$$\mathbf{F}^{\mathcal{F}}_{aero,fin,i} = \begin{pmatrix} X \\ Y \\ Z \end{pmatrix}_{fin} = \bar{q}S_{ref,fin} \begin{pmatrix} C_X \\ C_Y \\ C_Z \end{pmatrix}_{fin} \tag{3.10}$$

where $(X, Y, Z)_{fin}$ are the aerodynamic forces, orientated as the fin frame's axes. After being converted to the vehicle body frame, the forces generated by the four fins are summed.

$$\mathbf{F}^{\mathcal{B}}_{aero,fins} = \sum_{i=1}^{4} \mathbf{C}_{\mathcal{B},\mathcal{F}}\mathbf{F}^{\mathcal{F}}_{aero,fin,i} \tag{3.11}$$

The calculation of the aerodynamics for the fins is presented here. Given the reduced fin area with respect to the vehicle body area, the axial force of the fins is neglected, while only the component normal to the surface of the fin is considered (i.e., along the Y-axis of each fin reference frame) Simplício et al. (2019). Neglecting flow separation, the normal contribution has a sinusoidal dependence on the fin local angle of attack (α_{local}). $\bar{C}_{fin}$ is the maximum normal fin force coefficient. Thus, the normal fin force coefficient in the fin reference frame is:

$$C_{fin}(\alpha_{local}) = \bar{C}_{fin} \sin(\alpha_{local}) \tag{3.12}$$

The local angle of attack is computed differently for the two pairs of fin. Since one pair is associated with the control of the pitch plane (fins 1 and 2) and one with the control of the yaw plane (fins 3 and 4), they are respectively related to angle of attack and angle of sideslip:

$$\begin{aligned} \alpha_{local,i} &= \beta_{fin,i} - \alpha \qquad i = 1,2 \\ \alpha_{local,i} &= \beta_{fin,i} - \beta \qquad i = 3,4 \end{aligned} \tag{3.13}$$

The force normal to the fin surface is then calculated, in each fin frame, as:

$$F^{\mathcal{F}}_{Y_{fin,i}} = \bar{q} S_{fin} C_{fin}(\alpha_{local}) \tag{3.14}$$

Then, for each of the four fins, a rotation takes place, to express them in the body frame. The fins controlling the pitch plane give a component of the force along the $X_{\mathcal{B}}$ and $Z_{\mathcal{B}}$ axes:

$$\mathbf{F}^{\mathcal{B}}_{fin,i} = \begin{pmatrix} F_{Y_{fin,i}} \sin(-\beta_{fin,i}) \\ 0 \\ F_{Y_{fin,i}} \cos(-\beta_{fin,i}) \end{pmatrix} = \begin{bmatrix} \cos(-\beta_{fin,i}) & -\sin(-\beta_{fin,i}) & 0 \\ 0 & 0 & -1 \\ \sin(-\beta_{fin,i}) & \cos(-\beta_{fin,i}) & 0 \end{bmatrix} \begin{pmatrix} 0 \\ F^{\mathcal{F},pitch}_{Y_{fin,i}} \\ 0 \end{pmatrix} \qquad i = 1,2 \tag{3.15}$$

The fins controlling the yaw plane give a component along the $X_{\mathcal{B}}$ and $Y_{\mathcal{B}}$ axes:

$$\mathbf{F}^{\mathcal{B}}_{fin,i} = \begin{pmatrix} -F_{Y_{fin,i}} \sin(-\beta_{fin,i}) \\ F_{Y_{fin,i}} \cos(-\beta_{fin,i}) \\ 0 \end{pmatrix} = \begin{bmatrix} \cos(-\beta_{fin,i}) & -\sin(-\beta_{fin,i}) & 0 \\ \sin(-\beta_{fin,i}) & \cos(-\beta_{fin,i}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{pmatrix} 0 \\ F^{\mathcal{F},yaw}_{Y_{fin,i}} \\ 0 \end{pmatrix} \qquad i = 3,4 \tag{3.16}$$

3.3.2. Translational Dynamics

Starting from Newton's laws, the translational EOMs can be expressed in the inertial frame. However, that is not the most advantageous representation. Using a formulation with respect to the landing site brings a set of benefits. For example, it enables a straightforward interpretation of state variables (position and velocity in the first place), easing the analysis and understanding process. Moreover, the quantities at stake are smaller, avoiding possible numerical issues when large values are used, as in Earth-centered reference frames. Therefore, the EOMs are expressed in the **UEN** reference frame. Since it is fixed to the Earth's surface, it is a rotating frame. Hence, to ensure the validity of Newton's laws, which are postulated in the inertial frame, fictitious forces have to be introduced. Thus, the EOMs in **UEN** frame are:

$$\begin{aligned} \dot{\mathbf{r}}^{\mathbf{UEN}} &= \mathbf{v}^{\mathbf{UEN}} \\ \dot{\mathbf{v}}^{\mathbf{UEN}} &= \mathbf{g}^{\mathbf{UEN}} + \frac{1}{m}(\mathbf{F}^{\mathbf{UEN}}_{aero,body} + \mathbf{F}^{\mathbf{UEN}}_{\mathbf{T}} + \mathbf{F}^{\mathbf{UEN}}_{aero,fins}) - 2\boldsymbol{\omega}_{\oplus} \times \mathbf{v}^{\mathbf{UEN}} - \boldsymbol{\omega}_{\oplus} \times (\boldsymbol{\omega}_{\oplus} \times \mathbf{r}^{\mathbf{UEN}}) \end{aligned} \tag{3.17}$$

where $\mathbf{r}^{\mathbf{UEN}}$ and $\mathbf{v}^{\mathbf{UEN}}$ represent the position and velocity of the CoM of the vehicle in the landing site reference frame. $\mathbf{g}^{\mathbf{UEN}}, \mathbf{F}^{\mathbf{UEN}}_{\mathbf{T}}, \mathbf{F}^{\mathbf{UEN}}_{aero,body}, \mathbf{F}^{\mathbf{UEN}}_{aero,fins}$ represent the gravitational acceleration, the propulsive force, and the aerodynamic force of body and the fins, respectively,

expressed in landing site reference frame. The terms $2\boldsymbol{\omega}_\oplus \times \mathbf{v}^{\mathbf{UEN}}$ and $\boldsymbol{\omega}_\oplus \times (\boldsymbol{\omega}_\oplus \times \mathbf{r}^{\mathbf{UEN}})$ represent the apparent accelerations due to the vehicle's motion in the rotating frame: Coriolis and centrifugal acceleration terms, respectively. An assumption is made, regarding the rotation rate $\boldsymbol{\omega}_\oplus$ which formally depends on latitude. Since the lateral displacement between the vehicle and landing site is minimal, the variation in latitude and rotational rate is negligible. At the Equator, Earth rotates at $\boldsymbol{\omega}_\oplus = 7.292115 \times 10^{-5}$ rad/s.

Only for the benchmark problem, Earth's rotation is neglected as the **UEN** reference frame is considered inertial due to low altitude and short simulation time, as done in Lee and Lee (2022). For instance, considering the initial conditions of the Benchmark Scenario (Table 2.2), Coriolis acceleration is in the order of 10^{-3} m/s^2, while the centrifugal one 10^{-6} m/s^2, which are about 5 and 8 orders of magnitude smaller than the gravitational one. Instead, in the "Complete Scenario" Earth's rotation is included, as done in many similar works, such as Sagliano et al. (2021a). Considering its initial conditions, the apparent accelerations are approximately 10^{-2} m/s^2 on the lateral (east) component, 1-2 orders of magnitude smaller than typical aerodynamic accelerations. Considering 15% uncertainty in aerodynamic coefficients, the fictitious lateral acceleration is comparable to that caused by the uncertainty, similarly affecting lateral motion. Although not the most significant term, it is included in the RL training to enhance guidance policy fidelity. Additionally, its inclusion does not impact simulation time as it is computationally light. Finally, if the aerodynamic descent is also simulated, starting at 30 km of altitude with 1000 m/s of vertical velocity, as Falcon 9 does, the fictitious accelerations become more important, rising to about 10^{-1} m/s^2. Thus, modeling them in the EOMs increases the flexibility to simulate multiple scenarios.

3.3.3. External Moments

This section outlines the external moments acting on the vehicle: propulsion moment and aerodynamic moment, generated by the body of the vehicle and its fins. The gravitational moment would be generated by the differences in gravity acting on different points of the vehicle. However, given the relatively small size and the negligible gravity gradient over the vehicle, this is not considered in the problem.

Propulsion moments

Based on the propulsion forces defined in Section 3.3.1, and the position of the engine relative to the CoM in body frame ($\mathbf{x_T} - \mathbf{x}_{CoM}$), the total moments acting on the vehicle in the body frame are given by:

$$\mathbf{M}^{\mathcal{B}}_{\mathbf{T}} = (\mathbf{x_T} - \mathbf{x}_{CoM}) \times \mathbf{C}_{\mathcal{B},\mathcal{P}} \mathbf{T}^{\mathcal{P}} \tag{3.18}$$

The position of the gimbal point $\mathbf{x_T}$ is fixed in time.

Aerodynamic body moments

The aerodynamic moments are generally formulated as the sum of two components: the aerodynamic moment around a fixed reference point ($\mathbf{M}^{\mathcal{B}}_{aero,body,0}$) and the cross product of aerodynamic forces and distance between the CoM and a reference point ($\mathbf{x}_{ref} - \mathbf{x}_{CoM}$):

$$\mathbf{M}^{\mathcal{B}}_{aero,body} = \mathbf{M}^{\mathcal{B}}_{aero,body,0} + (\mathbf{x}_{ref} - \mathbf{x}_{CoM}) \times \mathbf{F}^{\mathcal{B}}_{aero,body} \tag{3.19}$$

$$\mathbf{M}^{\mathcal{B}}_{aero,body,0} = \bar{q} S_{ref} \begin{pmatrix} C_{l,0} b_{ref} \\ C_{m,0} c_{ref} \\ C_{n,0} b_{ref} \end{pmatrix} \tag{3.20}$$

$C_{l,0}$, $C_{m,0}$, and $C_{n,0}$ are the roll, pitch, and yaw damping moment coefficients, while b_{ref} and c_{ref} the aerodynamic reference lengths. Moment damping coefficients are very difficult to estimate for rockets and therefore they are assumed equal to zero in this work. As pointed out

in Sagliano et al. (2023), it is a conservative assumption, since the damping term would make the response slower and more stable.

Hence, it is assumed that the aerodynamic moments generated by the body of the vehicle around its CoM are caused only by the aerodynamic forces applied in the Center of Pressure (CP), calculated as the cross product between the aerodynamic forces and the CoM-CP offset:

$$\mathbf{M}^{\mathcal{B}}_{aero,body} = (\mathbf{x}_{CP} - \mathbf{x}_{CoM}) \times \mathbf{F}^{\mathcal{B}}_{aero,body} \tag{3.21}$$

The center of pressure $\mathbf{x}_{CP}$ varies by about 6 meters, depending on the angle of attack, sideslip and Mach number and it is tabulated together with the aerodynamic coefficients. Its variation is shown in Fig. C.1(c).

Aerodynamic fins moments

Similarly to the aerodynamic body moments, also those for the fins are calculated only using the fins aerodynamic forces and the distance between each of their CP and the vehicle CoM:

$$\mathbf{M}^{\mathcal{B}}_{aero,fins} = \sum_{i=1}^{4} (\mathbf{x}_{finCP,i} - \mathbf{x}_{CoM}) \times \mathbf{F}^{\mathcal{B}}_{aero,fin,i} \tag{3.22}$$

Given the small size, each fin's CP is assumed to be the fixed attachment point on the rocket.

3.3.4. Rotational Dynamics and kinematics

The rotational dynamics of the vehicle body, with respect to the inertial reference frame, is defined by the Euler equations, written as:

$$\dot{\boldsymbol{\omega}}^{\mathcal{B}} = \mathbf{J}^{-1}_{\mathcal{B}} [\mathbf{M}^{\mathcal{B}}_{aero,body} + \mathbf{M}^{\mathcal{B}}_{\mathbf{T}} + \mathbf{M}^{\mathcal{B}}_{aero,fins} - \boldsymbol{\omega}^{\mathcal{B}} \times \mathbf{J}_{\mathcal{B}} \boldsymbol{\omega}^{\mathcal{B}}] \tag{3.23}$$

$\boldsymbol{\omega}^{\mathcal{B}}$ is the rotation rates of the vehicle expressed in the body axes with respect to the **UEN** landing site reference frame. $\mathbf{J}_{\mathcal{B}}$ is the inertia matrix of the vehicle expressed in body frame, while $\mathbf{M}^{\mathcal{B}}_{\mathbf{T}}$, $\mathbf{M}^{\mathcal{B}}_{aero,body}$, $\mathbf{M}^{\mathcal{B}}_{aero,fins}$ are the propulsive moments and aerodynamic moments of body and fins, expressed in body axes, as defined earlier.

The vehicle's body axes attitude is propagated using the quaternion kinematics equation:

$$\dot{\mathbf{q}}^{\mathcal{B}} = \frac{1}{2} \Omega \cdot \boldsymbol{\omega}^{\mathcal{B}} = \frac{1}{2} \begin{bmatrix} q_4 & -q_3 & q_2 \\ q_3 & q_4 & -q_1 \\ -q_2 & q_1 & q_4 \\ -q_1 & -q_2 & -q_3 \end{bmatrix} \begin{pmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{pmatrix} \tag{3.24}$$

3.4. Environment models

This section presents the possible variants for each model, followed by a preliminary trade-off analysis to select the most adequate options. The chosen models for simulations and training are then described in detail. The choices are verified in Section 5.11, where an error analysis is performed a posteriori, to assess if the critical reasoning based on the requirements led to the correct selection. Given the requirements on terminal position and velocity constraints, the errors generated by a lower fidelity method shall be one order of magnitude less than the constraint. For example, position errors shall be at the submeter level (compared to the 10-meter landing radius **GR-04**), while velocity at centimeter per second level (**GR-05**).

3.4.1. Gravity Model

The gravity model used in this work includes only Earth's gravitational force, as third-body perturbations from other Solar System bodies are negligible for low-altitude flights.

Given the small altitude range, close to the surface (**MR-01**), and minimal changes in latitude and longitude (**MR-02**), high-order harmonics are unnecessary, as they would impact

Table 3.1: Earth reference parameters (Petit and Luzum, 2010).

Parameter	Reference value	Unit
$\mu_\oplus$	$3.9860044188 \times 10^{14}$	m^3/s^2
$R_\oplus$	6378136.6	m
$g_0 = \frac{\mu_\oplus}{R_\oplus^2}$	9.7982867	m/s^2

terminal position and velocity in the order of mm and mm/s. Even the largest spherical harmonic acceleration term (i.e., J_2) is almost three orders of magnitude smaller than the constant gravity ($\approx 10^{-2}$ m/s^2 (Wakker, 2015)). Also, the discrepancy between a central body gravity model and a constant one is about 0.01 m/s^2 (about 0.1%). Despite having the same order of magnitude, the error between constant gravity and central body gravity model increases with altitude, while the influence of the J_2 decreases.

Although gravitational errors from a constant gravity model are small compared to other forces in the dynamics, they generate similar accelerations as those from aerodynamic and density uncertainties, potentially propagating to cm and cm/s errors on terminal position and velocity. Therefore, a more advanced model is used. In the "Complete Scenario", a central body gravitational model is used, to account for these errors and assess their influence on the terminal position and velocity. It is preferred to the spherical harmonics term also due to the lower computational effort needed.

Considering only the central body point mass gravitational acceleration, it acts only on the radial axis, and it is a function of the radius from the center of the Earth:

$$\mathbf{g}^{\mathbf{UEN}} = \left(-\frac{\mu_\oplus}{(R_\oplus + h)^2}, 0, 0 \right) \tag{3.25}$$

where $\mu_\oplus$ and $R_\oplus$ are Earth's gravitational parameter and equatorial radius, as reported in Table 3.1. For the "Benchmark Scenario", a constant gravity model is employed.

3.4.2. Atmospheric Model

The simplest atmospheric model is the exponential one, which assumes constant temperature and gravity, to calculate the atmospheric density as an exponential function of the altitude.

$$\rho = \rho_0 e^{-\frac{h}{H_s}} \tag{3.26}$$

ρ_0 is the reference constant density at sea level and $H_s = \frac{RT}{g_0}$ is the scale height.

The U.S. Standard Atmosphere 1976 (US76) (NOOA et al., 1976) is a more realistic model, based on balloons and rocket observations that calculates temperature, pressure, and density only as functions of altitude, assuming a dry, perfect gas mixture. The NASA Earth Global Reference Atmosphere Model (GRAM) accounts for geographical and temporal changes of thermodynamic variables and wind components, also providing statistically perturbed variables useful for Monte Carlo simulations. However, it lacks a Python interface, making it unusable for this work. The most complex model, NRLMSISE00, accounts for geomagnetic and solar effects for higher accuracy but is highly computationally intensive, thus impractical for this work given the need for thousands of simulations to train NNs.

Correct modeling is crucial for the atmosphere, as aerodynamic forces depend on it and strongly affect the terminal accuracy. While not using the NRLMSISE00 model can lead to large terminal errors for long propagations, in short flights of 40 seconds like in this work, the terminal errors (**GR-04**, **GR-05**) are reduced to subcentimeter levels even using simpler models. The exponential model provides a similar profile as US76, but it can introduce local density errors of up to $\pm 20\%$ within 10 km of altitude, affecting aerodynamic force accuracy.

This error can combine with aerodynamics uncertainties (**GR-10**) resulting in force errors far beyond realistic values. The computational time for the two models is similar since both are expressed with two equations. Finally, US76 offers a more precise temperature profile, leading to a more accurate Mach number for aerodynamic coefficient interpolation.

Supported by the use of US76 in many other rocket landing studies, such as De Oliveira and Lavagna (2022), Simplício et al. (2019), and Lee and Lee (2022) it is the chosen model also here, as a trade-off between accuracy and runtime. The primary goal of this manuscript is to develop robust guidance adaptable to various realistic uncertain conditions, rather than to precise conditions measured for a specific location. Since the flight spans only a few kilometers with minimal atmospheric changes, US76 is the ideal atmospheric model for this thesis.

In the US76 model, for altitude below 86 km, the atmosphere is divided into a set of layers, where linear constant temperature rates for each layer (L_k) are assumed (Eq. (3.27)).

$$T = T_0 + L_k(r_x - r_{x_0}) \tag{3.27}$$

where T_0 and r_{x_0} are initial temperature and altitude, while L_k the thermal lapse rate.

The atmosphere below 86 km of altitude is assumed homogeneously mixed, in hydrostatic equilibrium (Eq. (3.29)), and the air is treated as if it were a perfect ideal gas (Eq. (3.28)).

$$p = \rho R T \tag{3.28}$$

$$dp = -g\rho dh \tag{3.29}$$

where p, represents atmospheric pressure and R the gas constant for air (assumed constant at 287 J/kg K). Manipulating the ideal gas and hydrostatic equilibrium equations, an analytical expression for atmospheric density (ρ) with respect to altitude can be found:

$$\rho = \rho_0 \left(\frac{T_0}{T}\right)^{\left(\frac{g}{L_k R} - 1\right)} \tag{3.30}$$

For the altitudes used in this thesis, the considered layer goes from 0 km to 11 km of geopotential height. In this layer, the temperature linearly decreases with a rate of L_k=-6.5 K/km.

3.4.3. Wind Model

The wind is very important in the simulations, because it acts as a perturbation on the vehicle, altering its aerodynamic angles, and eventually its motion. The wind induces a change of angle of attack and sideslip, which are critical for the dynamic loads. The wind is modeled in this thesis as the sum of two terms. The first depends on the season and location, and it is a steady component, stable for hours. The second, an unpredictable one, has a short temporal duration.

For the steady component, also known as mean wind, multiple profiles are generated using the Horizontal Wind Model 14 (HWM14) from the US Naval Research Laboratory (Drob et al., 2015), which is an empirical model based on data from satellites, sounding rockets, and atmospheric balloons. It provides the zonal (north-south) and meridional (east-west) wind velocity components of a given set of latitude, longitude, and altitude values. It assumes that the wind primarily affects the horizontal plane, with the vertical component being negligible, as it is typically orders of magnitude smaller compared to the horizontal wind. The model's output varies with geographical position and time of year, including solar disturbances. Therefore, in the **UEN** reference frame, the mean wind can be represented as:

$$\mathbf{v}_{wind,const}(\mathbf{r}) = \begin{pmatrix} 0 \\ w_{EW}(\mathbf{r}) \\ w_{NS}(\mathbf{r}) \end{pmatrix} \tag{3.31}$$

Moreover, the short-term unstable components, made of turbulence and wind gusts, are superimposed to the steady mean term, altering the wind profile locally.

Turbulence is typically modeled as a combination of large-scale wave-like perturbations and small-scale stochastic variations, as described in the Earth GRAM model (White and Hoffman, 2021). For this thesis scenario, given the relatively small altitude and short time frame, only small-scale perturbations are considered. In GRAM the wind velocity perturbation is also cross-correlated with the density perturbation. The lack of data on these correlations leads to the use of a first-order auto-regressive model in this simulator. This model computes a perturbation at each position from the correlated perturbation value at the previous position. Considering a normalized variate $\mu(\mathbf{r}_k)$ (where $\mu(\mathbf{r}_k)$ is the value of the deviation of the variable with respect to the mean, at the position $\mathbf{r}_k$, divided by the standard deviation), the perturbation at the following position $\mathbf{r}_{k+1}$ is calculated as:

$$\mu(\mathbf{r}_{k+1}) = p\mu(\mathbf{r}_k) + \sqrt{1-p^2}q(\mathbf{r}) \tag{3.32}$$

$q(\mathbf{r})$ is a Gaussian-distributed random number with a mean equal to 0 and standard deviation equal to 1, while p is the auto-correlation term between two values at two successive positions $\mathbf{r}_k$ and $\mathbf{r}_{k+1}$. In this work, given the short time scale and horizontal displacement, the auto-correlation factor is assumed only dependent on the vertical position. Therefore:

$$p(\delta\mathbf{r}) = exp\left(-\frac{\delta r_x}{L_z}\right) \tag{3.33}$$

$\delta r_x = |r_x(k+1) - r_x(k)|$ is the vertical displacement between two positions and L_z is the vertical scale parameter, tabulated as in Figure 1 from Justus et al. (1990).

On top of this, the wind gusts are added. They are modeled as "1-cosine" shape, so that there is a peak of the wind gust at a given altitude, and it is dumped out at altitudes above and below it. For each new simulation, there is a 50% chance of creating a wind gust. As shown in Leahy (2008) the equation used to model them is the following:

$$\mathbf{v}_{wind,gust} = \begin{cases} 0 & \text{if } h<0 \\ \frac{\mathbf{v}_{max}}{2}\left[1-\cos\left(\frac{\pi h}{d}\right)\right] & \text{if } 0<h<2\text{d} \\ 0 & \text{if } h>2\text{d} \end{cases} \tag{3.34}$$

d is the gust half-width, randomly varying between 125 m and 375 m. $\mathbf{v}_{max}$ the vector of the maximum magnitude of the gust in correspondence of the half-width, randomly varying between ± 5 m/s and h the position of the vehicle inside the gust width. The wind gust is superimposed on the sum of mean wind and turbulence. Thus, the total wind is computed as:

$$\mathbf{v}_{wind}(\mathbf{r}) = \begin{pmatrix} 0 \\ w_{EW}(\mathbf{r}) + \mu(\mathbf{r})\sigma_{w_{EW}}(\mathbf{r}) \\ w_{NS}(\mathbf{r}) + \mu(\mathbf{r})\sigma_{w_{NS}}(\mathbf{r}) \end{pmatrix} + \mathbf{v}_{wind,gust} \tag{3.35}$$

3.4.4. MCI model

MCI can be modeled with different levels of complexity. The mass is considered variable, given the fuel consumption during the flight. The CoM is assumed to lie on the rocket's longitudinal axis ($\mathbf{x}_{CoM} = [x_{CoM}, 0, 0]$), and the inertia matrix is diagonal, containing only the principal moments of inertia. Due to the vehicle's axisymmetry, two of them are equal:

$$\mathbf{J}(t) = \begin{bmatrix} J_A(t) & 0 & 0 \\ 0 & J_N(t) & 0 \\ 0 & 0 & J_N(t) \end{bmatrix} \tag{3.36}$$

During such a short flight, the fuel variation is not extreme and the change in CoM and inertia could be considered negligible, assuming them to be constant, as done in the Benchmark scenario.

For more accurate results, a lookup table or analytical models can be used, as the CoM position affects the moment arm in external moment calculations, and the inertia influences rotational dynamics. Errors caused by using a low-fidelity model are dependent on the amount of propellant used in flight and are hard to estimate a priori. For instance, if an actuator deflection is calculated with a fixed CoM, but the actual real-flight CoM varies, the lever arm and resulting control moment will differ from the expected values, potentially strongly affecting the rocket's motion. Therefore, in the Full scenario, a more accurate analytical model is used for the MCI. As described in Simplício et al. (2020), it is based on the fuel mass depletion, with respect to the initial value:

$$\eta(t) = \frac{m_{prop}(t)}{m_{prop}(t_0)} \tag{3.37}$$

m_{prop} is the total propellant mass, which is the sum of fuel and oxidized.

The CoM is calculated using the varying masses and height levels of fuel and oxidizer in the tanks:

$$x_{CoM}(t) = \frac{1}{m(t)}\left[m_{fuel}(t)\left(h_{tank,fuel} + \frac{h_{fuel}(t)}{2}\right) + m_{ox}(t)\left(h_{tank,ox} + \frac{h_{ox}(t)}{2}\right) + m_{dry}h_{dry}\right] \tag{3.38}$$

$h_{tank,fuel}$ and $h_{tank,ox}$ are the heights of the tanks' bases with respect to the bottom of the rocket, $m_{fuel}(t)$ and $m_{ox}(t)$ are the masses of fuel and oxidizer at a given time step, m_{dry} is the dry mass and h_{dry} is the height of the CoM of the dry rocket. Moreover, $h_{fuel}(t)$ and $h_{ox}(t)$ is the level of propellant in each tank at a given time step, calculated as:

$$h_x(t) = \eta(t) d_{tank,x} \tag{3.39}$$

$d_{tank,x}$ is the length of each tank. Therefore, given constant density of each propellant, $\frac{h_x(t)}{2}$ represents the height of the CoM of each propellant with respect to the tank's base.

The axial J_A and lateral components J_N are calculated as:

$$J_A(t) = \frac{1}{2} m_{prop}(t) r_{tank}^2 + J_{A,dry} \tag{3.40}$$

$$\begin{aligned} J_N(t) =& \frac{1}{12} m_{fuel}(t)\left(3r_{tank}^2 + h_{fuel}^2(t)\right) + \frac{1}{12} m_{ox}(t)\left(3r_{tank}^2 + h_{ox}^2(t)\right) + J_{N,dry} + \\ & m_{fuel}(t)\left(h_{tank,fuel} + \frac{h_{fuel}(t)}{2} - x_{CoM}(t)\right)^2 + m_{ox}(t)\left(h_{tank,ox} + \frac{h_{ox}(x)}{2} - x_{CoM}(t)\right)^2 + \\ & m_{dry}(h_{dry} - x_{CoM})^2 \end{aligned} \tag{3.41}$$

r_{tank} is the tank radius. All the time-independent elements are tabulated in Tab. 3.2.

Table 3.2: MCI parameters.

Variable	**Value**	**Unit**
$h_{tank,fuel}$	1.2	m
$h_{tank,ox}$	5.4	m
$d_{tank,fuel}$	3.3	m
$d_{tank,ox}$	5.97	m
r_{tank}	0.7	m
h_{dry}	3.585	m
$J_{A,dry}$	5880	kg m^2
$J_{N,dry}$	39000	kg m^2
m_{dry}	2750	kg

4

Reinforcement Learning

In this chapter, a general overview of RL is given, starting from a description of the fundamentals of Reinforcement Learning in Section 4.1. Then, in Section 4.2, the combination of neural networks and RL is presented to show how Deep Reinforcement Learning algorithms work, while Section 4.3 it presents an overview of the most used ones. In Section 4.4, the concepts of meta-Reinforcement Learning is described, to show how multiple tasks can be learned in a single framework, using RNN or the concept of attention. Finally, a trade-off to choose the most suitable methods is performed in Section 4.5.

4.1. Introduction to Reinforcement Learning

Reinforcement Learning is a branch of Machine Learning, where an agent learns based on experience, rather than data. It consists of the most basic idea of learning: the interaction between a learner and an environment produces information about cause and effect, about consequences of actions, and about how to act to achieve a goal Sutton and Barto (2018). Therefore, the intelligent agent learns to make sequential optimal decisions in a dynamic environment, to maximize the positive feedback it receives. As explained in Chapter 2, RL is preferred over Supervised and Unsupervised Learning due to its broader generalization capabilities.

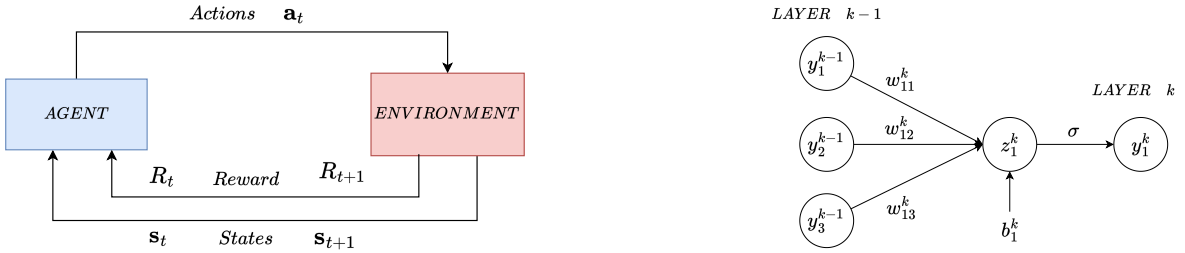
4.1.1. Reinforcement Learning Fundamentals

RL defines the learner as the **agent**, which repeatedly interacts with everything outside of it, known as the **environment**. As shown in Fig. 4.1(a), interactions between agent and environment take place with the latter passing **states** (or **observations**) to the former, which executes **actions**. Based on the actions performed, the agent receives a **reward**. The environment also determines the transition to new states, which are fed to the agent, so that it can take other actions, repeating the process until termination. The reward is a way to communicate to the agent **what** to achieve, and it expresses how good certain actions are to achieve a predefined goal. The concept of 'good' and 'bad' action is not necessarily binary and translating it into a mathematical formulation is one of the most challenging parts of RL. The purpose of this process is for the agent to learn a **policy**, which maps from states to actions, to execute the optimal one at every interaction.

4.1.2. Markov Decision Process

The theoretical formulation of a RL problem is based on the time-discrete Markov Decision Process (MDP), which is a mathematical framework for modeling sequential decision-making. This is not the only possible formalization and not all RL problems are formulated as MDPs.

A MDP is defined by a tuple of 4 elements: states, actions, reward, and state transition probability. In the RL problem formulated as MDP, the interaction between agent and environment takes place at each time step of a discrete sequence ($t = 0, 1, ..., n$). At each time step



(a) Agent-environment interaction.

(b) Neural network operations.

Figure 4.1: RL and NN architectures.

t, the agent is in a certain **state** $\mathbf{s}_t \in \mathcal{S}$, with $\mathcal{S}$ being the set of all possible states, and based on this it executes **actions** $\mathbf{a}_t \in \mathcal{A}$, with $\mathcal{A}$ being the set of all possible actions available at this particular state. As a consequence, the agent receives the next state $\mathbf{s}_{t+1}$ and a scalar **reward** $R_{t+1}(\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t+1}) \in \mathcal{R} \subset \mathbb{R}$, as shown in Fig. 4.1(a). The passage between the current and the following state is regulated by the **state transition probability** $\mathcal{P}(\mathbf{s}_{t+1}, R_{t+1}|\mathbf{s}_t, \mathbf{a}_t)$. It represents the conditional probability of moving to the next state $\mathbf{s}_{t+1}$ (and receiving a reward R_{t+1}), given the current state-action pair ($\mathbf{s}_t$ and $\mathbf{a}_t$). In a dynamic problem, it is equal to the propagation of the EOMs, given states and controls.

A system is defined as a MDP if and only if the state transition probability satisfies the Markov Property. This property states that the transition from the current to the next state solely depends on the latest state and action, which equivalently means that all the information about the previous transition is entirely captured in the present state, as described in Eq. (4.1)

$$\mathcal{P}(\mathbf{s}_{t+1}, R_{t+1}|\mathbf{s}_t, \mathbf{a}_t) = \mathcal{P}(\mathbf{s}_{t+1}, R_{t+1}|\mathbf{s}_t, \mathbf{a}_t, \mathbf{s}_{t-1}, \mathbf{a}_{t-1}, ..., \mathbf{s}_0, \mathbf{a}_0) \tag{4.1}$$

A generalization of the MDP, is the Partially Observable Markov Decision Process (POMDP), where the agent cannot observe the entire state vector, but only a subset of observations. It means that the ground truth is not completely known by the agent, limiting the amount of available information. While a MDP maps states to actions, a POMDP maps observations to actions. It represents many cases in the real world, where the agent has access only to a representation of the underlying states, rather than the actual ones. Unlike MDPs, where the executed actions are merely a function of the current states, in POMDPs past information influences the decision to be taken at the current time step.

4.1.3. Return, Policy and Value function

Return

RL methods maximize the cumulative sum of future rewards. Therefore, it is not the immediate reward at each time step to be maximized, but the cumulative reward over a trajectory (**return** G_t) starting at time step t, as shown in Eq. (4.2). Hence, suboptimal local actions could be taken, if they lead to a final larger return. For tasks with an undetermined final time, the agent may continue to interact with the environment to increase the return, for an infinite number of time steps ($T = \infty$). To avoid this issue, the sum of future **discounted** rewards is maximized:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... + \gamma^{T-1} R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{4.2}$$

γ is the **discount rate** and it represents the relative weight of future rewards with respect to immediate ones. It is bounded to $0 \leqslant \gamma < 1$, so that rewards infinitely in the future have a null contribution. If $\gamma = 0$, only the immediate reward is taken into account, while if $\gamma = 1$, the return is a sum of all future rewards, equally weighted.

Policy

The **policy** π is the law that rules the actions the agent takes based on the state it is in. It can be deterministic $\mathbf{a} = \pi(\mathbf{s})$, mapping a state to an action, or stochastic $\mathbf{a} \sim \pi(\mathbf{a}|\mathbf{s})$, representing the conditional probability of taking action $\mathbf{a}$, being in state $\mathbf{s}$. The final goal is to execute the optimal action for each state to maximize the return, resulting in the optimal policy π^*.

A **trajectory** is a sequence of n state-action pairs $\tau_n = \langle(\mathbf{s}_0, \mathbf{a}_0), ..., (\mathbf{s}_{n-1}, \mathbf{a}_{n-1})\rangle$ for the n time steps in a single episode.

Value functions

In almost all RL problems, the **value functions** are present.

The **state-value function** represents the "value" of being in a particular state. It expressed the expected return if the agent starts in a certain state $\mathbf{s}$ and acts according to the policy π for the rest of the episode:

$$V^\pi(\mathbf{s}) = \mathop{\mathbb{E}}_{\tau\sim\pi}(G_t|\mathbf{s}_t = \mathbf{s}) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|\mathbf{s}_t = \mathbf{s}\right) \tag{4.3}$$

$\mathop{\mathbb{E}}_{\tau\sim\pi}[\cdot]$ denotes the expected value given that the agent follows policy π.

The **state-action value function** is the expected return if the agent is in a state $\mathbf{s}$, executes action $\mathbf{a}$ and then acts according to the policy π for the rest of the episode:

$$Q^\pi(\mathbf{s}, \mathbf{a}) = \mathop{\mathbb{E}}_{\tau\sim\pi}(G_t|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}) = \mathop{\mathbb{E}}_{\tau\sim\pi}\left(\sum_{k=0}^{\infty}\gamma^k R_{t+k+1}|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}\right) \tag{4.4}$$

By following the optimal policy, the optimal state-value and state-action value functions $V^*(\mathbf{s})$ and $Q^*(\mathbf{s})$ can be obtained:

$$V^*(\mathbf{s}) = \mathop{max}_{\pi}\mathop{\mathbb{E}}_{\tau\sim\pi}(G_t|\mathbf{s}_t = \mathbf{s}) \qquad \text{and} \qquad Q^*(\mathbf{s}, \mathbf{a}) = \mathop{max}_{\pi}\mathop{\mathbb{E}}_{\tau\sim\pi}(G_t|\mathbf{s}_t = \mathbf{s}, \mathbf{a}_t = \mathbf{a}) \tag{4.5}$$

The difference between these two value functions is called **advantage function**. It represents how much a certain action is better than the one of the policy, by quantifying how much the expected return is improved (or worsened) by taking a specific action $\mathbf{a}$ in state $\mathbf{s}$, rather than selecting the action accordingly to the current policy $\pi(\mathbf{a}|\mathbf{s})$:

$$A^\pi(\mathbf{s}, \mathbf{a}) = Q^\pi(\mathbf{s}, \mathbf{a}) - V^\pi(\mathbf{s}) \tag{4.6}$$

4.1.4. Reinforcement Learning classification

The evolution of RL algorithms has led to different methods, making it is useful to classify them according to some properties. This basic classification gives a high-level idea of the different options available when using RL algorithms, but it is also useful to motivate design choices.

Model-based vs Model-free

The primary division that can be done is between model-based and model-free RL methods, depending on whether a model of the environment is required for learning.

Model-based algorithms need a model of the environment dynamics, that could be either provided to or learned by the agent. They are sample efficient, as they can predict future states and evaluate the results of possible different actions. The drawback is that the agent learns well how to behave in that particular environment but, since it is only an approximation of the real world, it delivers poorly once deployed in the real world. Hence, the agent is limited to performing well in a single environment. They are discarded in this thesis because the environment of interest is highly uncertain.

Model-free algorithms rely only on the feedback the agent receives from the environment. They do not require any knowledge of the model, making them more simple to implement and flexible to different scenarios, at a cost of less sample efficiency. This flexibility is very useful when the environment is not known, uncertain, or when the dynamics can change quickly.

On-Policy vs Off-Policy

The policy is what the agent uses to interact with the environment and take actions during learning, but it also is the final goal to be learned during the training. Depending on how the algorithm differentiates between these two functions, it can be defined as on-policy or off-policy.

Off-policy algorithms use two different policies for exploring and updating. The policy is updated by employing rewards and data collected using multiple different policies, such as those at previous iterations. Hence, there is a decoupling between the learning process and the update of the target policy. This allows for the collection of a wider number of data, making it more sample efficient. However, having dissimilar data from multiple policies (different from the current one) increases the variance and slows the convergence down. They are discarded in this thesis because they are inefficient in training for complex problems.

On-policy algorithms use the same policy to explore and exploit. At each iteration, the agent interacts with the environment using the most updated policy to make decisions. Several simulations are run to collect states, actions, and rewards. Then, the policy is updated and used to explore again at the following iterations. These algorithms are simple to implement because only the most recently collected data is used, not requiring large data storing capability.

Value-based vs Policy-based

Two approaches can be used to reach the optimal policy: value-based or policy-based.

In **value-based methods**, the agent iteratively learns an approximation of the state or state-action values to find the optimal value function, by evaluating the quality of all the states and/or actions it experiences. The optimal policy is then derived, selecting the action with the greatest expected return. These methods lack exploration, as the agent tends to choose actions with favorable expected returns. While effective for discrete states and small-scale problems, they require storing large amounts of data, which becomes impractical as the problem size grows. Thus, they are discarded in this thesis.

In **policy-based methods**, the agent directly learns the policy, without estimating state or action values. Not storing a large amount of data makes them suitable for large dimensional problems. The policy is typically parameterized by a NN, updated to maximize the expected return, using gradient-based or gradient-free methods. The agent learns a stochastic mapping between states and actions, with probability distributions as output, so that exploration is already included in the process. However, they are less sample-efficient (Arulkumaran et al., 2017).

4.1.5. Exploration and exploitation

A major challenge in RL in balancing **exploration** and **exploitation**, during training. Exploration allows the agent to discover new state-action pairs that might improve the policy's optimality, while exploitation focuses on choosing actions that yield the highest return. The agent's goal is to maximize cumulative reward, so it tends to favor high-reward actions. However, in a new environment, without exploration, the agent won't learn about potentially better state-action pairs. Conversely, too much exploration without exploiting known rewards slows down learning and convergence. Therefore, it is crucial to find the right balance.

The trade-off between these two components varies as the training proceeds. Initially, exploration should be prioritized to discover new state-action pairs, while later stages should emphasize exploitation to refine the policy for maximum return. Different RL algorithms handle this balance in various ways. Stochastic policy algorithms explore by sampling actions from a probability distribution during training and switch to deterministic actions during deployment. Deterministic policy algorithms, on the other hand, add noise to actions or use an ϵ-greedy method, where the agent primarily takes greedy actions but occasionally explores by selecting a random action with probability ϵ.

4.2. Deep Reinforcement Learning (DRL)

Early RL methods, known as tabular methods, stored values for each state-action pair, making them suitable only for small, discrete problems. Discretizing continuous spaces can introduce errors and exponentially increase the number of state-action pairs, leading to the "curse of dimensionality" (Bellman, 1984). To overcome this, function approximation techniques like radial basis functions, fuzzy logic systems, and linear regressions were developed. Among these, Artificial Neural Network (ANN) stands out as one of the best function approximation methods, thanks to its ability to capture non-linear relationships in high-dimensional problems. Networks with multiple layers are known as Deep Neural Networks (DNNs), and their application to RL is known as DRL. They address high-dimension problems previously intractable, mainly due to the computational limitations.

4.2.1. Artificial Neural Networks

ANNs are function approximators composed of interconnected nodes, called **neurons**, organized in subsequent layers, that mimic the structure of biological neural systems, mapping inputs to outputs. The first layer (**input layer**) receives the inputs, the intermediate ones (**hidden layers**) process the information, while the last layer (**output layer**) gives the final outputs. Each neuron acts as a mathematical gate to propagate information forward, while each layer captures different aspects of the information. Every neuron in one layer is connected to every neuron of the adjacent ones. A weight and a bias are associated with each connection.

The simplest architecture is the so-called **Fully connected NNs**, or **Multi Layer Perceptrons (MLP)**. They learn the optimal set of parameters $\boldsymbol{\theta}$ (i.e., weights and biases) that characterize the network, to find the best approximation of a function $\mathbf{y} = f^*(\mathbf{x}, \boldsymbol{\theta})$ which maps inputs $\mathbf{x}$ to outputs $\mathbf{y}$. In the following notation the superscript $^{(k)}$ indicates the k-th layer, the subscript $_{ij}$ indicates a connection between the i-th neuron in one layer and the j-th neuron in the next, while $_i$ indicates the i-th neuron in the current layer.

The first operation executed is a linear combination, for each neuron, of the weights $w_{ij}^{(k)}$ with the outputs from the previous layer $y_i^{(k-1)}$, adding the biases $w_{ij}^{(k)}$:

$$z_j^{(k)} = \sum_{i=1}^{N_i} w_{ij}^{(k)} y_i^{(k-1)} + b_j^{(k)} \quad \textbf{for} \quad j = 1, ..., N_j \tag{4.7}$$

N_j is the number of neurons in the current layer and N_i that of the previous layer.

These variables can be represented with vectors and matrices, for a more compact matrix notation (Eq. (4.8)). The neuron rescales the output with an activation function and passes it as output to the next layer's neurons (Eq. (4.9)), as shown in Fig. 4.1(b).

$$\mathbf{z}^{(k)} = \mathbf{w}^{(k)}\mathbf{y}^{(k-1)} + \mathbf{b}^{(k)} \tag{4.8}$$

$$\mathbf{y}^{(k)} = \sigma(\mathbf{z}^{(k)}) \tag{4.9}$$

At the end of the last layer, a cost function (J) is calculated.

After the forward pass from input to output, the cost function is backpropagated through the network, using the chain's rule of derivative to compute the partial derivative of the error with respect to the weights and biases, which are the set of parameters the NN depends on.

$$\frac{\partial J}{\partial \mathbf{w}^{(k)}} = \frac{\partial J}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}} \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{w}^{(k)}} \tag{4.10}$$

Using the derivatives, the set of parameters $\boldsymbol{\theta}$ is updated, for example using a gradient descent step. This improves the ability of the network to accurately map inputs to outputs. Repeating

this cycle for several iterations leads to a convergence of the parameters to the optimal set of weights and biases to best approximate f^* (Goodfellow et al., 2016). Algorithm 1 gives a general overview of the forward pass and backpropagation of a DNN.

Algorithm 1 Feedforward and backpropagation

Initialize parameters $\boldsymbol{\theta} = w_{ij}^{(k)}, b_i^{(k)}$
for $k = 1 : N$ **do** ▷ For every NN layer
  $\mathbf{z}^{(k)} = \mathbf{w}^{(k)}\mathbf{y}^{(k-1)} + \mathbf{b}^{(k)}$ ▷ Compute weighted sum for all neurons at k-th layer
  $\mathbf{y}^{(k)} = \sigma(\mathbf{z}^{(k)})$ ▷ Apply activation function
end for
$J = \mathbf{f}(\mathbf{y}^{(N)}, \boldsymbol{\theta})$ ▷ Calculate output cost function (algorithm dependent)
for $k = 1 : N$ **do**
  $\frac{\partial J}{\partial \mathbf{w}^{(k)}} = \frac{\partial J}{\partial \mathbf{y}^{(k)}} \frac{\partial \mathbf{y}^{(k)}}{\partial \mathbf{z}^{(k)}} \frac{\partial \mathbf{z}^{(k)}}{\partial \mathbf{w}^{(k)}}$ ▷ Derivative cost wrt weights
end for
Update set of parameters $\boldsymbol{\theta}$

4.3. Overview of Deep Reinforcement Learning algorithms

In addition to value-based and policy-based methods, DRL algorithms include actor-critic methods. These are hybrid algorithms, where the agent learns the value function and policy concurrently, combining advantages from both. The actor learns the policy and selects the action to take. Concurrently, the critic learns the value function which is used to evaluate the goodness of the action, estimating the expected return. Value function and policy can be learned with two independent networks or can be given as output from the same one.

4.3.1. Deep Q-Network (DQN)

Deep Q-Network (DQN) is the evolution of the Q-network method that uses NN, instead of tabular data, to approximate the value function, allowing higher-dimensional data. It is a value-based, off-policy algorithm, improving training stability with two innovative mechanism.

The first one is **Experience Replay**, which accelerates training by reusing past experiences of states, actions, and rewards stored in a replay memory. During training, the agent samples from this memory, allowing multiple updates using the same data, thereby reducing the need for new interactions. However, each update becomes slightly less impactful.

The second mechanism is the **Fixed Target Network**, where the parameters of a second value network are frozen, before being periodically updated to those of the value function network, avoiding abrupt network parameters changes and improving the training stability.

DQN is limited to discrete action spaces, with computational time increasing as the action space is finely discretized, making it inefficient for continuous control problems.

4.3.2. Policy Gradient Methods

Policy gradient methods are a policy-based family of algorithms, where the network learns a $\pi_{\boldsymbol{\theta}}$. The output of the policy network provides a stochastic representation of the action space, which is the probability of taking an action. Typically, the objective function to be maximized is the expected cumulative sum of rewards.

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}} \left(\sum_{t=0}^{T} \gamma^t R_t \right) \tag{4.11}$$

$\mathop{\mathbb{E}}_{\tau \sim \pi_{\boldsymbol{\theta}}}$ is the empirical average over a finite batch of samples from the policy distributions.

The core principle of policy gradient methods is that, at each iteration, the policy parameters $\boldsymbol{\theta}$ are updated using the stochastic gradient ascent rule in the direction of the estimate of the gradient with respect to the policy parameters $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k + \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_k)|_{\boldsymbol{\theta}_k} \tag{4.12}$$

α is the step size for the gradient step and $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ is a stochastic estimate of the gradient.

Policy gradient methods offer two key advantages over value-based ones: they handle continuous problems, essential for space guidance applications, and they avoid overestimating the value function because they don't learn it at all. However, since they typically are on-policy algorithms, they are less data-efficient, slowing down convergence performances.

4.3.3. Deep Deterministic Policy Gradient (DDPG)

DDPG is a model-free, off-policy algorithm with an actor-critic structure (Lillicrap et al., 2019). It is a deterministic policy gradient algorithm that can operate over continuous state and action spaces. It borrows experience replay and fixed target network from DQN to improve the training stability. Rather than sampling from a probability distribution, it uses a deterministic policy that outputs an action, adding noise ($\mathcal{N}$) to introduce exploration.

$$\pi'(\mathbf{s}_t|\boldsymbol{\theta}_t) = \pi(\mathbf{s}_t|\boldsymbol{\theta}_t) + \mathcal{N} \tag{4.13}$$

Being an actor-critic method, it simultaneously learns a value function and a policy. DDPG uses a conservative policy iteration on both the actor and the critic so that the target network values are constrained to change slowly, to avoid drastic updates that can affect the convergence.

4.3.4. Twin Delayed Deep Deterministic Policy Gradient (TD3)

Twin Delayed Deep Deterministic Policy Gradient (TD3) is an actor-critic, off-policy algorithm that builds on DDPG to enhance performance, particularly by addressing value function overestimation (Fujimoto et al., 2018). In an actor-critic model, policy and value updates are deeply coupled, because the value estimates diverge when the policy is poor and, vice versa, the policy degrades if the value estimate is inaccurate. Three strategies are used to mitigate this.

The first one is the **Clipped Double Q-learning**, where two independent value functions ($Q_{\boldsymbol{\theta}_1}$ and $Q_{\boldsymbol{\theta}_2}$) are learned, and the minimum of the two is used for the target loss. Underestimation is preferred as it's less prone to propagate errors compared to overestimation.

The second strategy is the **Delayed Policy Updates**, updating the policy network at a slower frequency compared to the value function to reduce the variance after several iterations.

The third strategy is the **Target Policy Smoothing** which adds noise to the target action to prevent the policy from exploiting overestimated value function errors. The noise acts as a regularizer, smoothing the value function to avoid policy overfitting. This avoids the policy exploiting a peak due to an error in value function approximation.

4.3.5. Soft-Actor Critic (SAC)

Soft-Actor Critic (SAC) is an off-policy, model-free algorithm, that introduces a stochastic component to improve DDPG, which is difficult to stabilize and sensitive to hyperparameters, due to interactions between the deterministic actor and value function (Haarnoja et al., 2018).

Similarly to TD3, it includes **Clipped Double Q-learning**, training two independent critic networks to avoid value function overestimation bias.

The other key feature is **Entropy Regularization**, which adds stochasticity to deterministic policy optimization by maximizing a weighted sum of cumulative return and entropy, balancing the exploitation-exploration trade-off. Entropy measures the unpredictability of the

agent's actions, promoting broader exploration of the state-action space and preventing the agent from getting stuck in suboptimal regions. Entropy ($\mathcal{H}$) is represented as a random variable (x) retrieved from a probability distribution $\mathcal{P}$ (Eq. (4.14)).

$$\mathcal{H}(\mathcal{P}) = \mathop{\mathbb{E}}_{x\sim\mathcal{P}}[-log\mathcal{P}(x)] \tag{4.14}$$

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\tau\sim\pi_{\boldsymbol{\theta}}}\left[\sum_{t=0}^{T}\gamma^t\Big(R_t(\mathbf{s}_t,\mathbf{a}_t) + \alpha\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t))\Big)\right] \tag{4.15}$$

α is the so-called temperature parameter, which controls the trade-off between entropy and reward, regulating the stochasticity of the policy. This parameter requires to be tuned to find the optimal level of entropy. After training, the stochasticity is typically removed to evaluate the policy's effectiveness by taking the expected value instead of sampling from a distribution. SAC has outperformed DDPG and TD3 in various benchmark continuous control tasks.

4.3.6. Trust Region Policy Optimization (TRPO)

Trust Region Policy Optimization (TRPO) is an on-policy, actor-critic algorithm that constrains the policy update step within a defined trust region. As shown in Eq. (4.16), it updates with the largest possible step to improve performance, while satisfying a KL-divergence constraint, which is a measure of distance between the current and the previous probability distributions of the policy. This constraint prevents the instability phenomenon experienced by standard policy gradient algorithms, where an overly large gradient ascent step can result in a destructive update. Using TRPO, this collapse is avoided and the performances improve monotonically during training, as proved in the original paper (Schulman et al., 2015).

$$\begin{aligned}\boldsymbol{\theta}_{k+1} &= \arg\max_{\boldsymbol{\theta}} \mathop{\mathbb{E}}_{\substack{\tau\sim\pi_{\boldsymbol{\theta}_k}\\ t=0,\dots,T}}\left[\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)}A_t^{\pi_{\boldsymbol{\theta}_k}}(\mathbf{s}_t,\mathbf{a}_t)\right]\\ &\text{subject to} \mathop{\mathbb{E}}_{\tau\sim\pi_{\boldsymbol{\theta}_k}}[KL(\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)), \pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t))] \leqslant \delta\end{aligned} \tag{4.16}$$

$\boldsymbol{\theta}_k$ and $\boldsymbol{\theta}$ are the set of parameters from the previous and current iterations, $KL(\cdot|\cdot)$ is the KL (Kullback-Leibler) divergence, and $A_t^{\pi_{\boldsymbol{\theta}_k}}(\mathbf{s}_t,\mathbf{a}_t)$ is the advantage function, at time step t. $\mathop{\mathbb{E}}_{\substack{\tau\sim\pi_{\boldsymbol{\theta}_k}\\ t=0,\dots,T}}[\cdot]$ represents the empirical average over a finite batch of samples, where each sample $(\mathbf{s}_t,\mathbf{a}_t)$ is sampled from the old policy $\pi_{\boldsymbol{\theta}_k}$. $\frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)}$ represents the ratio between taking a certain action, given some observations, with the new and old policies.

4.3.7. Proximal Policy Optimization (PPO)

PPO is an on-policy, actor-critic algorithm, considered an evolution of the TRPO algorithm (Schulman et al., 2017). Unlike TRPO, which is a second-order method and is complicated to implement due to the KL-constraint, PPO is a first-order method that includes the constraint inside the objective function, facilitating the algorithm implementation. The concept of conservative update between two iterations, to avoid destructive updates, is actuated by limiting the objective function using a clip operator. Recalling the policy probability ratio from TRPO (Eq.(4.17)), it is forced to stay in a small interval around 1, clipping it with the hyperparameter ϵ. The clipped objective function is shown in Eq. (4.18)

$$r_t(\boldsymbol{\theta}) = \frac{\pi_{\boldsymbol{\theta}}(\mathbf{a}_t|\mathbf{s}_t)}{\pi_{\boldsymbol{\theta}_k}(\mathbf{a}_t|\mathbf{s}_t)} \tag{4.17}$$

$$J^{\text{CLIP}}(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\substack{\tau\sim\pi_{\boldsymbol{\theta}_k}\\ t=0,\dots,T}}[\min[r_t(\boldsymbol{\theta}), \text{clip}(r_t(\boldsymbol{\theta}), 1-\epsilon, 1+\epsilon)]A^{\pi_{\boldsymbol{\theta}_k}}(\mathbf{s}_t,\mathbf{a}_t)] \tag{4.18}$$

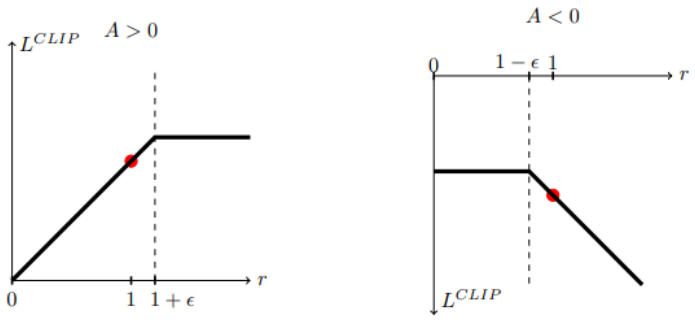


Figure 4.2: PPO clipped objective behavior.

The first term inside the minimum operator is the same objective as in TRPO (i.e., the product ratio r and advantage A), while the second one is the clipped term. By taking the minimum among the two, a conservative (but pessimistic) objective is obtained. The use of the clipped objective function has been shown capable of respecting the KL divergence constraint, aiding convergence by ensuring that the policy doesn't change drastically between updates. Figure 4.2 shows the behavior of the clipped function for positive and negative advantages. For instance, if the advantage function is positive, indicating that the chosen action is significantly better than others, and the probability of taking an action at the current step is larger than the previous one, the network's parameters are updated to select similar actions in the next iteration. However, the update is limited, to avoid exploiting too much that direction. Conversely, if the current action gives a negative advantage, the large negative objective value is not clipped, so that the parameters of the network are updated in the opposite direction, to avoid taking this bad action again. Hence, this is done to avoid the exploitation of outliers and abruptive changes, since the policy is updated at every iteration using an inherently limited set of data.

The objective function can be augmented with two terms. The first one is a quadratic error term of the value estimation, which is added only if the policy (actor) and the value (critic) are given as output from the same network. The second one is the entropy term, which can be added to increase the level of exploration. It is reminded that exploration is anyways included in PPO, given the stochastic policy from which actions are sampled. Equation (4.19) shows the complete objective function, where c_1 and c_2 are two parameters to regulate the weights of value function error and entropy exploration.

$$J^{\text{PPO}}(\boldsymbol{\theta}) = J^{\text{CLIP}}(\boldsymbol{\theta}) - c_1(V_{\boldsymbol{\theta}}(\mathbf{s}) - V_{target})^2 + c_2\mathcal{H}(\pi_{\boldsymbol{\theta}}(\cdot|\mathbf{s}_t)) \tag{4.19}$$

The parameters are then updated with Stochastic Gradient Descent (SGD) using Eq. (4.12). Since the advantage function is not known and it cannot be computed exactly, it is estimated, usually with the so-called Generalized Advantage Estimator (GAE), which uses the estimation of the value function. Therefore, the advantage function at time t_k is estimated using Eq. (4.20), while Eq. (4.21) represents the temporal difference error.

$$\hat{A}_k = \sum_{k'=k}^{N-1} (\lambda)^{k'-k}\delta_{k'} \tag{4.20}$$

$$\delta_k = R_k + \gamma V_{k+1}^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_{k+1}) - V_k^{\pi_{\boldsymbol{\theta}}}(\mathbf{s}_k) \tag{4.21}$$

It is remarked that the policy keeps its stochasticity only during training, to explore vaster spaces, while it is set as deterministic during deployment, by using the mean of the probability distribution. Since it is an on-policy algorithm, the collection of samples is alternated with the policy updates. In the rollouts, the agent interacts with the environment, using the current policy to collect states, actions and rewards. Then, they are used to update the policy, using

minibatch SGD for several epochs. Weights and biases are updated repeatedly, with results from different minibatches, to reduce computational time and average out statistical outliers.

PPO shows great results on many continuous control benchmarks, outperforming TRPO, using a much simpler implementation. It is vastly used in space guidance applications, currently considered the state-of-the-art algorithm, due to its consistent and reliable performances.

4.4. Meta-Reinforcement Learning

Meta-RL applies Meta-Learning to DRL (Beck et al., 2023). The goal of any Meta-Learning technique can be summarized in *learning to learn*. Unlike traditional RL, which trains a single policy for a single task, meta-RL trains an agent to succeed in various slightly different tasks $\mathbf{T}$, sampled from a continuous distribution $\mathcal{P}(\mathbf{T})$, similarly to how humans leverage prior experience for new, related tasks. This enables the agent to generalize and adapt its policy efficiently, making it more flexible in new, unseen environments. Each task $\mathbf{T}$ represents a different problem, with its own set of trajectories $D_{\mathbf{T}} = \tau_i \quad (i = 1, ..., n)$ and return $G_{\mathbf{T}}$.

The objective of meta-RL is to train an agent on a limited set of tasks, enabling it to perform optimally in any task within that distribution, even those not encountered during training. The goal is to learn network parameters $\boldsymbol{\theta}$ so that $\pi_{\boldsymbol{\theta}}(\mathbf{s}|\mathbf{a}; f_{\boldsymbol{\theta}}(\mathbf{T}))$ is the optimal control policy for tasks $\mathbf{T}$, maximizing the expected cumulative sum of reward over all the different tasks. Hence, the objective function is:

$$J(\boldsymbol{\theta}) = \mathop{\mathbb{E}}_{\mathbf{T}\sim\mathcal{P}(\mathbf{T})} \left\{ \mathop{\mathbb{E}}_{\tau\sim\pi(\cdot;f_{\boldsymbol{\theta}}(\mathbf{T}))} \left[G_{\mathbf{T}}(\tau) \right] \right\} \tag{4.22}$$

Meta-RL addresses the challenges that traditional DRL algorithms face when solving multiple related problems with varying environments or observations. Standard fully connected NNs often struggle to specialize outputs for different tasks, typically requiring larger networks, more data, and slower training. Unlike standard NNs, meta-RL excels in finding optimal policies for problems that can't be formulated as MDPs, such as POMDPs, where the optimal control depends on both current and past observations.

Two common approaches to implement meta-RL are Model-Agnostic Meta Learning (MAML) and memory-based NNs.

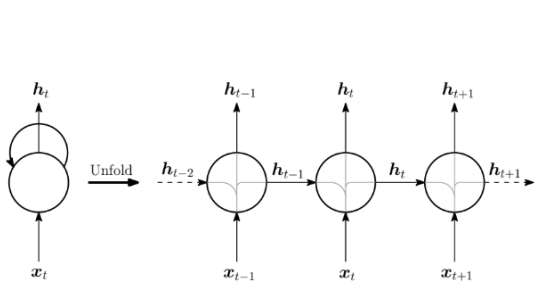
MAML, a gradient-based method, operates on two levels: an inner level updates parameters for a single MDP, and an outer "meta-update" tunes parameters to maximize returns across all tasks using only a few interactions with the environment, as shown in Eq. (4.23).

$$\begin{aligned} &\texttt{INNER} \Rightarrow && \boldsymbol{\theta}^{[j]} = \boldsymbol{\theta}_{(k)} + \alpha \nabla_{\boldsymbol{\theta}} J^{[j]}(\boldsymbol{\theta})|_{\boldsymbol{\theta}_{(k)}} \\ &\texttt{OUTER} \Rightarrow && \boldsymbol{\theta}_{(k+1)} = \boldsymbol{\theta}_{(k)} + \beta_k \left[\nabla_{\boldsymbol{\theta}} \textstyle\sum_{\mathcal{T}^{[j]}\sim\rho(\mathcal{T})} J^{[j]} \left(\boldsymbol{\theta}^{[j]} \right)_{\boldsymbol{\theta}_{(k)}} \right] \end{aligned} \tag{4.23}$$

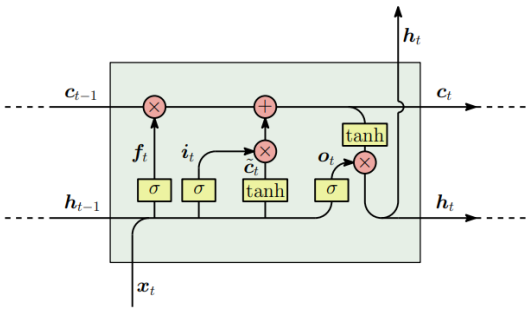
j is one of the n-th tasks of the inner level and k is the k-th iteration of the outer level. β_k is the step size of the outer level gradient step. This method is suitable for cases where the agent has "few shots" to adapt to a new environment or task, to update its parameters, such as a robot trained to cook in multiple kitchens that has to adapt to a new one, quickly.

The second approach uses "memory-augmented" neural networks, where the architecture is modified to include internal memory, allowing the network to learn temporal sequences (Wang et al., 2017). They use **recurrent** or **attention** layers, which help recognize features specific to each task by leveraging internal parameters that track temporal sequences. This method is particularly effective for tasks with the same dynamics but different uncertainties, enabling the agent to use the network's "memory" to recognize patterns and make better decisions.

Summing up, MAML learns on two levels, with the inner level that quickly specializes in new tasks, while the outer level slowly tunes the network parameters for generalization across



(a) Recurrent Neural Network, from Federici and Zavoli (2024).



(b) Architecture of a LSTM cell, from Federici (2022).

Figure 4.3: RNN and LSTM architectures.

tasks. On the other hand, memory-based neural networks are simultaneously trained on a set of tasks embedding features from a task distribution, which helps the network efficiently handle related problems. Additionally, MAML uses only one set of parameters $\boldsymbol{\theta}$, tuned with standard backpropagation, while meta-recurrent networks have additional hidden states. These two parameter sets facilitate faster convergence in uncertain dynamic systems, as hidden states develop their own policies to address multi-task environments.

Hence, memory-based networks are chosen to be used in the landing guidance application of this work, for their ability to recognize and generalize patterns across multiple tasks.

4.4.1. Recurrent neural network (RNN)

Unlike fully connected networks, RNNs retain memory to generalize an optimal control policy across multiple tasks. This family of networks was initially born to complete sentences in sound or language generation, where a single element depends not only on the last word. To enable this characteristic, they contain at least one *feedback connection*. At each step, the network's output is given as feedback, retaining information about the input data's temporal variation through internal hidden states $\mathbf{h}$. They are the essential constituent of RNNs, representing the "memory" of the network that keeps track of temporal dependencies in the input data.

Figure 4.3(a) and Eq. (4.24) illustrate a single RNN cell and its unfolded representation, where the output of a cell ($\mathbf{y}_t$) at a given time step t, does not only depend on the input data ($\mathbf{x}_t$) and the set of internal parameters $\boldsymbol{\theta}$, but also on the hidden states at the previous time step ($\mathbf{h}_{t-1}$). The length of unfolding, which corresponds to how many timesteps are retained by the hidden states, is a tuning parameter .

$$\begin{aligned} \mathbf{h}_t &= \sigma(\mathbf{w}_{hh}\mathbf{h}_{t-1} + \mathbf{w}_{hx}\mathbf{x}_t + \mathbf{b}_h) \\ \mathbf{y}_t &= \sigma(\mathbf{w}_{yh}\mathbf{h}_t + \mathbf{b}_y) \end{aligned} \tag{4.24}$$

$\mathbf{w}_{kk}$ and $\mathbf{b}_k$ are weight and bias matrices and $\sigma(\cdot)$ are activation functions. RNNs are trained using backpropagation through time (BPTT). Similarly to Eq. (4.10), the calculation of gradients is propagated for a number of time steps equal to the unfolding length of the recurrent layer, allowing to capture time dependencies. However, when gradients are propagated for many time steps, if they are too small or big, they tend to vanish or explode, losing information several time steps away from the current input data point. This is a problem of standard RNN, blocking the network from correctly understanding long-term dependencies. More refined RNNs, such LSTM cells, avoid this problem.

Long-Short Term Memory (LSTM)

LSTM networks were specifically designed to solve the problem of vanishing/exploding gradients (Hochreiter and Schmidhuber, 1997). The key idea is to have an additional cell state $\mathbf{c}_t$, separated from the hidden state $\mathbf{h}_t$. While standard RNN structure contains a single layer,

each module of the unrolled LSTM owns 4 interacting layers (or "gates") to get from the input to the output. The cell and hidden states act as a sort of summary of the most relevant feature of the previous input sequence. While the hidden state can be considered the "Short-Term memory", the cell state is the "Long-Term" one, capturing many time dependencies in the data. During the forward pass, the four gates decide how much information to let go through, as shown in the LSTM cell in Fig. 4.3(b). LSTM are very flexible also after training because their weights and biases ($\boldsymbol{\theta}$) are fixed, while the hidden and cell states are free to adapt to the input sequence also during deployment. The components of the LSTM network are:

- **Forget gate** f: It creates the *forget vector* f, based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$. By multiplying the previous cell state $\mathbf{c}_{t-1}$, it decides which information to drop (i.e. forget) from the memory.
- **Input gate** i: Based on the current input $\mathbf{x}_t$ and the previous hidden state $\mathbf{h}_{t-1}$, it creates the *input vector* $\mathbf{i}_t$, that represents which values to update. The input vector is transformed into a new estimate for the cell state $\tilde{\mathbf{c}}_t$, to be summed to the previous cell state, representing the new information that is worth to be included in the cell state.
- **Memory gate** m: It updates the previous cell state $\mathbf{c}_{t-1}$, using the forget vector $\mathbf{f}_t$ and new cell state estimate $\tilde{\mathbf{c}}_t$, in order to obtain the new cell state $\mathbf{c}_t$.
- **Output gate** o: It gives as output $\mathbf{o}_t$ the new hidden state $\mathbf{h}_t$, based on the previous hidden state and on the current input and cell state.

4.4.2. Concept of attention

Transformer NNs offer an efficient alternative to RNNs for processing sequential data by using attention layers instead of recurrent layers. This design allows Transformers to process sequences in parallel, unlike RNNs, which must handle inputs sequentially. Attention and Transformers, introduced in 2017 in "Attention is all you need" (Vaswani et al., 2017), revolutionized AI, especially generative AI, becoming the foundation of Large Language Models such as ChatGPT, Bard, Gemini, etc.

Before Transformers, RNNs and LSTMs were state-of-the-art for sequence modeling, but they were inherently limited by their sequential processing nature, which prevents parallelization and capturing long-distance dependencies. Feedback loops and hidden states in RNNs also tend to prioritize more recent information, making it difficult to discern dependencies between distant elements in a sequence. **Attention**-based networks eliminate these constraints, allowing parallelization and capturing dependencies over any distance. Additionally, by processing a sequence altogether, they do not compress past information in fixed-size hidden states, as done in RNNs, avoiding suffering from vanishing/exploding gradients.

The attention mechanism works by calculating similarities between elements in sequences, focusing on relevant parts for the task. Instead of creating a single context vector, as RNNs do, attention assigns weights to each element (i.e., token) based on its relevance. It relies on three components: query, key, and value, mapping a query and key-value pairs to an output by determining the affinity between queries and keys, to which values are associated.

The **query** is a representation of the information being sought, guiding the model to focus on relevant parts of the sequence. Each element of a sequence is associated with a **key**. Therefore, a concatenation of the keys is a high-dimensional representation of the input sequence. The **value** is the information linked with each key, depicting details or content associated with it.

Keys and queries are compared, and their affinity gives a set of weights, called attention score, used to compute a weighted sum of values and keys-queries pairs.

To summarize, a sequence is split into several tokens. For each token, three high-dimensional representations are created: the query, the key, and the value. They are vectors in a hyper-

space, encapsulating the meanings and features of the token. The similarities between keys and queries are computed, to generate scores, which are used to update the values, using a weighted sum. In this way, the "meaning" of each token is improved by updating its high-dimensional representation, based on how different tokens relate to each other.

4.4.3. Transformer neural network

In the first implementation in Vaswani et al. (2017), the Transformer is composed of an encoder and a decoder. The encoder maps the input sequence to an intermediate sequence of continuous representation, which the decoder then uses to generate the output sequence one element at a time, with each step's output serving as input for the next, in an auto-regressive manner. This is intuitive in a translation task, where the original sentence is first processed to an intermediate meaning, which is then translated into the target language word by word.

An input embedding and a positional encoding preprocess the inputs. Then, a Transformer layer is composed of a multi-head attention sub-layer and a feed-forward position-wise network sub-layer, each with a residual connection around it and followed by a normalization layer.

Embeddings

Embedding is a step done outside of the Transformer block and it manipulates the data to feed them as correct input to the Transformer. It converts data from an $n \times m$ input matrix to a $n \times d_{model}$ matrix, compliant with the Transformer dimension. n is the length of the sequence, while m is the size of the token. For instance, in this thesis, m is the number of states fed to the network. d_{model} is the dimension of the Transformer, also referred to as the attention dimension. In natural language processing, this module also has the task of first converting a token (i.e. word) into a numeric representation.

$$\mathbf{E} = \mathbf{w}_E \mathbf{Y} + \mathbf{b}_E \tag{4.25}$$

$\mathbf{Y}$ is the input data to the embedding layer, $\mathbf{w}$ and $\mathbf{b}$ are weights and biases and $\mathbf{E}$ is the output of the linear embedding layer. If there are multiple Transformer layers ($k > 1$), the embedding layer is only applied in the first one; subsequent layers use the output from the previous one as their input. $\mathbf{I}^k$ and $\mathbf{H^k}$ are the input and the output of the k-th layer.

$$\mathbf{I}^k = \begin{cases} \mathbf{E} & k{=}1 \\ \mathbf{H}^{k-1} & \text{otherwise} \end{cases} \tag{4.26}$$

Position encoding

In RNNs, the position of each token is inherently understood through sequential data processing. However, Transformers process data in parallel and lack this information. Hence, positional encoding is added to the input data using sinusoidal functions with varying wavelengths to encode elements' relative positions in the sequence (Vaswani et al., 2017).

Multi-head attention

The Multi-head attention is the heart of the Transformer. The inputs to this layer are the sequences of queries, keys, and values, made by n tokens with d_{model} dimensions. They are projections of the input sequence into three different matrices. The layer outputs a weighted sum of values, which is a function of the contextual similarity between queries and keys. It represents the additional significance, based on the tokens' similarity, to be added to each original embedding to update their meanings.

If the queries and keys-values pairs come from the same sequence, the attention is called self-attention, and it determines how relevant is a particular element with respect to other elements in the same sequence. When time series are processed, it is useful to use a masked version of the attention, so that tokens are compared only to previous elements. Usually, the

mask acts by setting to negative infinity elements that appear in future time steps, using activation functions, like softmax, that associate null weights to negative values.

For a single attention head, the attention function is computed simultaneously, using the **scaled masked dot-product**, on a set of queries, keys, and values packed into matrices $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ (Eq. (4.27)). The dot-product between queries and keys represents how similar/coincident are two tokens representations, giving the scores for each pair. The softmax function is applied to scale the weights between 0 and 1 before the weighted sum of the values. By multiplying the attention scores by the values, only those values multiplied by high scores are kept, filtering out irrelevant elements with poor affinity. The masked operator (χ) is used mainly in cases of temporal sequences.

$$\mathbf{A}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) = \text{softmax}\left(\frac{\chi(\mathbf{Q}\mathbf{K}^T)}{\sqrt{d_k}}\right)\mathbf{V} \tag{4.27}$$

The outputs of an attention layer are "delta vectors" representing relationships between tokens, to be added to the embeddings to modify and update their high-dimensional representations.

Instead of having a single attention head, with d_{model}-dimensional tokens, multi-head attention is performed in Transformers. The original queries, keys, and values matrices are split and projected h times (h is the number of attention heads), using different learned linear projections of dimensions d_k, d_k, and d_v for each token. This allows to capture multiple nuances of each element. The attention function (Eq. (4.27)) is applied in parallel to all the projected queries, keys, and values versions yielding a d_v-dimensional output. Finally, the outputs are concatenated, resulting in the final d_{model}-dimensional values, as shown in Eq. (4.28).

$$\begin{aligned} \text{Multi-Head Attention}(\mathbf{Q}, \mathbf{K}, \mathbf{V}) &= \text{Concat}[\text{Head}_1, \text{Head}_2, ..., \text{Head}_h]\mathbf{w}^O \\ \text{where} \quad \text{Head}_i &= \mathbf{A}(\mathbf{Q}\mathbf{w}_i^Q, \mathbf{K}\mathbf{w}_i^K, \mathbf{Q}\mathbf{w}_i^V) \end{aligned} \tag{4.28}$$

$\mathbf{w}_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $\mathbf{w}_i^K \in \mathbb{R}^{d_{model} \times d_k}$ and $\mathbf{w}_i^V \in \mathbb{R}^{d_{model} \times d_v}$ are the matrices used for the projection of queries, keys and values, respectively. $\mathbf{w}_i^0 \in \mathrm{R}^{hd_v \times d_{model}}$ is the matrix to concatenate from the multiple head attention. All these weight matrices are parameters learned by the Transformers, adjusting and updating them during the training process, to give the optimal input-output mapping.

Position-wise Feed-Forward Networks

A position-wise feed-forward network is applied to the output of the multi-head attention layer so that the correct output from one Transformer layer is given as input to the next Transformer layer. For the last Transformer layer, the outputs are the action probabilities (the so-called logits). The feed-forward layer consists of two linear transformations with a ReLU activation, applied to each position separately and independently (Eq. (4.29)). $\mathbf{X}$ is the input, while the $\mathbf{w}$ and $\mathbf{b}$ the learned weights and biases. A ReLU is an activation function defined as the maximum between its argument and zero.

$$\mathbf{f}(\mathbf{X}) = \text{ReLU}(0, \mathbf{X}\mathbf{w}_1 + \mathbf{b}_1)\mathbf{w}_2 + \mathbf{b}_2 \tag{4.29}$$

Layer normalization and residual connection

Layer normalization is usually used in combination with residual connections around each sub-layer (Ba et al., 2016). They are used in RNNs and Transformers to stabilize the hidden state dynamics, to make training faster and more straightforward (Federici and Furfaro, 2024). The two operations (Eq. (4.30)) are applied to each Transformer's sub-layer independently.

$$\mathbf{Y} = \text{LayerNorm}(\mathbf{X} + \text{SubLayer}(\mathbf{X})) \tag{4.30}$$

"SubLayer" represents a function, such as multi-head or feedforward, $\mathbf{X}$ is the input to the sub-layer, and "LayerNorm" is the layer normalization operation. The residual connection is the operation between brackets in Eq. (4.30): a sum of inputs and outputs of the sub-layer.

The layer normalization is a normalization of the residual connection, and it is carried out using its mean and standard deviation, as shown in Eq. (4.31).

$$\mathbf{h}(\mathbf{Y}) = \left[\frac{1}{[\boldsymbol{\sigma}]_{\times d_{model}}} \odot (\mathbf{Y} - [\boldsymbol{\mu}]_{\times d_{model}}) \right] \mathbf{G} + [\mathbf{b}]_{n\times} \tag{4.31}$$

$\mathbf{G} \in \mathbb{R}^{d_{model} \times d_{model}}$ and $\mathbf{b} \in \mathbb{R}^{d_{model}}$ are weights and bias matrices, $\odot$ is the element wise product operator and $[\cdot]_{\times d_{model}}$, $[\cdot]_{n\times}$ are operations to adjust the dimensions. $\mathbf{Y}$ is the residual connection output and $\boldsymbol{\mu}$, $\boldsymbol{\sigma}$ are the mean and standard deviation of the residual connection.

4.4.4. GTrXL

Due to their capability of grasping more distant relationships, attention-based networks provided significant gains in performance over LSTM in many SL domains, such as language modeling. Their ability to efficiently handle sequential information would make them the perfect candidate for partially observable RL problems, where the trajectory spans a large number of steps and the crucial observations for a decision are distributed over the entire length of the episode (Parisotto et al., 2019). However, their use in RL problems didn't take off rapidly, and most of the problems are still solved using LSTM, due to the easier LSTM implementation and the Transformer's difficulties of converging to a meaningful policy.

Parisotto et al. (2019) proposed an evolution of the Transformer architecture, suited to be used in a RL framework. This new architecture is called **Gated Transformer XL (GTrXL)**.

Its structure, shown in Fig. 4.4(a-b), is similar to the standard Transformer. The two main elements are the masked multi-head attention and the Position-Wise Feed Forward network. However, the normalization layer is now placed before each sub-layer, and a gating mechanism, is located immediately after each sub-layer, replacing the residual connection.

Considering that the objective of GTrXL is to process an input $\mathbf{Y}$ of total length L, the sequence is first divided into τ segments of length n, where each of the τ-th segments is

$$\mathbf{Y}_\tau = [\mathbf{y}_{\tau,1}, \ldots, \mathbf{y}_{\tau,n}] \in \mathbb{R}^{n\times m}, \quad \tau = 1, \ldots, L/n \tag{4.32}$$

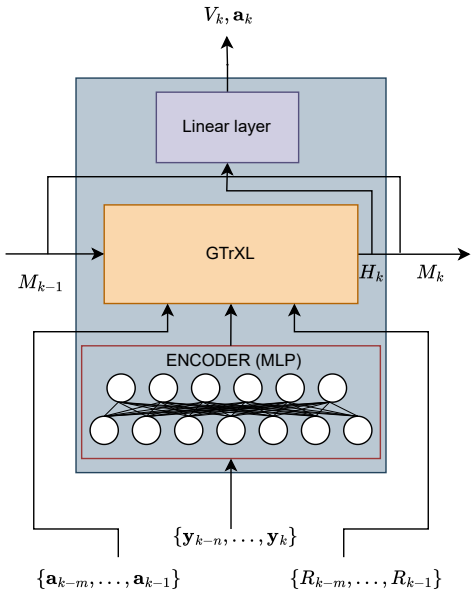
In the rocket landing problem, $\mathbf{Y}_\tau$ is a sequence of n temporally consecutive observations (each with dimension m). For the first Transformer layer ($k = 1$), the input $\mathbf{Y}_\tau$ passes through the initial encoder of fully connected layers, becoming $\mathbf{E}_\tau$. $\mathbf{E}_\tau$ can be augmented by adding a set of actions and rewards from the previous m time steps. Therefore, the input to the k-th Transformer layer is $\mathbf{I}_\tau^k$ and it follows the rule explained in Eq. (4.26). $\mathbf{E}_\tau$ is a high-dimensional representation of the input states sequence $\mathbf{Y}_\tau$. Moreover, the input to the Transformer layer is augmented by a memory, which contains the outputs of that specific k-th Transformer layer from the previous T segments (Eq. (4.33)).

$$\mathbf{M}_\tau^k = \left[\mathbf{H}_{\tau-T}^k, \mathbf{H}_{\tau-T+1}^k, \ldots, \mathbf{H}_{\tau-1}^k \right] \tag{4.33}$$

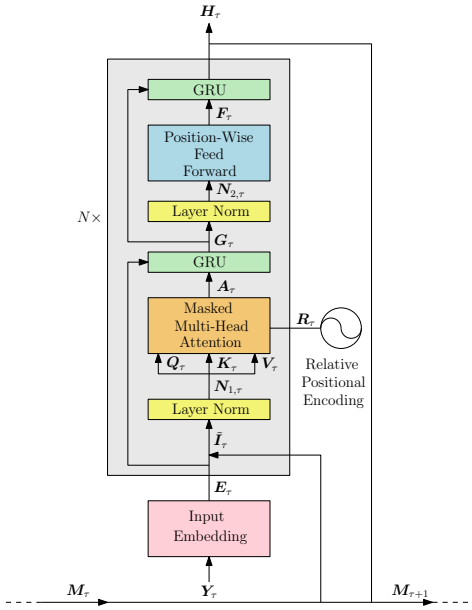
$$\tilde{\mathbf{I}}_\tau^k = \left[\mathrm{sg}(\mathbf{M}_\tau^k), \mathbf{I}_\tau^k \right] \tag{4.34}$$

The augmented input, shown in Eq. (4.34), uses a stop-gradient operator $\mathrm{sg}(\cdot)$, to keep the memory constant during weight updates, preventing the gradient from flowing backward during backpropagation. Then, it goes through the normalization layer (Eq. (4.35)).

$$\mathbf{N}_{1,\tau}^k = \mathbf{h}(\tilde{\mathbf{I}}_\tau^k) \tag{4.35}$$



(a) Schematic of GTrXL-based neural network.



(b) Schematic of GTrXL module, from Federici and Furfaro (2024).

Figure 4.4: GTrXL network architecture.

After that, the queries $\mathbf{Q}_\tau^k$, keys $\mathbf{K}_\tau^k$ and values $\mathbf{V}_\tau^k$ are generated to be fed to the multi-head attention. The queries are generated only from the part of $\mathbf{N}_{1,\tau}^k$ referred to the current input $\mathbf{I}_\tau^k$, excluding the previous memory. The queries are compared to key-value pairs that contain memory elements.

The output of the multi-head attention $\mathbf{A}_\tau^k$ is first passed through a ReLU, then through the gating mechanism, together with the current input of the Transformer layer:

$$\mathbf{G}_\tau^k = \mathbf{g}(\mathbf{I}_\tau^k, \max(0, \mathbf{A}_\tau^k)) \tag{4.36}$$

The three final steps to get the output $\mathbf{H}_\tau^k$ are: a second normalization layer (Eq. (4.37)), a position-wise fully connected network (Eq. (4.38)), and a gating mechanism (Eq. (4.39)), applied to the output of Eq. (4.36) and to the output of the fully connected network.

$$\mathbf{N}_{2,\tau}^k = \mathbf{h}(\mathbf{G}_\tau^k) \tag{4.37}$$

$$\mathbf{F}_\tau^k = \mathbf{f}(\mathbf{N}_{2,\tau}^k) \tag{4.38}$$

$$\mathbf{H}_\tau^k = \mathbf{g}(\mathbf{G}_\tau^k, \max(0, \mathbf{F}_\tau^k)) \tag{4.39}$$

Parisotto et al. (2019) found out that GTrXL achieved a stable, fast and reliable training. In challenging, high-dimensional continuous environments, GTrXL consistently outperformed LSTM in terms of policy performance and robustness to hyperparameter sensitivity.

This improvement is attributed to the placement of the normalization layer only on the input stream of the sub-layers, allowing an identity mapping from input to output. This helps during early training phases when the sub-layers output near-zero values, providing untransformed observations to the policy and value outputs. This approach facilitates learning a Markovian policy first, which is then refined with the use of memory to adapt to multiple tasks (Parisotto et al., 2019). In standard Transformers, input sequences are divided into shorter segments and processed one at a time. In contrast, GTrXL reuses its output from previous segments as a form of memory, similar to hidden states in RNNs.

Considering an architecture as in Fig. 4.4(a), the encoder input is composed by the last n observations $\mathbf{y}_t$ (from timesteps $t-n$ to t). Thus, the GTrXL block takes the encoder output, the memory $\mathbf{M}_t$ composed of the last T outputs of the Transformer layer, and the last $m-1$ actions and rewards as augmented input. The output of GTrXL $\mathbf{H}_t$ passes through a linear layer to obtain the current actions $\mathbf{a}_t$ and value function V_t. Recursively, the previous output of the Transformer $\mathbf{H}_{t-1}$ depends on the preceding n time steps and the previous memory vector $\mathbf{M}_{t-1}$. Hence, each time step's output relies on the most recent n steps of the most recent trajectory and the T vectors from the last T GTrXL outputs, containing information about the last T sequences of length n, from current and previous trajectories.

4.5. Trade-off

After presenting different RL algorithms and NN architectures, a trade-off analysis is performed to choose the most suitable methods to achieve the goals of this thesis. The outcome of the analysis is rationally explained, to show the critical thinking behind it. A set of criteria, shown in Tab. 4.1 together with their explanation, is identified to drive the trade-off process.

Table 4.1: Trade-off criteria.

Criterion	**Rationale**
Problem compatibility	The method should be compatible and usable with the definition of the problem at hand.
Implementation availability	The method should be available in a library, or should be implementable in a reasonable amount of time
Use in literature	Is the method widely used and considered state-of-the-art?
Potential improvements	Does the method bring innovation and improvements beyond current capabilities?

The trade-off for RL algorithms is straightforward. DQN is excluded due to its inability to handle continuous actions, making it incompatible with the problem analyzed. It also showed poor performances compared to PPO even in simpler problems (Wilson and Riccardi, 2021). DDPG, TD3, and SAC are also discarded as they are off-policy algorithms, using data from outdated policies and making network updates less meaningful. In contrast, PPO is an on-policy algorithm that offers better convergence, particularly in complex problems. It's an evolution of TRPO with easier implementation and it is widely used in continuous control problems, including space guidance (Gaudet et al., 2020b; Federici et al., 2021). Furthermore, Federici et al. (2023) found that PPO performs better than TD3 and SAC on the Earth-Mars transfer orbit problem. `Ray` provides a PPO implementation compatible with recurrent and attention-based NNs, unlike DDPG, TD3, and SAC which lack such implementations. Given the time constraints of this thesis, it is unfeasible to develop and test such methods from scratch. For the aforementioned reasons, PPO is chosen as RL algorithm.

Regarding NN architectures, simple feedforward networks are ruled out, due to their inability to leverage past information to take optimal actions. Recurrent layers have been largely used in literature, providing decent results in many space guidance problems, while attention-based networks have been introduced more recently. For instance, Transformers have vast applications in SL, but much less in RL. However, they have a huge potential in capturing long-distance relationships in time series, improving results with respect to recurrent layers, especially in complex problems. Therefore, it is chosen to explore the most innovative approach in this thesis, to ameliorate the performances and to compare the results with the LSTM state-of-the-art. The attention-based network used is the GTrXL (Parisotto et al., 2019), which is developed as a modification to the vanilla Transformer to enhance its capabilities in RL. The implementation is available in open-source libraries, such as `Rllib-Ray`.

5

Simulator Development and Verification

Figure 5.1 illustrates the top-level architecture of the simulator, highlighting its various modules. External forces and moments are calculated using the environment and vehicle models, based on current states and controls. These are then input into the equations of motion block, which outputs the propagated states. They are fed to the navigation module. If there is no navigation model, the navigation outputs are the same as the inputs: the propagated states. In this thesis, navigation errors are introduced as Gaussian white noise to simulate imperfect state knowledge, producing the estimated states. They are fed to the guidance module, which uses them to output the optimal control actions. The controls commanded by the guidance system, consisting of fin deflections and thrust vector, are perturbed to represent imperfect modeling of the actuators. Closing the loop, the actual states and controls are then used to calculate forces and moments, to propagate for another time step.

Section 5.1 presents a preliminary integrator analysis, while all the following sections illustrate the verification of the models used in the simulator. Section 5.12 shows the models and initial conditions used in the two simulation scenarios employed in this thesis.

5.1. Integrator

Based on the literature survey, the most used integrator in short-duration landing problems is a fixed time step Runge Kutta 4th order (RK4) (Gaudet et al., 2020b, 2022). Since the forces at play do not change significantly during the short flight, the same order of magnitude of "dynamics speed" is exhibited throughout the entire simulation. Thus, given the absence of different time scales, a variable-step integrator to capture them is not needed. Using a fixed time step, increasing the integrator's order would only add more prediction steps between time epochs, increasing function evaluations and computational time. The running frequency of the onboard G&C system constraints the simulation time step to be small, reducing local errors and allowing for a lower-order integrator because the G&C frequent updates mitigate also numerical errors.

An integrator analysis is performed to select the best time step, using the nominal optimal trajectory. `ode45` is chosen for two reasons: firstly, there are equivalent formulations in MATLAB and Python (`scipy`), ensuring reproducibility and fair comparison between OCP results in MATLAB and RL results in Python. Secondly, `ode45` is a low-order integrator and it supports the use of a fixed time step, aligning with the earlier reasoning for opting for a fixed-step low-order integrator. When using a fixed time step, `ode45` is RK4. A third independent RK4 implementation is used to verify that the fixed time step versions of `ode45` in MATLAB and Python match the theoretical RK4 version. After this verification, the built-in MATLAB and

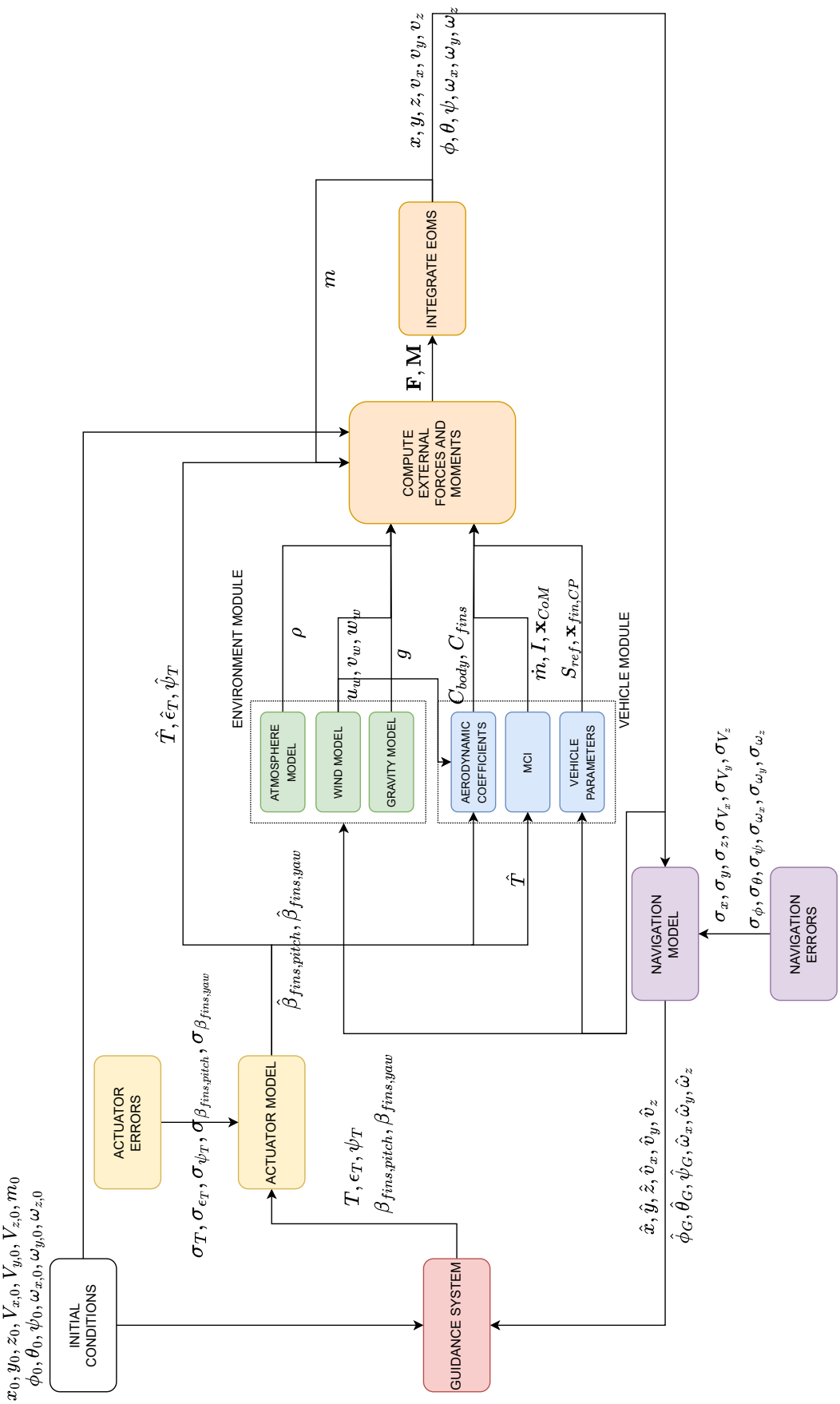
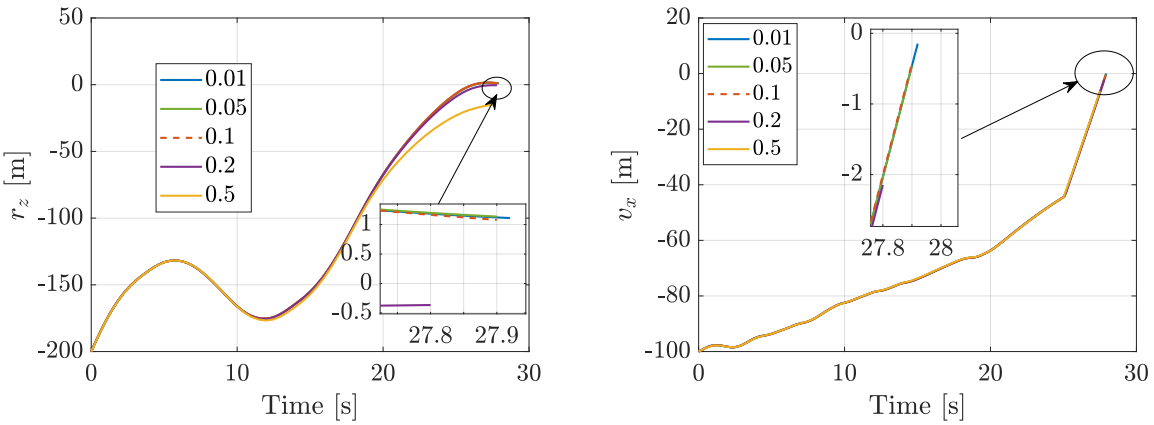


Figure 5.1: Top-level simulator architecture.



(a) r_z propagation accuracy using different time steps.

(b) v_x propagation accuracy using different time steps.

Figure 5.2: Propagation of state variables using different time steps.

Python versions are used for their easy handling of event functions and data retrieval.

Afterward, a time step analysis is performed to trade-off accuracy and run time. It is essential to keep CPU time low, as it significantly impacts the RL training process, which involves hundreds of thousands of simulations. Figures 5.2(a-b) show results for two state variables (r_z and v_x) using different time steps. They are representative of trends across all variables, but only two are shown for the sake of conciseness. The simulation with a time step of 0.01 seconds yields the most accurate results, but its runtime is too large to be used in NN training. It can be immediately seen that a time step of 0.5 seconds is too large, giving significant errors on both vertical velocity and z-position. For instance, the latter is 15 m, exceeding the final position constraint. Simulations with 0.1 s and 0.05 s as time steps yield similar errors, though the latter is discarded because it has double the runtime. A step of 0.2 seconds reduces the run time even more, to the detriment of an increase in terminal errors. It is discarded because the error on vertical velocity is too large. By ending 0.15 s earlier, it violates the restrictive terminal constraint of 2 m/s.

Therefore, a time step of 0.1 s is selected as an acceptable trade-off for the RL problem. The simulation ends 0.02 s earlier, due to the larger time step, providing an error of 0.5 m/s on vertical velocity. Table 5.1 shows the CPU times on an Intel® Core™ i7-8565U processor.

5.2. Frame transformations

All the relevant transformations between reference frames are verified, comparing the results with analytical expression, using multiple angles, to ensure the same output.

UEN-frame to $\mathcal{B}$-frame

One example of the verification is reported here. The transformation from the landing site **UEN** reference frame to the body reference frame is defined by the attitude angles (roll, pitch, yaw). The rotation matrix is the same as reported in Equation (B.12):

$$\begin{aligned}\mathbf{C}_{\mathcal{B},UEN} &= \mathbf{C}_1(\phi)\mathbf{C}_2(\theta)\mathbf{C}_3(\psi) = \\ &= \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \cos\theta\sin\phi \\ \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\phi\cos\theta \end{bmatrix}\end{aligned} \tag{5.1}$$

Using for example a set of Euler angles equal to ϕ=45 deg, θ=30 deg, ψ=60 deg, the unit vector $\hat{\mathbf{x}}_{UEN} = (0, 0, 1)$ should become $\hat{\mathbf{x}}_{\mathcal{B}} = (-1/2, \sqrt{6}/4, \sqrt{6}/4)$. The output of the transformation function in the simulator is the same as the analytical one, verifying the correct implementation.

Similarly, every other frame transformation involved in the dynamics is verified.

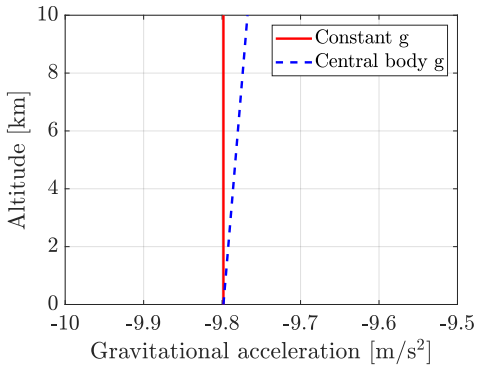
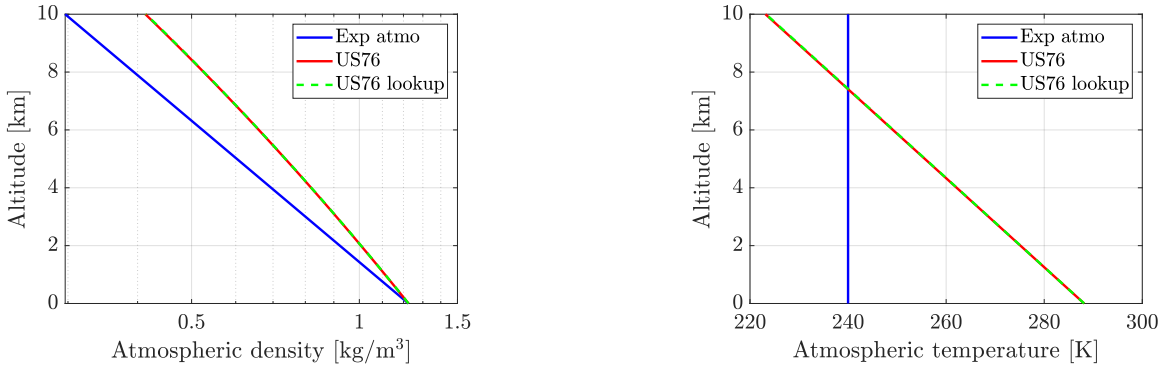


Figure 5.3: Gravity comparison constant vs central body.

Table 5.1: Run times.

dt	Run time
0.01 s	27.096 s
0.05 s	5.706 s
0.1 s	2.628 s
0.2 s	1.174 s
0.5 s	0.508 s



(a) Density comparison.

(b) Temperature comparison.

Figure 5.4: Atmosphere models verification.

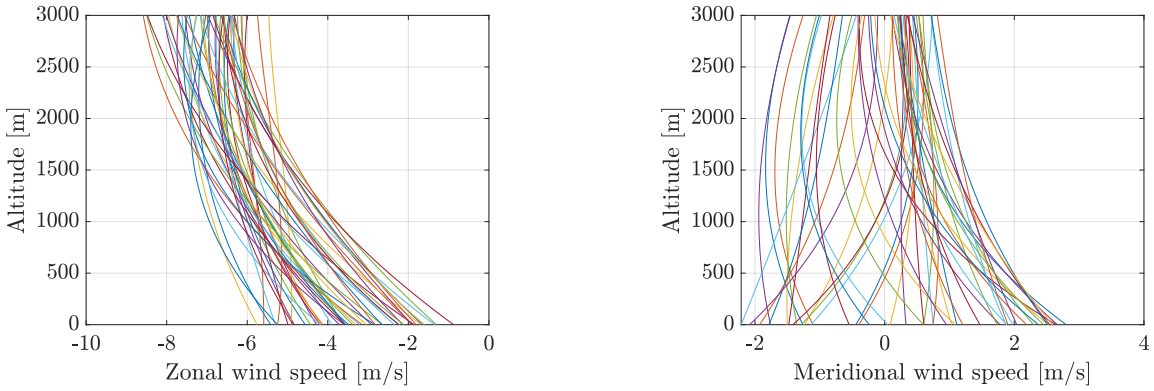
5.3. Gravity model

The gravitational acceleration, using the central body analytical formulation, is plotted and compared to the constant value of 9.7982867 m/s^2 from Table 3.1.
In Fig. 5.3, a comparison between the constant gravity value and the magnitude of the central body gravity field (as in Eq. (3.25)) is presented. As expected, it shows a decreasing gravitational acceleration with altitude, when considering a central body model.

5.4. Atmospheric model

The density of the US76 model is compared to that of the exponential atmosphere model, as the two should be relatively close, given the small altitude variations considered in this problem. Moreover, the US76 analytical model is compared to a lookup table provided by Marco Sagliano. For US76, reference values are p_0=101325 Pa, ρ_0=1.224999156 kg/m^3 and T_0=288.15 K at h=0 m, while H_s=7050 m (corresponding to a constant temperature of 240 K) and ρ_0=1.224999156 kg/m^3 are used for the exponential atmosphere. Figure 5.4(a) shows the three models, plotted up to 10 km of altitude, which is a reasonable altitude range for landing scenarios with slightly larger initial conditions than the ones used in this thesis.

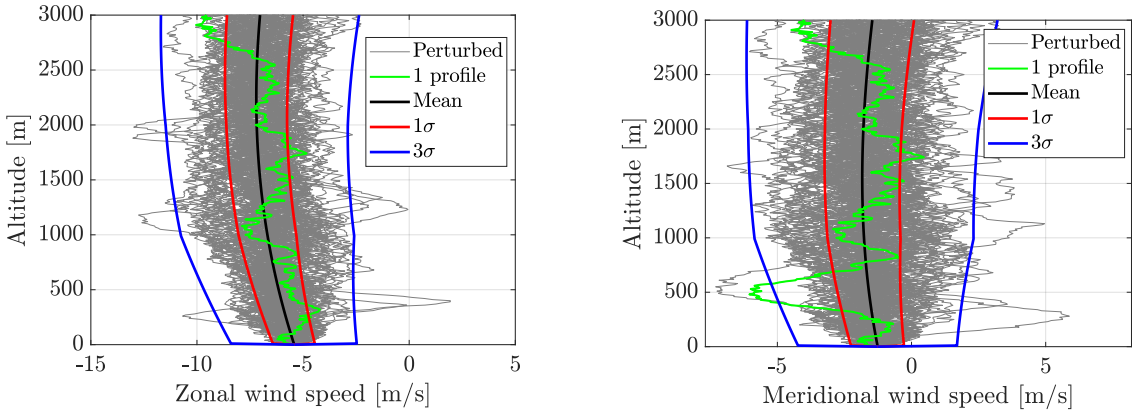
The density is coincident for the two US76 formulations, verifying the correct implementation of the analytical model. Moreover, there is an expected difference between the exponential and the US76 densities, due to their different underlying assumptions, but a similar trend is shown by both models, consistent with the assumptions. The local errors between US76 and the exponential model are about 10% at 2 km of altitude, justifying the use of the more accurate model. Furthermore, in Fig. 5.4(b), it is shown that the temperatures reflect the expected trends of constant value for exponential atmosphere and linearly decreasing for US76.



(a) East-west constant wind profiles.

(b) North-south constant wind profiles.

Figure 5.5: 50 constant wind profiles for Kourou over a year.



(a) East-west wind profiles.

(b) North-south wind profiles.

Figure 5.6: 100 complete wind profiles for a single mean profile.

5.5. Wind model

The wind is modeled as the sum of a constant term, a turbulent one, and wind gusts.

For the constant component, a set of profiles from different days and hours over a year are obtained from the Horizontal Wind Model 14. In Fig. 5.5(a-b), 50 constant profiles, equally distributed over 365 days, are plotted, taking Kourou (French Guyana) as the location. These figures show differences throughout the year, as well as a persistent trend where the zonal wind component is consistently larger than the meridional one. As a new simulation starts, a profile is sampled from this set, adding turbulence and gusts on top of it.

Figures 5.6(a-b) present 100 complete wind profiles for the horizontal east-west and north-south components, where turbulence and gusts are superimposed to a single constant profile. In some cases, the ”cosine” shape that represents a discrete wind gust is clearly visible. 1σ and 3σ dispersions are shown for the turbulence, illustrating the dispersion of wind velocities.

5.6. Aerodynamic Model

The aerodynamic models, both for the fins and the body vehicle, are verified by critical reasoning on the outcome of a set of cases. A few test cases are reported in Appendix C. The results of these tests show that the aerodynamics of the body and fins are correctly modeled, generating forces and moments with correct orientation, direction, and magnitude, as expected from a qualitative analysis of the flight motion with certain aerodynamic angles.

An auxiliary sideslip angle is introduced to correctly calculate the aerodynamic coefficients and the center of pressure locations, during the descent phase, when the vehicle has an angle

of attack close to 180 deg (or -180 deg). It is defined as the supplementary angle of the sideslip angle if the angle of attack is larger than $\pi/2$ or smaller than $-\pi/2$ (i.e., $\cos\alpha < 0$):

$$\beta^* = \pi - \beta \qquad \text{if } \cos\alpha < 0 \tag{5.2}$$

The reason for this auxiliary angle is that the aerodynamic database is two-dimensional. Hence, the interpolation of the coefficients depends only on a single aerodynamic angle, rather than both angle of attack and sideslip simultaneously. Thus, the computation of the aerodynamics, for both the body of the vehicle and the fins, is decoupled for the two aerodynamic angles. Since these aerodynamic forces are treated separately, this adjustment has to be done, to ensure both aerodynamic angles are used correctly. Moreover, the ambiguity of the sideslip angle and its supplementary angle stems from calculating it using the arcsin, which is bounded it between $\pm\frac{\pi}{2}$. Given this boundary, for a $v^{\mathcal{B}}_{A,y}$ velocity, the sideslip angle is the same if the vehicle is either ascending or descending, due to the arcsin and norm operators (Eq. (B.2)). Since the aerodynamic data is based on an ascending rocket, the forces would appear the same, though in reality, they have different directions. Therefore, during descent, the supplementary sideslip angle must be used for accurate interpolation of the C_Y coefficient, center of pressure due to the sideslip angle, and fins aerodynamic coefficient on the yaw axis. Two examples of the use of this auxiliary angle are presented during the verification of the center of pressure of the body vehicle and the local angle of attack of the fins on the yaw axis, in Appendix C.1.1 and C.1.2. Figure C.2 also gives a visualization of β^*, clarifying its use during the descent flight. Hence, its correct introduction and use are confirmed by these test cases.

5.7. Propulsion Model

The main concern with the propulsion force is the correct rotation from the propulsion frame to the body frame, according to the TVC angles. Therefore, the verification of this rotation is done for cases where the visualization of the thrust direction is straightforward. Similar to what is done for the aerodynamics, a test case is reported in Appendix C.2, showing the correct implementation of this model.

5.8. MCI Model

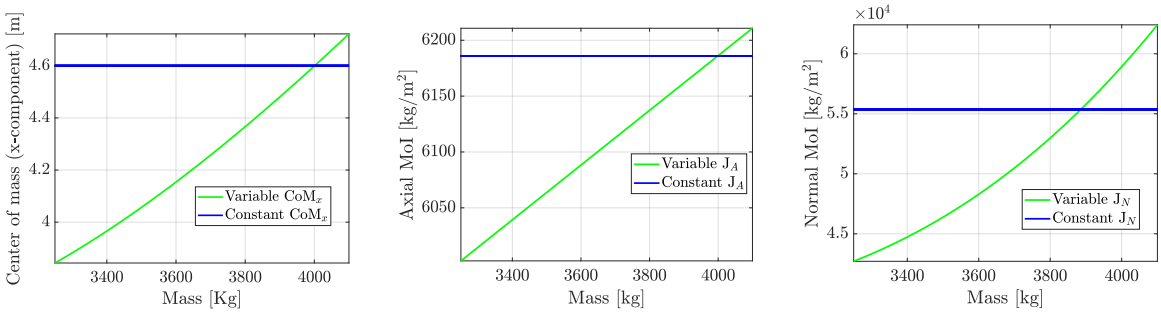
In Fig. 5.7(a-c), the verification of the MCI model is shown, confirming that the variable model behaves as expected from the analytical expressions in Eq. (3.38), (3.40), and (3.41). It demonstrates a decreasing center of mass and moments of inertia, as the mass reduces. Indeed, the dry mass is concentrated in the lower part of the vehicle, where the engine elements account for the largest part. On the other hand, the constant model shows constant values for the entire mass range, as it should be. While the differences in inertia between the models are less relevant in the dynamics equations, a CoM variation of more than 0.5 m strongly affects the lever arm used in moment calculations, confirming the importance of using a variable MCI model.

5.9. MATLAB-Python comparison

Two identical flight simulators are developed for the OCP and RL parts, in MATLAB and Python, respectively. This is done because typical Optimal Control solvers, such as GPOPS or ICLOCS, have MATLAB implementations, while there are many Python libraries available, such as `Gym`, `Ray`, and `Tensorflow`, optimized to create a RL framework to train NNs. Nevertheless, it must be tested that the outputs of two the two simulators are identical.

The verification consists of feeding the control outputs from the optimization into both simulators, propagating the EOMs using the same integrator and interpolation methods, and checking that the difference is always tiny, for all variables.

In Fig. 5.8, it can be seen the errors for one position, velocity, angular rate, and quaternion



(a) Center of mass. **(b)** Axial moment of inertia. **(c)** Normal moment of inertia.

Figure 5.7: MCI models comparison and verification.

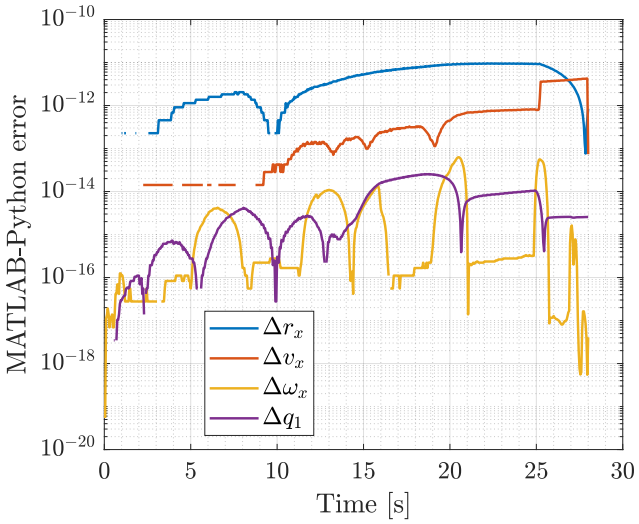


Figure 5.8: MATLAB-Python errors.

variables, are between machine epsilon and 10^{-11}, confirming sufficient accuracy. The missing data points indicate errors exactly equal to zero. All the other states present errors in the same order of magnitude, or lower, but their figures are not reported, for the sake of conciseness.

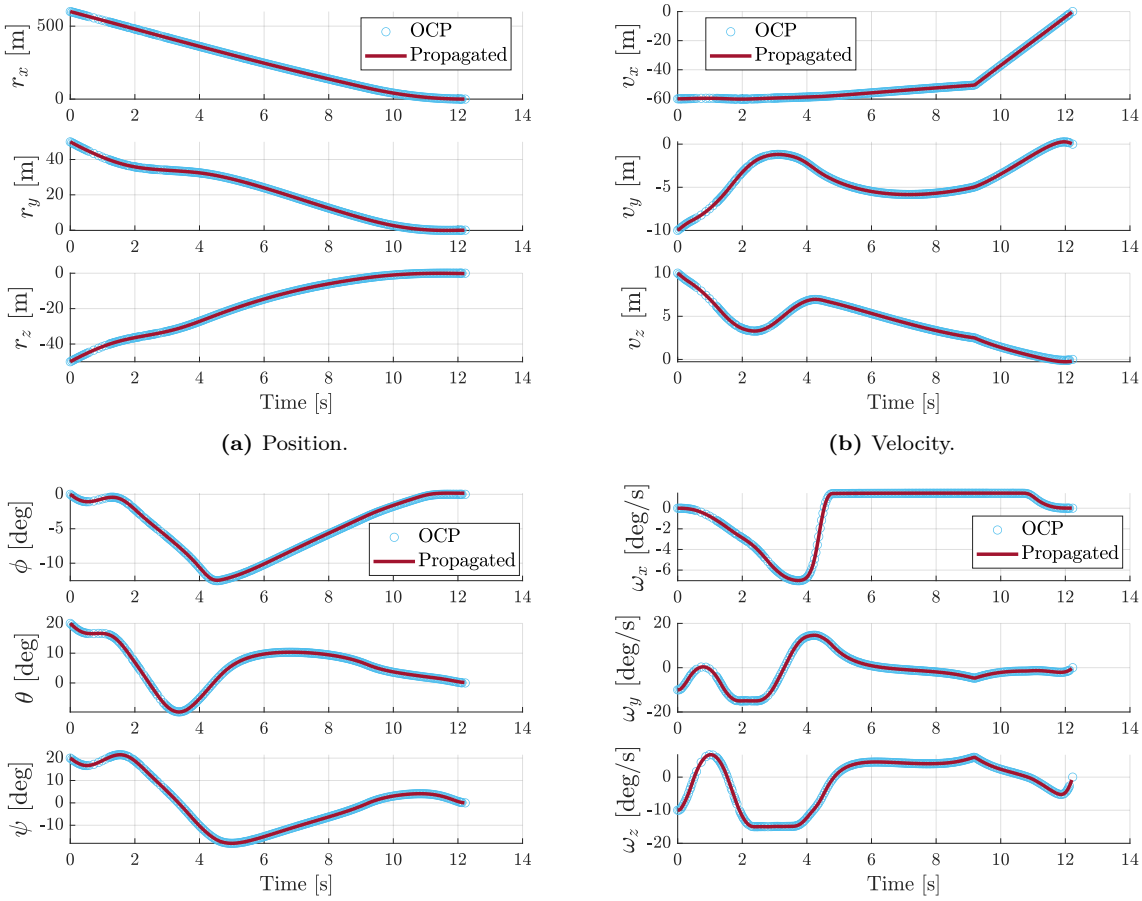
5.10. Simulator verification

5.10.1. Benchmark choice

A benchmark is chosen to compare and verify the flight simulator and optimal control solution developed in this thesis. The work from Lee and Lee (2022) is identified as the most compatible problem, because it is the only one implementing a feedback policy, using thrust magnitude, TVC, and fins as guidance outputs, as in this thesis. They develop a multi-phase guidance strategy for a reusable rocket, in a 6-DOF framework, for both aerodynamic descent and powered landing phases. Due to the interest in only the powered landing phase, all the considerations about the aerodynamic descent are not reported here. The reference vehicle of the benchmark is the same one used in this thesis, as shown in Table 2.1. The initial conditions of the benchmark are summarized in the "Benchmark Scenario" column in Table 5.2.

They develop a guidance system that aims at a fuel-optimal precise pinpoint landing, satisfying path constraints on vertical angle, angular rates, maximum and minimum thrust magnitude, deflection angles, and rates of TVC and fins. Different from the direct optimization method used in this thesis, the benchmark paper solves the problem using SCvx methods.

The benchmark introduces some assumptions to facilitate the solution and convergence of the convex optimization process. For example, the reference frame is centered on the landing



(a) Position. **(b)** Velocity. **(c)** Euler angles. **(d)** Angular rates.

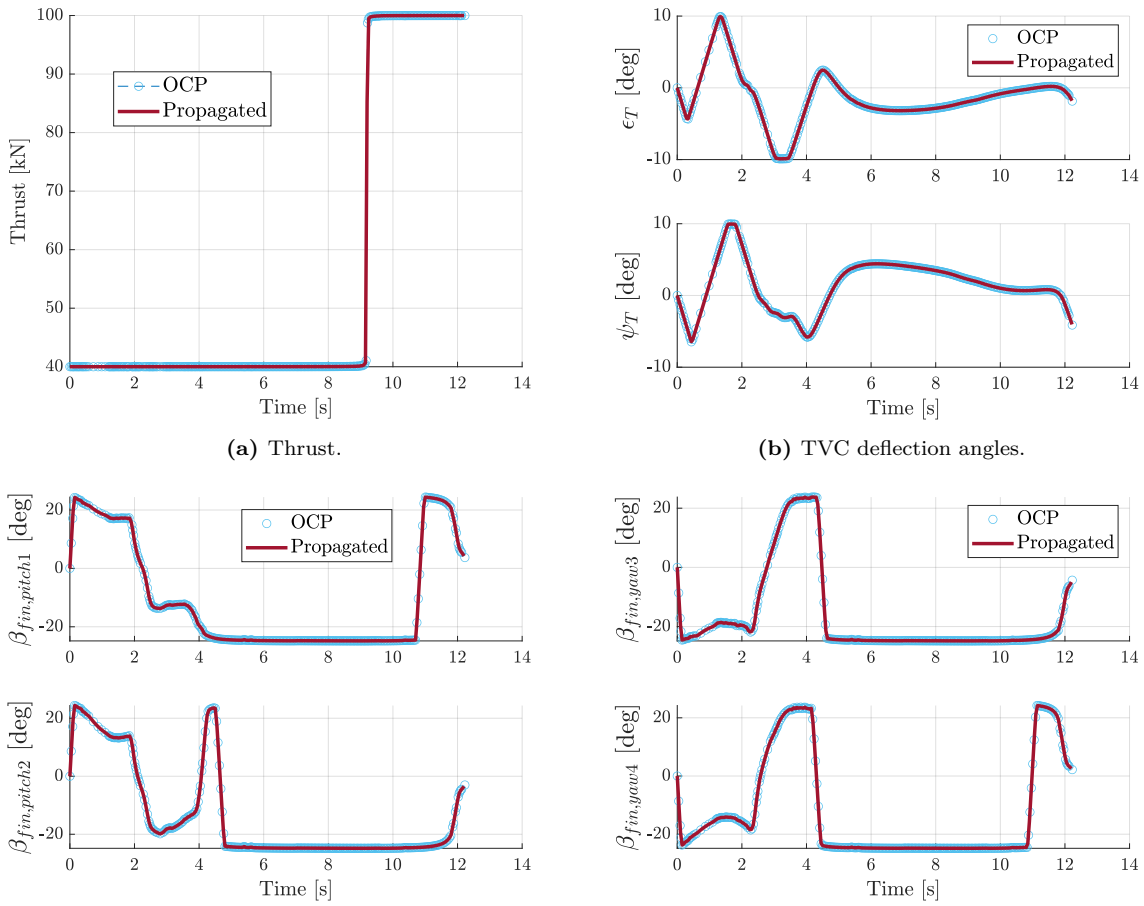
Figure 5.9: States nominal trajectory. - Benchmark scenario

site, but assumes inertial, neglecting the fictitious forces introduced by the Earth's rotation in a rotating frame. Furthermore, wind is not included, while constant gravitational acceleration and constant center of mass and inertia are used. To replicate the benchmark, these assumptions and simplified models are kept, but for the "Complete Scenario" the complexity of the models is increased, to reduce the gap between simulation and results.

5.10.2. Verification

After the verification of each block, the entire simulator is verified in two steps. First, the optimal trajectory for the Benchmark scenario is computed using the direct optimization method explained in Chapter 6. The results are compared with those obtained by Lee and Lee (2022). Even if two different optimization methods are employed, the outcome should be similar. Afterward, the optimal controls from each node are fed into the simulator in open-loop, to check the physical feasibility of the OCP solver's discrete solution.

In Fig. 5.9(a)-(d), the translational and rotational states for the Benchmark Scenario are shown, comparing the outputs of the OCP software at each node to the open-loop solution obtained by interpolating the controls and propagating the dynamics, using `ode45` with a time step of 0.1 s. The nominal optimal trajectory is generated with direct optimization. A direct collocation with Hermite-Simpson transcription method is used in ICLOCS, as explained in Section 6.1.1. The optimal trajectory is obtained using six mesh refinement iterations, starting with 15 nodes and ending with 297, taking a total of 2 hours and 27 minutes. It can be seen that the propagated open-loop trajectory closely follows the solutions at the nodes, meaning that the optimal one is physically feasible. The propagated trajectory ends 0.03 s before



(a) Thrust.

(b) TVC deflection angles.

(c) Fins pitch pair deflection angles.

(d) Fins yaw pair deflection angles.

Figure 5.10: Controls nominal trajectory. - Benchmark scenario

the terminal node, introducing small errors in the terminal states, such as at cm level for the horizontal position. However, this is less than a single time step, and it is due to a combination of the tolerance errors of the discretized optimal control solution and numerical integration and interpolation errors. In Fig. 5.10(a)-(d), the controls are shown, confirming that the discretized and interpolated variables are nearly identical.

Moreover, the time series shown are very similar to those presented by Lee and Lee (2022). The final time is 12.23 seconds, and the fuel mass depleted is 242.72 kg, very close to the values obtained by the authors of the paper taken as the benchmark: 12.12 s and 247.40 kg. The differences are due to the different optimization methods used.

This concludes the verification of the flight simulator, showing that each module has been correctly implemented and that the entire software is capable of replicating literature results. Also the OCP setup is verified, comparing its results to the solution of the Benchmark paper.

5.11. Model choices verification

This section explains the choices of the models used in RL, visualizing some results. As shown in Table 5.2, the models used in the OCP to replicate the Benchmark Scenario from Lee and Lee (2022) are the simple ones. However, they are not adequate in an operational scenario, missing out on several realistic aspects of the flight that affect the trajectory.

To address this, open-loop simulations are conducted using more complex models from Table 5.2, such as variable MCI, central body gravity, wind, and Earth's rotation, while using controls from the simpler Benchmark scenario.

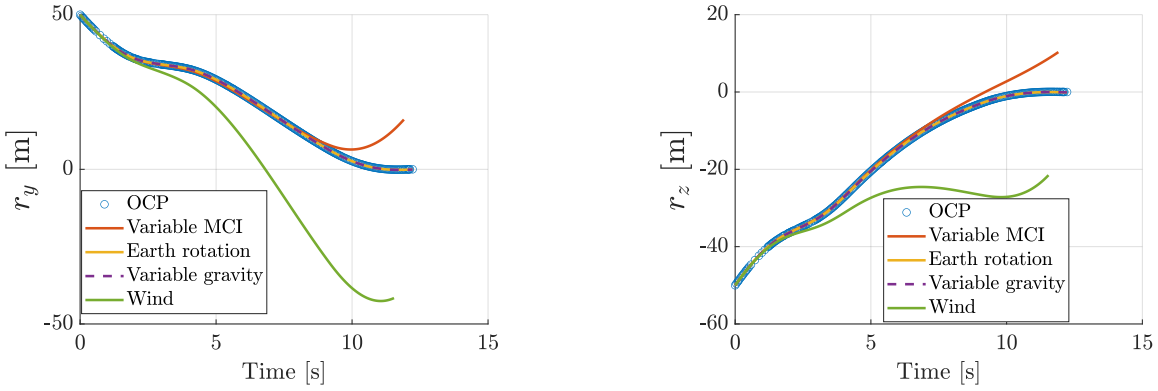
This approach assesses how more realistic models affect the rocket's behavior using optimal controls from simplified models, serving as a proxy for modeling errors of lower fidelity models.

Figures 5.11(a-e) show some state variables from the open-loop simulations using controls from the "Benchmark Scenario", but increasing one-at-a-time the complexity of each model.

The introduction of wind significantly impacts the results, creating lateral forces that cause horizontal drifting, leading to up to 40 m of horizontal terminal position, larger than the 10 m constraint. Neglecting the wind, which is present in real missions, also leads to 8 m/s horizontal velocity and 24 degrees of vertical angle, violating the 1.5 m/s and 3-degree constraints, respectively. The wind changes the angle of attack, modifying the aerodynamic forces and moments applied to the rocket, altering also the rotational motion. Out of the four models considered, wind is the only one that affects roll motion because it directly influences the combination of aerodynamic forces generated by the four fins. Thus, it is fundamental to introduce wind modeling in the RL training to include these realistic effects.

Similarly, running an open-loop simulation with varying CoM and inertia introduces considerable errors with respect to the simulation with constant values. With variable CoM, the pitch and yaw moment arms change throughout the flight, compared to the fixed ones from the Benchmark trajectory. The combination of lever arm and actuator deflections determines the correct steering moment for the vehicle. Therefore, if the actual moment arm differs from the simulated one, the moments generated by the fins and TVC vary from those calculated with a constant moment arm. This discrepancy alters the rotational motion, and consequently, the translational one, leading to up to 10 m/s of horizontal terminal velocity, exceeding the 1.5 m/s constraint. Also, the errors in position and vertical angle are larger than their constraints, up to 15 m and 7 degrees, respectively. This effect is more pronounced towards the end of the flight, where the thrust is maximum. Thus, including a variable MCI model in RL is crucial for accurate results. The influence of larger moments on the deviation from the translational trajectory confirms the need for a 6-DOF formulation.

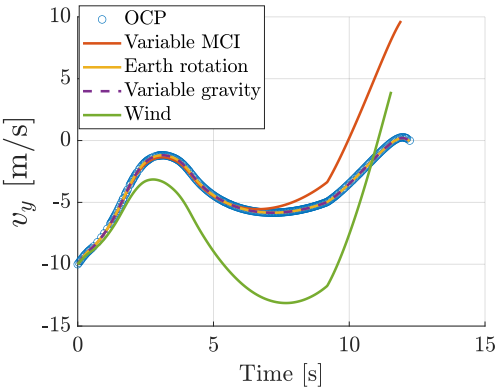
Conversely, the introduction of the Earth's rotation and the central body gravity acceleration are almost negligible. As already anticipated in Section 3.3.2 and Section 3.4.1, the open-loop simulations confirm that the errors introduced by lower fidelity models are expected to have small orders of magnitude, due to the low altitude and velocity of this problem. However, these two more complex models are used in this thesis because they reduce the gap with real-world flights without increasing the runtime, as done by Sagliano et al. (2021a) to test the guidance strategy in conditions as close as possible to those experienced by a real vehicle. This choice also makes the simulator more flexible and adaptable to flights at higher altitudes and velocities, where simpler gravitational and rotation models generate larger errors.



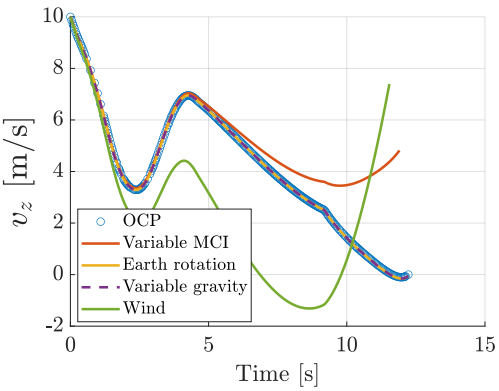
(a) Horizontal Y Position.

(b) Horizontal Z Position.

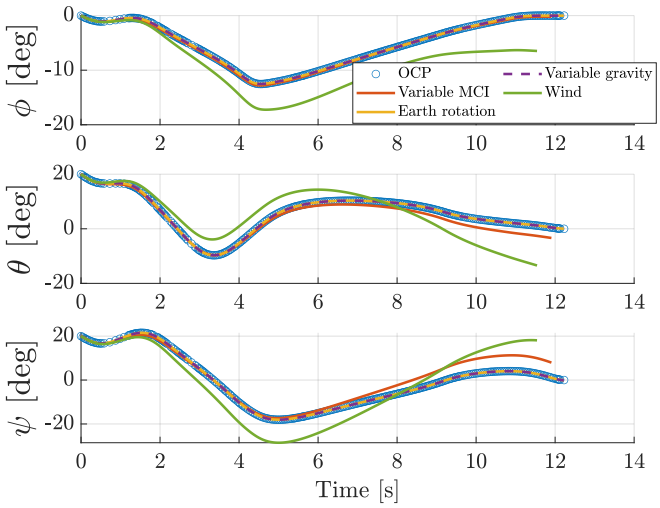
Figure 5.11: Comparison open-loop simulations using higher fidelity models.



(c) Horizontal Y velocity.



(d) Horizontal Z velocity.



(e) Euler angles

Figure 5.11: Comparison open-loop simulations using higher fidelity models. - Concluded

5.12. Simulation scenarios

After having verified the flight simulator and optimal control solution with the simpler scenario used in the Benchmark paper, the fidelity of the simulation is increased by employing more complex models and extended initial conditions. Table 5.2 shows a summary of the two scenarios. The rationale behind the choices of initial conditions in the Complete Scenario was already given in Section 2.5.

The choice of more accurate environment and vehicle models in the Complete Scenario is driven by the desire to reduce modeling errors and the simulation-reality gap. The rationale behind these choices is presented in the error analysis done in Section 3.4, and illustrated in the model choices verification done in Section 5.11. A trade-off is performed for each model, to improve the accuracy of the simulations, without excessively increasing the runtime.

The following chapters use the Complete Scenario, embedding more accurate models to produce realistic guidance and control strategies.

Table 5.2: Model and initial conditions comparison.

Model	**Benchmark Scenario**	**Complete Scenario**
Earth rotation	No	Yes
Atmosphere	US76	US76
Gravity	Constant	Central body
MCI	Constant CoM & Inertia	Variable CoM & Inertia
Wind	No	HM14+turbolence+gust
Initial position [m]	(600, 50, -50)	(2000, 200, -200)
Initial velocity [m/s]	(-60, -10, 10)	(-100, -25, 25)
Initial attitude [deg]	(0, 20, 20)	(0, 20, 20)
Initial angular rate [deg/s]	(0, -10, -10)	(0, -5, -5)

6

Optimal guidance and control

In this chapter, the conventional optimal guidance and control is developed, to be compared with the results from RL. Section 6.1 explains the choice of the optimization software, collocation method, and mathematical formulation of the problem. Section 6.2 illustrates the optimal trajectory in the Complete Scenario, and presents the evaluation of the optimality of the solution. Finally, Section 6.3 describes the LQR controller design process and presents the results of two Monte Carlo closed-loop campaigns, using two different controller architectures.

6.1. Optimal Control Problem setup

6.1.1. ICLOCS2

The software chosen to generate the offline optimal trajectory is ICLOCS2 (Nie et al., 2018). It is an open-source toolbox developed to solve a set of OCPs on MATLAB, providing flexibility and a wide selection of solution methods, facilitating the design and implementation process.

ICLOCS2 offers only a direct transcription method but many options are available discretization methods, NLP solvers, and derivative calculations. For a comprehensive description of the options, the reader may refer to Nie et al. (2018) or to the GitHub page [1]. The transcription method used in this thesis is direct collocation, while the NLP solver is IPOPT, an Interior-Point Optimizer (Wächter and Biegler, 2006). Other solvers like WORHP and `fmincon` are discarded. The first does not provide a MATLAB interface, while the second is slower with a higher chance of failure.

6.1.2. Hermite-Simpson direct collocation method

Direct collocation is a method used to transcribe an infinite-dimensional OCP into a large, sparse NLP problem, which is then solved using an NLP solver. The NLP problem consists in minimizing a cost function, satisfying a set of states and controls constraints at the collocation points (Betts, 2010). Transcription is a process that includes domain discretization, discrete-to-continuous conversion of states and controls, and characterization of differential and integral operators (Sagliano et al., 2017). Hence, the continuous functions are discretized as polynomial splines, transcribing differential dynamic constraints into a set of algebraic constraints (Kelly, 2017). The order of the spline and the distribution of collocation nodes depends on the discretization method used. The constraints, such as the state equations, path constraints, or boundary conditions, are expressed as equalities or inequalities relationships.

The discretization method used is Hermite-Simpson, which is a h-method that uses a large number of trajectory segments and low-order polynomials for interpolation, expressing the states as cubic polynomials, and the controls as piecewise linear functions. As shown in Fig.

[1] `https://github.com/ImperialCollegeLondon/ICLOCS`

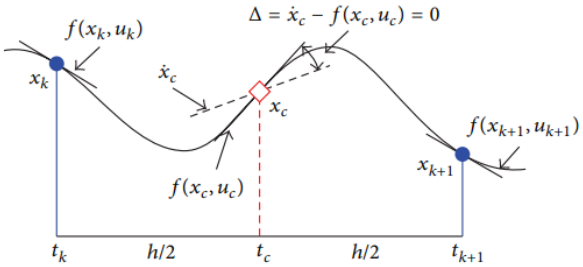


Figure 6.1: Example of Hermite-Simpson collocation method. Taken from Topputo and Zhang (2014).

6.1, in the Hermite-Simpson direct collocation method, the state and control NLP variables (x_k,u_k, x_{k+1}, u_{k+1}), as well as the state time derivatives ($f(x_k, u_k)$,$f(x_{k+1}, u_{k+1})$) are set at the nodes (t_k, t_{k+1}) of each discretization interval. In this way, a third-order polynomial is constructed to satisfy the EOMs only at the nodes. At the midpoints (t_c) of each trajectory segment, the polynomial is interpolated and the difference (Δ) with respect to the dynamics equations is calculated. Minimizing this difference, and the other path constraints, the NLP solver optimizes the selection of states and controls at the nodes (x_k,u_k, x_{k+1}, u_{k+1}) to produce a polynomial that approximates the dynamics within the accuracy of the numerical integration (Topputo and Zhang, 2014).

Tests have also been done with hp-methods such as hpLGR (multiple Legendre-Gauss-Lobatto (LGR) polynomials of various orders) and globalLGR (a single global LGR polynomial of high orders), but the results were worse. The optimization took much longer, due to a very large increase of nodes after a pair of mesh refinement iterations. Based on discretization errors, the software offers the possibility to let the algorithm automatically choose the most suitable discretization method, and it every time chose the Hermite-Simpson one.

6.1.3. Mathematical formulation of OCP

Starting from the initial conditions described in Table 5.2, the purpose of the guidance algorithm is to find the optimal trajectory for pinpoint landing, consuming the least amount of fuel possible. Thus, the goal is to maximize the terminal mass of the vehicle. Expressed in Mayer form, the objective function is:

$$J = \phi[\mathbf{x}(t_f), t_f] = -m(t_f) \tag{6.1}$$

The chosen reference frame is **UEN**. It is remarked that all the quantities are expressed in the correct **UEN** frame, but the reference frame superscripts and the time dependency are dropped for easiness of notation.

After having tried both options, it was found out that using the TVC and fin deflection angles as controls would persistently lead to their rate constraints violation, due to the inability of the software to correctly account for them. Therefore, to enforce the constraints on the TVC and fins rates, they are used directly as control variables. Therefore, the state is augmented by adding the TVC and fin deflection angles as state variables, representing a first-order system. Hence, the state variables are 21:

$$\mathbf{x} = \left[\mathbf{r}, \mathbf{v}, \boldsymbol{\omega}, \mathbf{q}, m, T, \epsilon_T, \psi_T, \beta_{fins,pitch_1}, \beta_{fins,pitch_2}, \beta_{fins,yaw_3}, \beta_{fins,yaw_4}\right] \tag{6.2}$$

The control variables are 7: the thrust magnitude and the TVC and fin deflection rates:

$$\mathbf{u} = \left[u_T, u_{\dot{\epsilon}_T}, u_{\dot{\psi}_T}, u_{\dot{\boldsymbol{\beta}}_{fins,pitch1}}, u_{\dot{\boldsymbol{\beta}}_{fins,pitch2}}, u_{\dot{\boldsymbol{\beta}}_{fins,yaw3}}, u_{\dot{\boldsymbol{\beta}}_{fins,yaw4}}\right] \tag{6.3}$$

At each collocation node, the following set of constraints must be satisfied. The first constraints to be satisfied are those related to the dynamics of the problem.

$$\dot{\mathbf{x}} = \mathbf{f}[\mathbf{x}(t), \mathbf{u}(t)] \tag{6.4}$$

Therefore, the state equations are a set of 21 equality constraints:

$$\begin{aligned}
\dot{\mathbf{r}} =& \mathbf{v} \\
\dot{\mathbf{v}} =& \mathbf{g} + \frac{1}{m}\left[\mathbf{F}_T(\epsilon_T, \psi_T) + \mathbf{F}_{aero,body} + \mathbf{F}_{aero,fins}(\boldsymbol{\beta}_{fins})\right] - 2\boldsymbol{\omega}_{\oplus} \times \mathbf{v} - \boldsymbol{\omega}_{\oplus} \times (\boldsymbol{\omega}_{\oplus} \times \mathbf{r}) \\
\dot{\boldsymbol{\omega}} =& \mathbf{J}^{-1}[\mathbf{M}_{aero,body} + \mathbf{M}_{\mathbf{T}}(\epsilon_T, \psi_T) + \mathbf{M}_{aero,fins}(\boldsymbol{\beta}_{fins}) - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}] \\
\dot{\mathbf{q}} =& \frac{1}{2} Q\boldsymbol{\omega} \\
\dot{m} =& \frac{u_T}{g_0 I_{sp}} \\
\dot{\epsilon}_T =& \mathbf{u}_{\dot{\epsilon}_T} \\
\dot{\psi}_T =& \mathbf{u}_{\dot{\psi}_T} \\
\dot{\boldsymbol{\beta}}_{fins} =& \mathbf{u}_{\dot{\boldsymbol{\beta}}_{fins}}
\end{aligned} \tag{6.5}$$

Moreover, there are three path constraints, one as equality constraint and two as inequality constraints, expressed as:

$$\mathbf{g}_L \leqslant \mathbf{g}[\mathbf{x}(t), \mathbf{u}(t)] \leqslant \mathbf{g}_U \tag{6.6}$$

The first one is to ensure that the norm of quaternions is equal to one.

$$\sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2} = 1 \tag{6.7}$$

The glideslope constraint limits the flight of the vehicle within a prescribed cone with origin on the landing site.

$$\frac{r_x}{\sqrt{r_y^2 + r_z^2}} \geqslant \tan \gamma_{gs} \qquad \gamma_{gs} = 20 \text{ deg} \tag{6.8}$$

The vertical angle, which is defined as the angle between the $X_{\mathcal{B}}$ axis and the X_{UEN} axes, limits the inclination of the vehicle with respect to the landing site frame orientation.

$$\delta_{max} \leqslant \arccos\left(\cos\theta \cos\psi\right) \qquad \delta_{max} = 70 \text{ deg} \tag{6.9}$$

This NLP is a two-point boundary value problem, because there are boundary conditions on state variables at both the initial and final time step, in the form of:

$$\boldsymbol{\psi}[\mathbf{x}(t_0), \mathbf{x}(t_f)] = \mathbf{0} \tag{6.10}$$

The set of equality constraints in Eq. (6.11) represents the state variables' initial boundary conditions, with values taken from Table 2.2. No initial condition is imposed on the controls.

$$\begin{gathered}
r_x(t_0) = r_{x_0} \qquad r_y(t_0) = r_{y_0} \qquad r_z(t_0) = r_{z_0} \\
v_x(t_0) = v_{x_0} \qquad v_y(t_0) = v_{y_0} \qquad v_z(t_0) = v_{z_0} \\
\omega_x(t_0) = \omega_{x_0} \qquad \omega_y(t_0) = \omega_{y_0} \qquad \omega_z(t_0) = \omega_{z_0} \\
q_1(t_0) = q_{1_0} \qquad q_2(t_0) = q_{2_0} \qquad q_3(t_0) = q_{3_0} \qquad q_4(t_0) = q_{4_0} \\
m(t_0) = m_0 \qquad \epsilon_T(t_0) = 0 \qquad \psi_T(t_0) = 0 \\
\beta_{fin,pitch1}(t_0) = 0 \qquad \beta_{fin,pitch2}(t_0) = 0 \\
\beta_{fin,yaw3}(t_0) = 0 \qquad \beta_{fin,yaw4}(t_0) = 0
\end{gathered} \tag{6.11}$$

The set of equality constraints in Eq. (6.12) represents the state variables' final boundary conditions. They all are equality constraints, except for the mass, which is bounded between

the vehicle's dry mass and its initial mass. Furthermore, the "augmented" state variables are not constrained at the terminal time step.

$$\begin{gathered} r_x(t_f) = CoM_x^{\mathbf{UEN}}\,\mathrm{m} \qquad r_y(t_f) = 0\,\mathrm{m} \qquad r_z(t_f) = 0\,\mathrm{m} \\ v_x(t_f) = 0\,\mathrm{m/s} \qquad v_y(t_f) = 0\,\mathrm{m/s} \qquad v_z(t_f) = 0\,\mathrm{m/s} \\ \omega_x(t_f) = 0\,\mathrm{deg/s} \qquad \omega_y(t_f) = 0\,\mathrm{deg/s} \qquad \omega_z(t_f) = 0\,\mathrm{deg/s} \\ q_1(t_f) = 0 \qquad q_2(t_f) = 0 \qquad q_3(t_f) = 0 \qquad q_4(t_f) = 1 \\ m_{dry} < m(t_f) < 4000\,\mathrm{kg} \end{gathered} \tag{6.12}$$

The terminal state for the altitude is chosen such that the simulation terminates when the bottom of the rocket touches the ground.

Some states and all control variables are subject to inequality constraints, bounding them between upper and lower limits throughout the trajectory, based on the requirements.

$$\begin{aligned} -15\,\mathrm{deg/s} \leqslant \omega_x(t) \leqslant 15\,\mathrm{deg/s} \qquad & -15\,\mathrm{deg/s} \leqslant \omega_y(t) \leqslant 15\,\mathrm{deg/s} \\ -15\,\mathrm{deg/s} \leqslant \omega_z(t) \leqslant 15\,\mathrm{deg/s} \qquad & 40\,\mathrm{kN} \leqslant u_T(t) \leqslant 100\,\mathrm{kN} \\ -10\,\mathrm{deg} \leqslant \epsilon_T \leqslant 10\,\mathrm{deg} \qquad & -10\,\mathrm{deg} \leqslant \psi_T \leqslant 10\,\mathrm{deg} \\ -25\,\mathrm{deg} \leqslant \beta_{fin,pitch1}(t) \leqslant 25\,\mathrm{deg} \qquad & -25\,\mathrm{deg} \leqslant \beta_{fin,pitch2}(t) \leqslant 25\,\mathrm{deg} \\ -25\,\mathrm{deg} \leqslant \beta_{fin,yaw3}(t) \leqslant 25\,\mathrm{deg} \qquad & -25\,\mathrm{deg} \leqslant \beta_{fin,yaw4}(t) \leqslant 25\,\mathrm{deg} \\ -15\,\mathrm{deg/s} \leqslant u_{\dot{\epsilon}_T}(t) \leqslant 15\,\mathrm{deg/s} \qquad & -15\,\mathrm{deg/s} \leqslant u_{\dot{\psi}_T}(t) \leqslant 15\,\mathrm{deg/s} \\ -200\,\mathrm{deg/s} \leqslant u_{\dot{\beta}_{fins,pitch1}}(t) \leqslant 200\,\mathrm{deg/s} \qquad & -200\,\mathrm{deg/s} \leqslant u_{\dot{\beta}_{fins,pitch2}}(t) \leqslant 200\,\mathrm{deg/s} \\ -200\,\mathrm{deg/s} \leqslant u_{\dot{\beta}_{fins,yaw3}}(t) \leqslant 200\,\mathrm{deg/s} \qquad & -200\,\mathrm{deg/s} \leqslant u_{\dot{\beta}_{fins,yaw4}}(t) \leqslant 200\,\mathrm{deg/}s \end{aligned} \tag{6.13}$$

6.2. OCP Results

6.2.1. Nominal trajectory

The nominal optimal trajectory for the Complete Scenario is generated without wind and uncertainties. Compared to the Benchmark Scenario, the thrust range is here reduced by about 10% on both boundaries, using [45, 90] kN instead of [40, 100] kN. This provides a control action margin for thrust magnitude in the closed-loop simulations.

To optimize the trajectory, numerical derivatives are chosen over analytical ones due to two main issues. Firstly, computing analytical derivatives requires the use of symbolic toolboxes in MATLAB or Python due to the multiple coupled relationships between state variables. Secondly, errors were found in ICLOCS when handling analytical gradients, Jacobians, and Hessians, even in simpler cases. Unfortunately, they could not be fixed. The use of automatic differentiation tools like ADiGator was ruled out due to compatibility issues.

All variables in the optimization are scaled using min-max normalization, ensuring they are bounded between -0.5 and 0.5. In this way, all the EOMs are in the same order of magnitude, which is beneficial for the solver. The maximum and minimum values for scaling are the user-defined allowable limits for each variable.

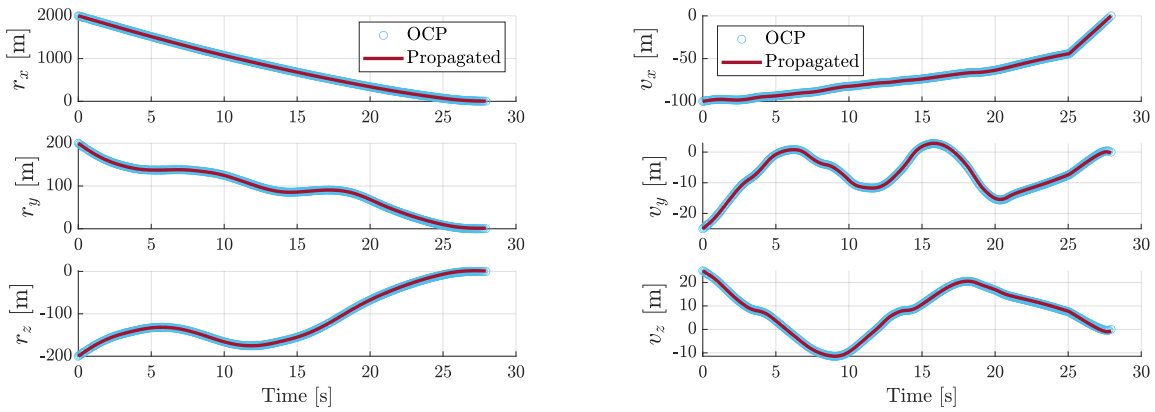
ICLOCS has many tunable parameters and trial and error runs were performed to find the best combination which increases solution accuracy and convergence speed. Some fundamental ones are the number of mesh refinement iterations, the initial number of nodes, and the discretization method. It has been noticed that the optimization problem is very sensitive to the tuning of these parameters, giving different results even for small changes.

The best set of parameters found includes five mesh refinement iterations and 10 uniformly distributed initial nodes. The final number of nodes grew to 447, and the optimization process took 14 hours and 6 minutes. The mesh refinement process adds nodes in intervals where local

errors exceed user-defined tolerances. Since the software does not allow for a direct selection of the number of nodes, this choice is influenced by the two user-defined parameters that regulate the maximum and minimum distance between two nodes. For this optimization, a minimum node distance of 0.0035 s was chosen, without limits on the maximum distance.

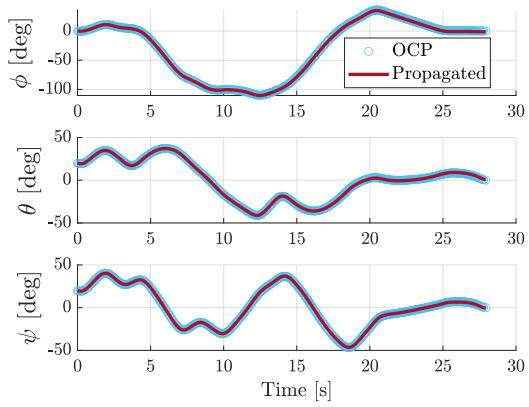
The initial guess for the state variables is a linear interpolation between initial and terminal values, expressed in Eq. (6.11) and Eq. (6.12). The TVC and fin deflections are initially set to zero. The initial guess for the control variables is the following: the thrust magnitude is set constant to its minimum value, while the TVC and fin deflection rates are set to zero.

Figures 6.2(a-d) show the plots for the translational and rotational motion state variable, superimposing the open-loop propagated solution to the discretized OCP one. Since they are coincident, the Hermite-Simpson splines for the states and controls respect the true dynamics not only at the nodes but in the entire trajectory, meaning that the OCP solution is physically feasible. Moreover, terminal and path constraints are respected. For example, 15 deg/s of rotational rates are never exceeded, while the vertical angle is always below the maximum allowed, given that the pitch and yaw peaks are distant in time. More interestingly, it can be observed in Fig. 6.3 that the thrust correctly captures a bang-bang min-max profile, which is known to be the optimal one in undisturbed cases (Lu, 2018). Finally, also the TVC and fin deflections are close to bang-bang profiles, respecting their rate constraints. Given a lack of literature and since they all coordinately affect the rotational motion, it is hard to infer the optimality of these profiles. What is noticed is that both fins in each set have a coordinated deflection, to maximize the effect on their rotational axis.

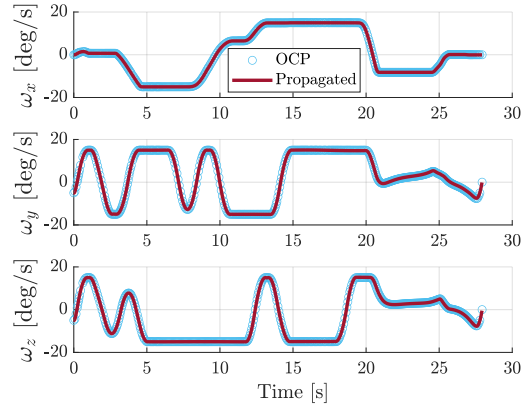


(a) Position.

(b) Velocity.

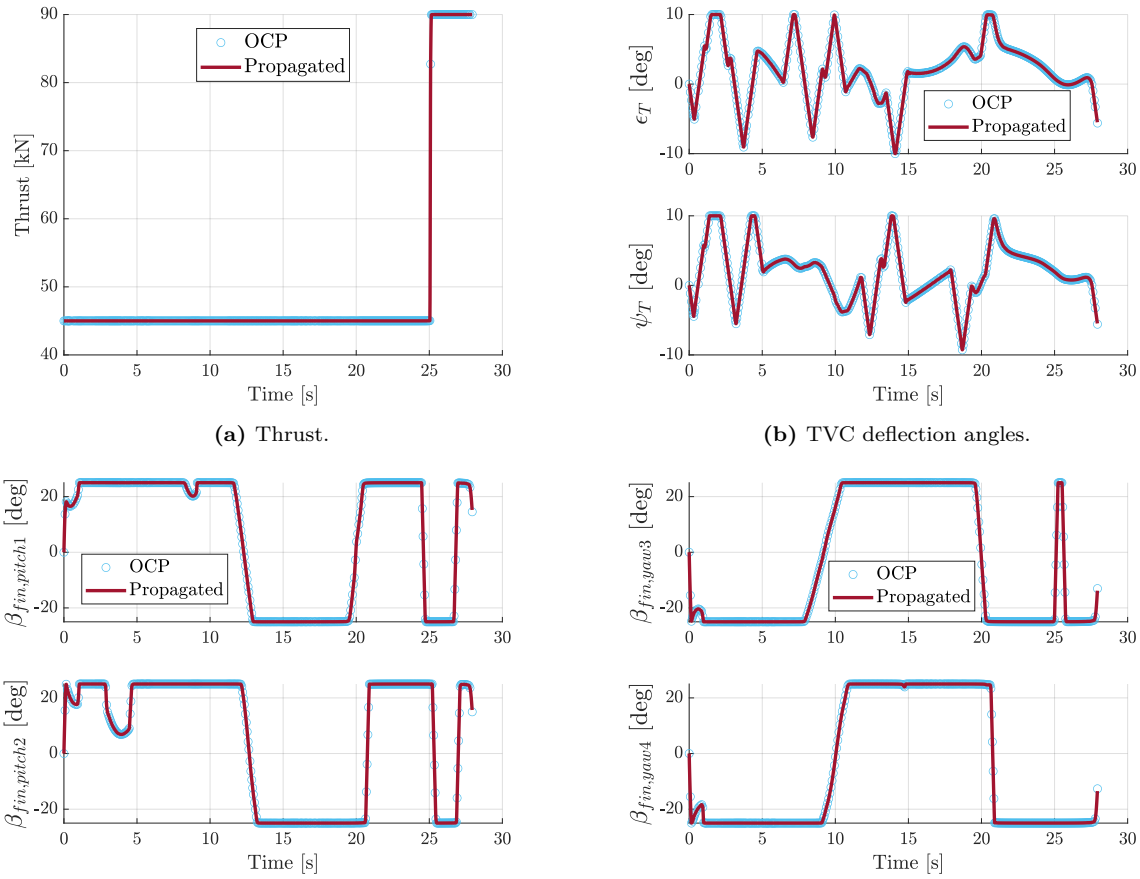


(c) Euler angles.



(d) Angular rates.

Figure 6.2: States nominal trajectory. - Complete Scenario



(a) Thrust.

(b) TVC deflection angles.

(c) Fins pitch pair deflection angles.

(d) Fins yaw pair deflection angles.

Figure 6.3: Controls nominal trajectory. - Complete Scenario

6.2.2. Optimality of the solution

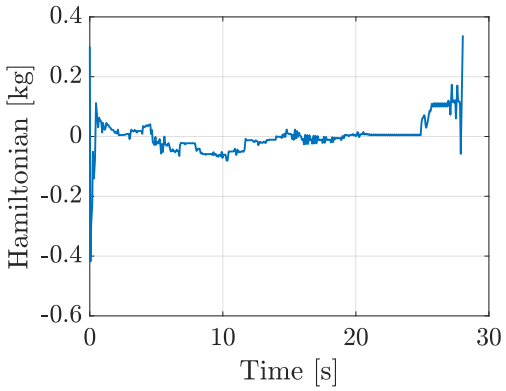
The optimality of the solution is evaluated using the Hamiltonian, which is the product between the costates and the state derivatives. According to the *Pontryagin Maximum Principle*, the solution is optimal when the controls are chosen to minimize the Hamiltonian, from the set of possible controls $\mathcal{U}$ that satisfy the state and control constraints (Bryson, 1979).

$$\mathbf{u} = \arg\min_{\mathbf{u}\in\mathcal{U}} \mathcal{H} \tag{6.14}$$

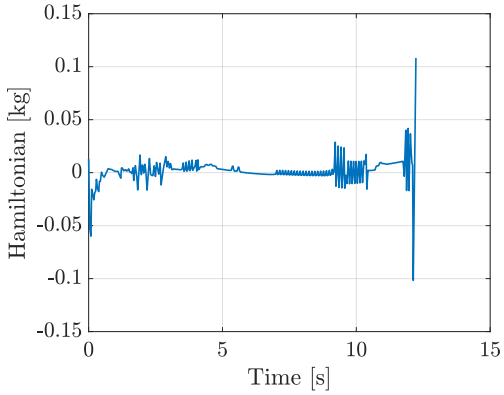
Considering the 6-DOF dynamics from Eq. (6.5), and the absence of a Lagrangian cost term, the corresponding Hamiltonian is:

$$\mathcal{H} = \boldsymbol{\lambda}_\mathbf{r}\dot{\mathbf{r}} + \boldsymbol{\lambda}_\mathbf{v}\dot{\mathbf{v}} + \boldsymbol{\lambda}_{\boldsymbol{\omega}}\dot{\boldsymbol{\omega}} + \boldsymbol{\lambda}_\mathbf{q}\dot{\mathbf{q}} + \boldsymbol{\lambda}_{\boldsymbol{\beta}_{TVC}}\dot{\boldsymbol{\beta}}_{TVC} + \boldsymbol{\lambda}_{\boldsymbol{\beta}_{fins}}\dot{\boldsymbol{\beta}}_{fins} + \lambda_m\dot{m} \tag{6.15}$$

Theoretically, if the generated trajectory is optimal, it would yield a Hamiltonian close to zero for the entire simulation time. The mathematical reason for this is that there is no explicit time dependency in the EOMs and the time of flight is a free optimization variable. Therefore, the total time derivative of the Hamiltonian is equal to its partial derivative with respect to time. Since the derivative of the Hamiltonian with respect to time is zero (Eq. (6.16)), the Hamiltonian is constant. Moreover, at the final time, it is constrained by the partial derivative of the Mayer cost term with respect to the final time (Eq. (6.17)). Given that the final time is unknown and the Mayer term does not explicitly include it, this derivative is zero, implying that the Hamiltonian should be zero and constant everywhere.



(a) Hamiltonian. - Complete Scenario



(b) Hamiltonian. - Benchmark Scenario

Figure 6.4: Hamiltonian for the two scenarios.

$$\frac{d\mathcal{H}}{dt} = \frac{\partial \mathcal{H}}{\partial t} = 0 \tag{6.16}$$

$$\mathcal{H}(t_f) + \frac{\partial \Phi}{\partial t_f} = 0 \tag{6.17}$$

Figures 6.4(a-b) show that, in both cases, it is very close to the expected result. The deviations from the exact zero are due to the numerical tolerance used in the optimization software. By reducing the tolerances on state and control variables, or increasing the number of nodes or mesh refinements, the solution's optimality improves, bringing the Hamiltonian closer to zero. However, smaller tolerances increase computational time, making it challenging to find an optimal trajectory within a reasonable timeframe. Therefore, a trade-off between optimality and computational time is necessary. Eventually, with the chosen tolerances, the optimal trajectories are generated, and the corresponding Hamiltonians, shown in Fig. 6.4(a), are deemed to be acceptable. In fact, the optimal trajectory serves only as a good reference that needs to be tracked when all the perturbations are added.

Additionally, the Hamiltonian error increases in the final flight portion, occurring when thrust magnitude shifts from minimum to maximum. As thrust increases, the optimizer struggles more to enforce the required tolerance, leading to larger errors between the nodes.

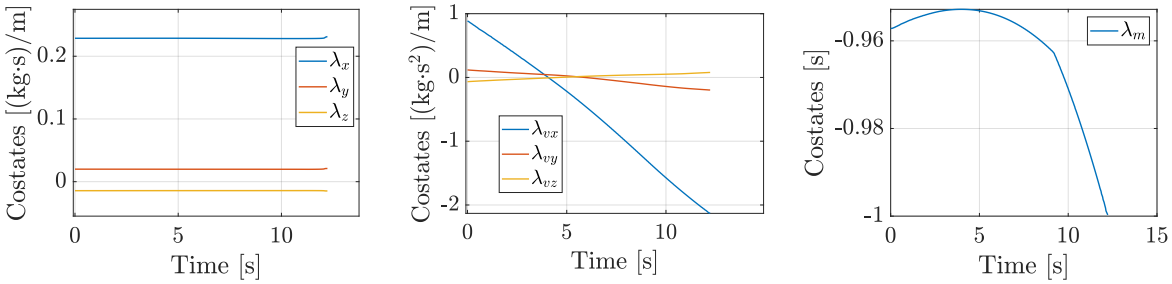
The costate derivatives $\dot{\lambda}_{\mathbf{x}}$ are the derivative of the Hamiltonian with respect to the states $\mathbf{x}$, taken with a negative sign.

$$\dot{\boldsymbol{\lambda}}_{\mathbf{x}} = -\frac{\partial \mathcal{H}}{\partial \mathbf{x}} \tag{6.18}$$

Some observations can be made on some costate vector components, to verify the solution's correctness and optimality. For instance, neglecting higher-order dependencies, such as those in the atmospheric or gravity model, it could be stated that, as a first-order approximation, the Hamiltonian has no explicit dependency on position. Hence, the derivatives of the position costates should be zero, and, consequently, the position costates should be constant. Indeed, Fig. 6.5(a) confirms these predictions. Similarly, velocity costates should be approximately linear, since they are the integral of the position costates, as confirmed in Fig. 6.5(b).

Ultimately, given the *transversality condition* in Eq. (6.19), the final value for the mass costate should be -1. Indeed, the final derivative of the Mayer cost $(-m_f)$ with respect to the mass is -1, as shown in Fig. 6.5(c).

$$\boldsymbol{\lambda}(t_f) = \frac{\partial \Phi}{\partial \mathbf{x}}\Big|_{t=t_f} \tag{6.19}$$



(a) Position costates. **(b)** Velocity costates. **(c)** Mass costate.

Figure 6.5: Example of costates for the Benchmark Scenario.

6.3. Closed-loop simulations

The optimal trajectory is found for a nominal scenario, with given initial conditions, without winds and uncertainties. If open-loop simulations are run, including these effects, they diverge from the nominal trajectory, ending up far from the landing site, with dispersed final states.

Hence, under the assumption of perfect knowledge of the navigation state, an LQR controller is designed to close the GNC loop. The goal is to track the optimal nominal trajectory, removing the errors generated by wind, uncertainties, and dispersed initial conditions.

6.3.1. Linear Quadratic Regulator (LQR)

A LQR is a controller based on the optimal control theory for linear systems, where a linear quadratic cost function (Eq. (6.20)) is minimized, balancing the state errors and control efforts.

$$J = \int_0^\infty (\mathbf{x}\mathbf{Q}\mathbf{x}^T + \mathbf{u}\mathbf{R}\mathbf{u}^T)dt \tag{6.20}$$

Here, $\mathbf{x}\mathbf{Q}\mathbf{x}^T$ represents the state errors, while $\mathbf{u}\mathbf{R}\mathbf{u}^T$ the control efforts. $\mathbf{Q}$ and $\mathbf{R}$ are the weighting matrices that balance how much state deviation and control effort are allowed, directly affecting the controller response.

A method, known as Bryson's rule (Bryson, 1979), can be used to tune $\mathbf{Q}$ and $\mathbf{R}$ matrices for the required performances, using a proportionality between the matrices diagonal entries and the maximum allowable errors on states and controls, as shown in Eq. (6.21). Each i-th diagonal element of the $\mathbf{Q}$ or $\mathbf{R}$ matrix corresponds to the i-th state or control variable.

$$\mathbf{Q} = \text{diag}\left\{\frac{1}{\Delta\mathbf{x}^2_{max}}\right\} \qquad \mathbf{R} = \text{diag}\left\{\frac{1}{\Delta\mathbf{u}^2_{max}}\right\} \tag{6.21}$$

A linearized system is used, expressed as:

$$\boldsymbol{\Delta}\dot{\mathbf{x}} = \mathbf{A}\boldsymbol{\Delta}\mathbf{x} + \mathbf{B}\boldsymbol{\Delta}\mathbf{u} \tag{6.22}$$

$$\boldsymbol{\Delta}\mathbf{y} = \mathbf{C}\boldsymbol{\Delta}\mathbf{x} + \mathbf{D}\boldsymbol{\Delta}\mathbf{u} \tag{6.23}$$

$\boldsymbol{\Delta}\mathbf{x} \in \mathrm{R}^n$ and $\boldsymbol{\Delta}\mathbf{u} \in \mathrm{R}^m$ are vectors representing the deviation from the reference state ($\mathbf{x_0}$) and control ($\mathbf{u_0}$) vectors, respectively. $\mathbf{A} \in \mathrm{R}^{n\times n}$ and $\mathbf{B} \in \mathrm{R}^{n\times m}$ are state and control matrices calculated as the derivatives of the equations of motion, with respect to the state and control vectors, respectively, computed at the reference states and controls. $\boldsymbol{\Delta}\mathbf{y} \in \mathrm{R}^k$ is the deviation output vector, while $\mathbf{C} \in \mathrm{R}^{k\times n}$ and $\mathbf{D} \in \mathrm{R}^{k\times m}$ are the output and the direct transmission matrix, with the latter usually set to zero. Given the state feedback, the control deviation is calculated in Eq. (6.24), while the control gain matrix $\mathbf{K}$ is presented in Eq. (6.25).

$$\boldsymbol{\Delta}\mathbf{u} = -\mathbf{K}\boldsymbol{\Delta}\mathbf{x} \tag{6.24}$$

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}\mathbf{P} \tag{6.25}$$

$\mathbf{P}$ is a positive definite matrix, obtained as solution of the Riccati equation, in Eq. (6.26).

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}\mathbf{P} + \mathbf{Q} = \mathbf{0} \tag{6.26}$$

The Riccati equation is solved by the `lqr` command in MATLAB, which outputs the solution $\mathbf{P}$, the optimal control gains $\mathbf{K}$ and the eigenvalues of the closed-loop system.

6.3.2. LQR controller synthesis process

A LQR controller is chosen due to its design simplicity and widespread use in Multiple Input Multiple Output (MIMO) systems. Unlike other feedback control techniques, such as Proportional, Integral, Derivative (PID) controllers, where gains are obtained by pole placement, LQR gains are obtained through an optimization problem (Mooij, 1998). LQR loses some of its most attractive properties when not all the states of the controlled system are available. For example, in estimation-state based LQR, there might be small perturbations that bring the stability margins to zero. Though more advanced techniques could be employed, for the sake of keeping the design simple, LQR was selected, by assuming perfect states knowledge.

Given the changes in the dynamics, constant feedback gains would not provide sufficiently accurate performances over the entire operational range. For instance, dynamic pressure significantly varies throughout the flight, strongly affecting the control authority of the actuators. Thus, gain scheduling is performed, generating gains for different flight conditions.

The controller architecture consists of two loops. The outer loop adjusts the translational motion providing the trajectory corrective thrust magnitude and attitude actions, while the inner loop controls the rotational motion stabilizing the attitude and angular rates using TVC and fins deflections. This decoupling relies on the fact that if the rotational dynamics is at least one order of magnitude faster than the translational one, frequency separation takes place, and therefore the two loops can be designed separately.

In Fig. 6.6, the controller architecture is detailed. In the outer loop, the measured position and velocity are subtracted from the references provided by the guidance block. These differences are multiplied by the outer loop $\mathbf{K}$ matrix, to produce the necessary delta thrust magnitude and Euler angles attitude, which are summed to their reference values. In the inner loop, errors between reference and measured Euler angles and angular rates are multiplied by the inner $\mathbf{K}$ matrix to compute the delta TVC and fin deflections, which are summed to their reference values. Since LQR cannot satisfy any constraint by definition, a numerical rate limiter and a deflection saturation are applied to the control variables, according to the physical constraints of the actuators. Thrust magnitude, TVC, and fin deflections are the control inputs to the plant, where the EOMs propagate the dynamics to the next time step. The measured states are the output of the plant and are given as feedback to the controller loops.

The process used to design the controller is inspired by the one used in Sagliano et al. (2023), involving the following main steps:

1. First of all, it is chosen to synthesize the controllers at 20 operating points.
2. The altitude is selected as the scheduling parameter because it decreases monotonically, and the rocket is not intended to hover. The 20 operating points are evenly distributed at 100-meter intervals between 1950 m and 50 m.
3. The EOMs are linearized, to obtain a Linear Time Invariant (LTI) system, using numerical linearization, because the interrelationships between the states in different equations and the use of a lookup table for the aerodynamic coefficients, make analytical linearization not practical. The linearization is performed around reference states and controls evaluated along the trajectory at the operating points, using central finite differences.
4. For each operating point, $\mathbf{Q}$ and $\mathbf{R}$ matrices are chosen and the $\mathbf{K}$ matrix is calculated using the MATLAB `lqr.m` routine to solve the Riccati equation. The resulting $\mathbf{K}$ matrices

Table 6.1: Weights for Q matrix.

Error variable	Error value
$\Delta r_{y_{max}}$	0.25 m
$\Delta r_{z_{max}}$	0.25 m
$\Delta v_{x_{max}}$	0.05 m/s
$\Delta v_{y_{max}}$	0.05 m/s
$\Delta v_{z_{max}}$	0.05 m/s
$\Delta \phi_{max}$	0.1 deg
$\Delta \theta_{max}$	0.1 deg
$\Delta \psi_{max}$	0.1 deg
$\Delta \omega_{x_{max}}$	0.01 deg/s
$\Delta \omega_{y_{max}}$	0.01 deg/s
$\Delta \omega_{z_{max}}$	0.01 deg/s

Table 6.2: Weights for R matrix.

Error variable	Error value
ΔT_{max}	1000 N
$\Delta \beta_{TVC,y_{max}}$	0.01 deg
$\Delta \beta_{TVC,z_{max}}$	0.01 deg
$\Delta \beta_{fin,P1_{max}}$	0.1 deg
$\Delta \beta_{fin,P2_{max}}$	0.1 deg
$\Delta \beta_{fin,Y3_{max}}$	0.1 deg
$\Delta \beta_{fin,Y4_{max}}$	0.1 deg
$\Delta \phi_{max}$	0.1 deg
$\Delta \theta_{max}$	0.1 deg
$\Delta \psi_{max}$	0.1 deg

are stored in lookup tables, for interpolation during the simulations.

5. To check the performances, linear analysis of time response in the LTI system is done.
6. Finally, closed-loop non-linear uncertain 6-DOF simulations are run with the controllers, interpolating the **K** matrices based on the altitude.

To achieve the desired performance, the LQR controllers must be carefully tuned. The **Q** and **R** matrices regulate the **K** and are adjusted using Bryson's Rule. The tuning process is iterative, and not straightforward. If the linear or non-linear analysis shows insufficient performance, the **Q** and **R** matrices are adjusted, returning to step 4. If the controllers do not cover the entire trajectory, additional operational points may be added in step 2. The **Q** and **R** matrices act as weights prioritizing certain variables and are refined iteratively by adjusting weights based on the error in key variables until satisfactory results are achieved.

The tuning is done for the nominal trajectory, with linearized dynamics, while perturbations are added later. Once the satisfactory results are achieved, the weights are frozen and the **K** matrices are saved, to be used in closed-loop simulations. Table 6.2, lists states and controls maximum allowable errors used in the diagonal elements of **Q** and **R** matrices.

Gain scheduling is then performed for each channel, linearly interpolating the gains in between two operational points (h_i and h_{i+1}), according to the scheduling parameter (h). The altitude channel, used as scheduling parameter, is removed from the feedback gain matrix.

$$\mathbf{K}(h) = \mathbf{K}(h_i) + \frac{\mathbf{K}(h_{i+1}) - \mathbf{K}(h_i)}{h - h_i} \tag{6.27}$$

The unit step response is used to evaluate controller performance in response to a sudden step input change. It shows how quickly the system reaches the desired output, if there's a steady-state error, and whether the transient response is stable, oscillates, or diverges from the steady-state. The step response helps in tuning the weights of the **Q** and **R** matrices. For instance, if the response is too slow or has large steady-state errors, the **R** matrix is reduced to allow more control effort, or **Q** elements are increased to enforce smaller state errors.

This linear analysis excludes perturbations, using nominal initial conditions and linearized dynamics. It only indicates the tuning quality on a set of states close to the ones where the linearization takes place.

After proper tuning, Monte Carlo campaigns are conducted to assess the controllers' performance on a non-linear 6-DOF scenario, including uncertainties and dispersed initial conditions. The robustness of the G&C policy is tested against external perturbations, which are not embedded neither in the nominal trajectory (guidance), nor in the LQR design (control). However, correctly tuned controllers can compensate for these errors, up to a certain extent.

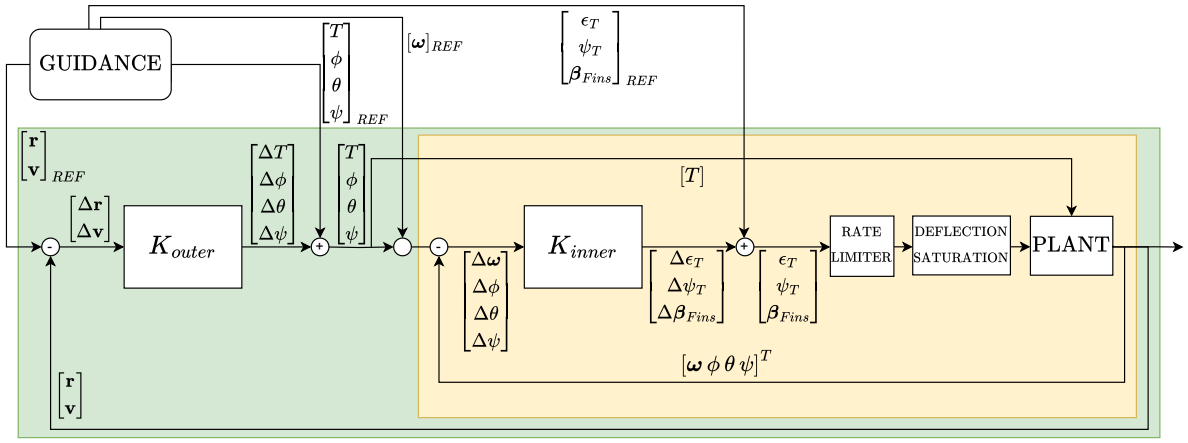


Figure 6.6: Controller architecture.

6.3.3. Results

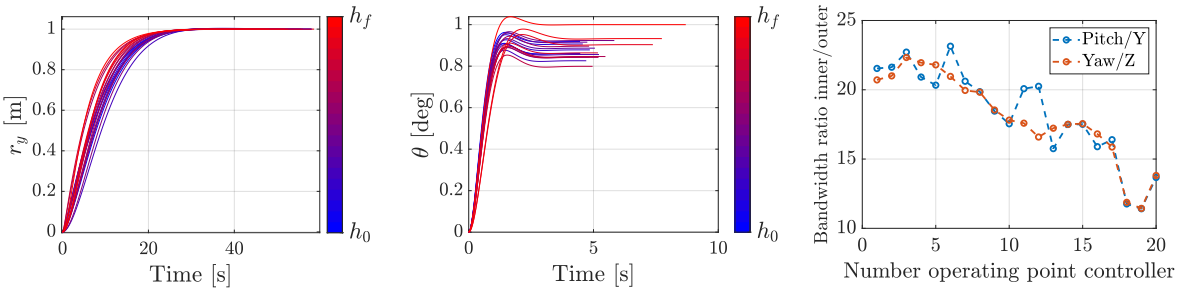
The tuning process of the LQR controllers is lengthy and not trivial. The success of the 6-DOF simulations heavily depends on the chosen weights for the controllers, which are highly sensitive to small changes due to the strong coupling between translational and rotational motion.

For example, in the inner loop, TVC deflection angles control rotational motion by translating attitude and angular rate errors into thrust moments, directing part of the thrust vector perpendicular to the longitudinal axis of the rocket. However, the effectiveness of the deflection in creating moments depends on thrust magnitude, which is controlled by the outer loop. Hence, there is a clear relationship between the two loops, even if their gains are designed independently. For instance, the inner loop TVC gains do not take into account the actual thrust during the closed-loop simulations, but they are pre-calculated to create deflections based on the nominal trajectory, which uses minimum thrust for most of the flight. However, an issue can arise when the thrust magnitude is increased in the outer loop to correct for translational errors. It can induce too large moments that cannot be counteracted by the fins alone, due to their limited surface area and decreasing dynamic pressure, as the rocket approaches the ground. This can potentially make the vehicle unstable, so that it may point with the ogive down, or start to rotate quickly without the possibility to recover from such conditions. Thus, while the two loops work correctly if taken singularly, in reality, they are coupled.

One solution to this issue is to synthesize less aggressive controllers. By allowing larger errors and encouraging smaller control efforts, the thrust magnitude is increased less drastically, and the TVC deflections are more limited. This generates actions that deviate less from the reference ones, avoiding unexpected, destructive moments. However, the drawback is that larger errors propagate during the simulations, more often leading to terminal constraints violation. Finding the correct weight matrices that trade off accuracy on the terminal constraints while avoiding instability is a long and delicate tuning process. A faster outer loop with larger thrust adjustments reduces position and velocity errors but can negatively affect the inner loop. Similarly, a more aggressive inner loop reduces steady-state error in the linear analysis but may create large moments, leading to instabilities in some non-linear simulations.

Hence, it is accepted to have larger terminal errors and a steady state error in the inner loop, as a compromise to prevent instabilities in the non-linear analysis. In the 6-DOF perturbed Monte Carlo campaigns these issues are exacerbated especially due to initial dispersion, which introduces significant initial errors that the controller tries to compensate for.

The step responses of two state variables for the inner and outer loop are presented in Fig. 6.7(a,b). Each line represents the controller at 20 different operating points, from an altitude of 1950 m (h_0) to 50 m (h_f). It stands out that the outer loop is slower, but it does



(a) Step response r_y. **(b)** Step response θ. **(c)** Bandwidth ratio inner/outer loop.

Figure 6.7: Linear analysis LQR tuning.

not present a relevant steady-state error. Conversely, the faster inner loop has a steady-state error, reduced as much as possible to increase the ability to fulfill terminal constraints without causing instabilities. Figure 6.7(c) confirms the frequency separation, with the inner loop's bandwidth being one order of magnitude larger than the outer one, indicating that rotational motion is faster than translational motion, as it should be in a two-loop controller architecture.

Figures 6.8 show the states and controls for the Monte Carlo closed-loop campaigns, including wind, uncertainties, and initial dispersions as presented in Table 2.2. The controller runs at 10 Hz, commanding updates every 0.1 s, while the dynamic propagation is one order of magnitude faster, with a time step of 0.01 s. The nominal trajectory tracked is the one presented in Fig. 6.2, while the controllers are synthesized using the $\mathbf{Q}$ and $\mathbf{R}$ matrices shown in Table 6.2. Out of 1000 runs presented, they all manage to track the nominal optimal trajectory to a certain extent, with no one that diverges significantly. This is remarkable since LQR is not an inherently robust control technique because it does not account for model uncertainties and disturbances.

Terminal attitude (Fig. 6.8(g-i)) and vertical velocity (Fig. 6.8(d)) consistently remain within prescribed constraints, while position, horizontal velocity, and angular rates are the most commonly exceeded terminal values.

The final position dispersion has a bias, most likely due to the influence of the wind, because the E-W wind (y-axis) component is almost always negative, blowing up to 15 m/s during the entire flight (Fig. 5.6a). As it can be seen in Fig. 6.8(b), all trajectories deviate from the nominal one, resulting in terminal position values shifted by several meters (Fig. 6.10(a)). This effect is less pronounced on the z-axis, since the N-S wind is weaker (Fig. 6.8(c)).

When the outer controller compensates for the position error, it influences the horizontal velocity. Figures 6.8(b,e) show that close to the end of the flight, the y-velocity component increases positively to correct the negative y-position. However, since this component is already slightly positive at the end of the nominal trajectory, this effect is enhanced, violating the terminal y-velocity constraint in a significant number of cases (Fig. 6.10(c), Table 6.4).

The balance between the last two mentioned effects depends on the tuning of the controllers. What has been noticed is that giving too high gains to the horizontal velocities can bring unrecoverable instabilities in the first half of the flight, due to the too large moment induced by a significant thrust magnitude increase. Thus, these gains are kept low.

Figures 6.8(j-l) show local violations of the 15 deg/s angular rate constraint during flight. While the nominal trajectory enforced this in the mathematical formulation, LQR does not account for constraints. Hence, if the controller commands corrections that significantly increase angular rates, nothing can be done to limit them. Since this is not a hard constraint, as long as the vehicle can handle these large rates without diverging, they are accepted.

The angular rates terminal constraint violations are caused by the shape of the nominal trajectory. As shown in Fig. 6.8(k,l), the nominal angular rates y- and z-components have a significant curvature in the very terminal phase. This sudden change takes place because the

rocket arrives inclined from a horizontally displaced position, and brings its vertical angle to zero only in the final seconds. In a rapid sequence, the vehicle first decreases the angular rates from 0 to -10 deg/s to go toward a vertical attitude and then reduces them to 0 deg/s to meet the angular rates constraints, keeping the vehicle vertically. If there is even a slight error of 1-2 m/s in tracking the velocity, the rocket does not have time to drive the angular rates to zero, landing with a few negative degrees per second of residual values. In these cases, the vehicle could only meet the terminal constraints on angular rates and vertical velocity if it could fly a few more centimeters below ground, which is impossible. Moreover, given the larger vertical velocity, the last control update before touchdown is done at a slightly higher altitude, giving reference angular rates larger than zero. The combination of worst angular rates tracking, due to a less aggressive controller, and larger landing velocity leads to the constraint violation.

This hypothesis becomes clear when looking at Fig. 6.10(b): there is a batch of results, using the controller architecture explained in Section 6.3.4, that have terminal rates around the target. Those are simulations where the vertical velocity is zero at a few centimeters above ground, giving the vehicle enough time to correct the angular rates.

Thrust magnitude, TVC, and fin deflection profiles, presented in Fig. 6.8(m-p), show that the controllers are not very aggressive. Deviations from the nominal controls are concentrated in the first part of the flight, where the state errors are large due to the very dispersed initial conditions. For example, the initial thrust is almost always commanded to the maximum, even if the reference would be at the minimum value. Similar trends are present for TVC and fins. Additionally, some corrections are present mid-flight, where wind and uncertainties play a significant role. Finally, while the inner loop controls show minor deviations at the end of the flight, the thrust is significantly changed to counteract position and velocity errors.

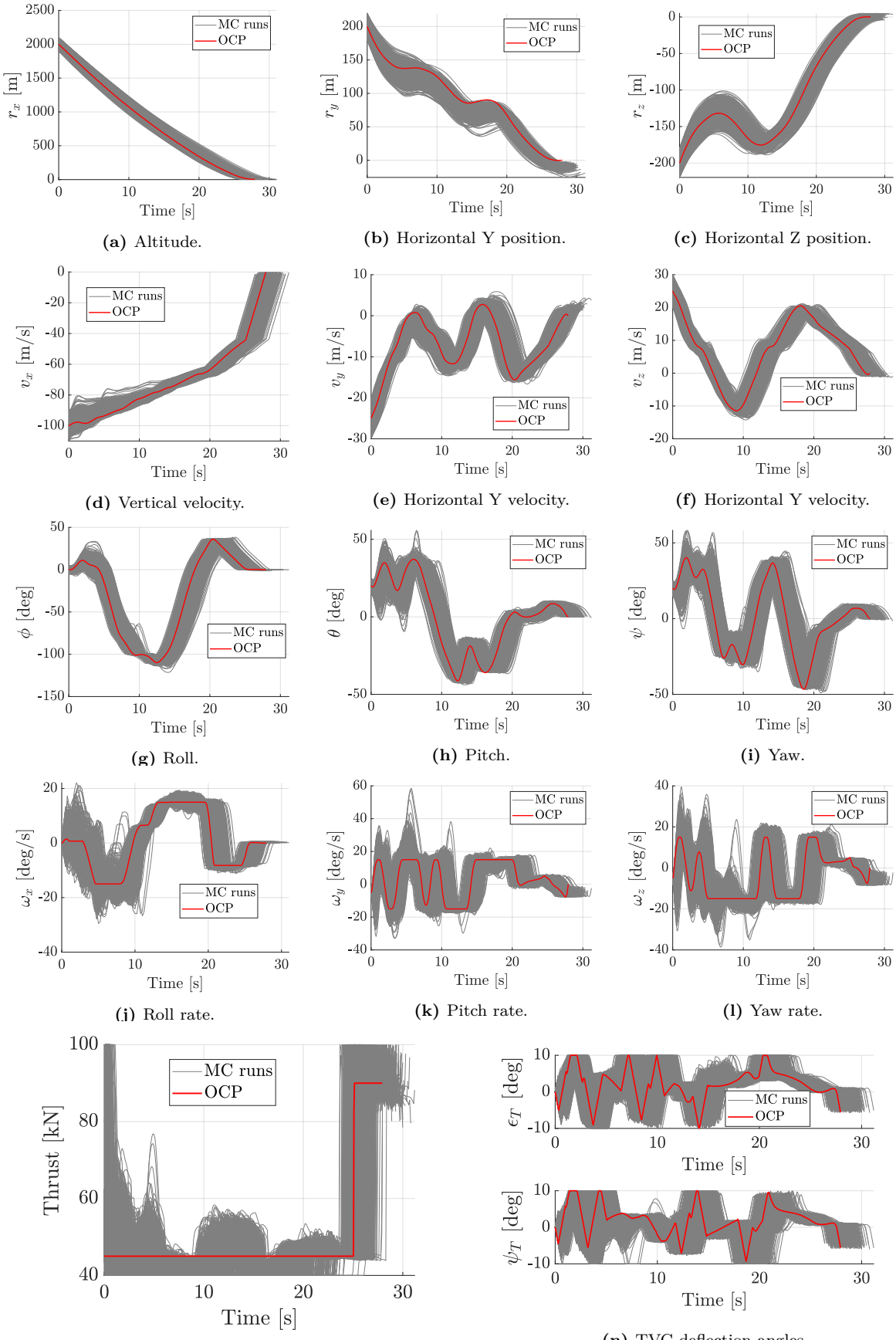
Given the good terminal results, it also means that the extensive tuning procedure has been performed decently, since with LQR there is no way to impose constraints a priori.

In any case, the coupling between guidance (outer loop) and inner loop (control) clearly shows that improvements can be made to the architecture. More sophisticated control techniques could be introduced, accounting for this coupling, but going beyond an LQR controller is outside of the scope of this thesis. Hence, modifications to the current architecture are done.

6.3.4. Addition of RCS

As first change, TVC deflection angles are moved to the outer loop, addressing the issue of thrust moments generated by controls from both loops, originally designed to be independent. In this new architecture, the feedback gains corresponding to the TVC deflections are synthesized while intrinsically accounting for the thrust magnitude. With this change, the outer loop controls the translational motion of the vehicle generating forces along the body axes, thanks to the combination of thrust magnitude and TVC deflections.

However, a new issue arises: the thrust vector in the outer loop not only generates forces for translational motion but also induces moments on the pitch and yaw axes, impacting the inner loop. Ideally, with sufficient frequency separation, the inner loop actuators quickly compensate and maintain the desired attitude and angular rates. However, the inner loop's only actuators are the fins, which lack sufficient control authority due to their small size and low dynamic pressure. Figure 6.9(a) shows that the peak moment generated by the fins is generally lower than the moment generated by thrust. The fins reach their peak effectiveness at around 20 seconds when the velocity is about 60 m/s and their local aerodynamic angles are maximized. While thrust-generated moments are unaffected by flight conditions, the ones created by the fins depend on the vehicle's attitude and dynamic pressure. After 27 seconds, as the velocity drops below 20 m/s, the fins' effectiveness sharply decreases. Control profiles in Fig. 6.3(g-h) indicate that the fins are nearly always at maximum deflection, but the dynamic pressure is too low to generate significant moments (Fig. 6.9(a)).



(a) Altitude.

(b) Horizontal Y position.

(c) Horizontal Z position.

(d) Vertical velocity.

(e) Horizontal Y velocity.

(f) Horizontal Y velocity.

(g) Roll.

(h) Pitch.

(i) Yaw.

(j) Roll rate.

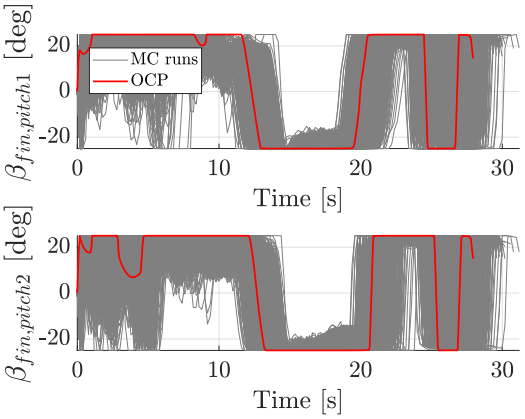
(k) Pitch rate.

(l) Yaw rate.

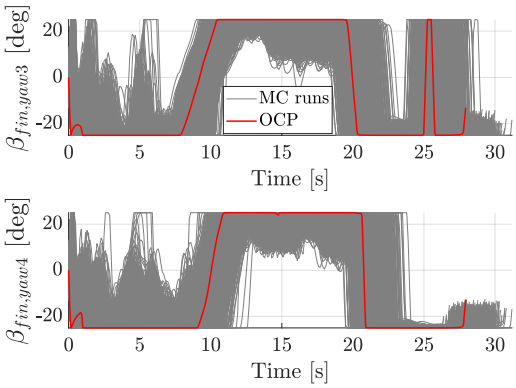
(m) Thrust.

(n) TVC deflection angles.

Figure 6.8: Closed-loop 1000 Monte Carlo simulations results.

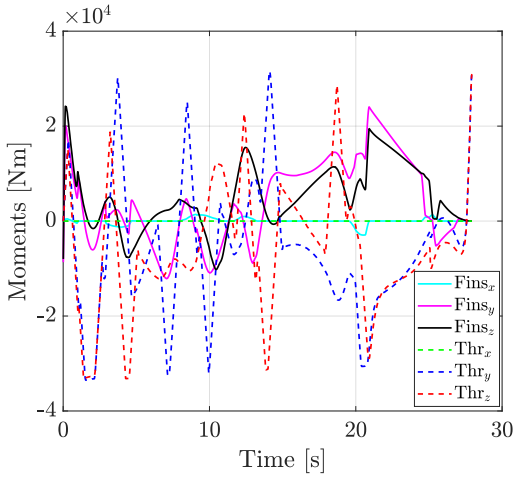


(o) Fins pitch pair deflection angles.

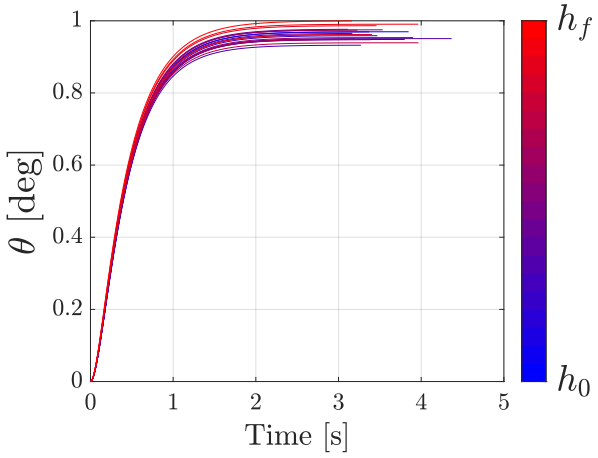


(p) Fins yaw pair deflection angles.

Figure 6.8: Closed-loop 1000 Monte Carlo simulations results. - Concluded



(a) Moments generated by fins and TVC in nominal trajectory.



(b) Step response θ RCS architecture.

Figure 6.9: Moments generation analysis and step response second LQR architecture.

Therefore, fins alone do not provide enough control authority to control the inner loop, and additional actuators have to be introduced.

Other than aerodynamic surfaces, a very common set of actuators to control the attitude of a vehicle during terminal landing, as well as in space, is the RCS. It can be made of cold gas thrusters, mono-propellant, or bi-propellant hypergolic thrusters. For the Falcon 9 first stage, the RCS consists of eight nitrogen cold gas thrusters arranged in four pairs radially around the rocket, similar to the placement of the fins (SpaceX, 2021). Each pair has two thrusters fixed in opposite orientations at each location. These thrusters are used during the exoatmospheric phase as the primary control mean, while in the terminal landing they assist aerodynamic fins in controlling attitude and angular rates at low dynamic pressure.

In the first design iteration, the RCS is added to the inner loop and it is modeled to provide control moments around each body axis, while the generated forces are neglected in the translational equations due to their smaller magnitude compared to the thrust force. The allowable torque generated by the RCS is preliminarily limited to 9000 Nm on the pitch and yaw axes. With the booster measuring 15 m in length and the center of mass positioned about 3.5-4.5 m from the bottom, placing the thrusters near the top creates a moment arm of approximately 10 meters. The available thrust magnitude depends on the type of propellant:

Draco bi-propellant hypergolic engines used on the Dragon capsule produce up to 400 N, while the upgraded SuperDraco reaches 73 kN (SpaceX, a). Ariane V used a mono-propellant hydrazine engine that generates 420 N for attitude control (ArianeGroup, 2020). The Space Shuttle used hypergolic bi-propellant engines to control attitude during separation, on-orbit operations, and re-entry, with 38 primary engines generating 3869 N each, and 6 Vernier engines producing 111 N each (Blevins and Hohmann, 1975).

While Space Shuttle, Dragon, and Ariane V are different-sized vehicles with different RCS missions, Falcon 9 is more similar to the reference vehicle in this thesis. These other vehicles primarily use RCS when the fuel mass is still significant, whereas, during terminal landing, Falcon 9's fuel mass is close to the dry mass of the first stage, reducing the torque needed for the same angular acceleration. Considering that SpaceX preferred nitrogen-based cold gas thrusters over the already developed hypergolic Draco engines for reliability, hazard avoidance, and rapid reusability, it is reasonable to assume a maximum thrust of 450 N per thruster, close to the Draco thrusters' output. It is hard to find any source of reliable information to confirm this hypothesis. This assumption also ensures that the generated force is negligible in the translational motion, so the 9000 Nm torque limit on the pitch and yaw axes is achieved by the 10-meter lever arm and the 450 N thrust from two thrusters directed along the same axis. Given a rocket radius of 0.75 m, the moment is limited to 1350 Nm on the roll axis.

The RCS torque limit poses a ceiling on the inner loop's control authority. Also in this architecture, if the outer loop controller is too aggressive in correcting position and velocity errors, it could generate excessive moments that the inner loop actuators (fins and RCS) cannot counteract, potentially destabilizing the rocket. In a real space vehicle design, once encountering such limitations, a new design iteration would take place, possibly involving different RCS actuators or increasing their distance from the center of mass. However, iterating on the vehicle design is beyond the scope of this thesis.

Therefore, a careful tuning of the $\mathbf{Q}$ and $\mathbf{R}$ matrices is performed. In this architecture, the controllers are more aggressive than the previous one, but still with caution to avoid unstable or divergent simulations. However, as improvement, the steady-state errors present in the inner loop of the previous architecture are almost eliminated, as shown in Fig. 6.9(b).

Figures 6.10(a-f) show the terminal state dispersions with the current controller architecture, compared to the Monte Carlo campaign using the previous version without RCS.

The inclusion of RCS allows the inner loop to control the rocket's orientation more quickly, minimizing the impact of the thrust vector's torques in the outer loop. This enables the outer loop to be more aggressive in correcting position and velocity errors without destabilizing the rotational motion. Moreover, moving the TVC to the outer loop allows the outer controller to use the entire thrust force for more direct and effective corrections.

The separation between the two loops is particularly beneficial in the final seconds of flight when the vehicle's velocity drops, and the coupling between translational and rotational motion increases. During this phase, fins have minimal control authority due to low dynamic pressure, making the thrust vector influence both motions. Using only TVC, both loops must be less

Table 6.3: Statistic of Monte Carlo campaigns using different LQR architectures.

Training	**Norm Horizontal Position [m]**		**Norm Horizontal Velocity [m/s]**		**Vertical Velocity [m/s]**		**Norm Angular rates [deg/s]**		**Vertical angle [deg]**		**Mass Consumption [kg]**	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
TVC+fins	6.94	2.18	1.45	0.40	-0.53	0.61	3.73	1.73	0.55	0.25	502.3	11.48
RCS+fins	3.55	1.04	1.39	0.35	-0.59	0.68	2.44	2.11	0.28	0.17	500.5	11.77

Table 6.4: Success rates and failure causes

Training	PSR (out of 1000)	Altitude	Horizontal position	Horizontal velocity	Vertical angle	Angular rates
TVC+fins	390	0	76	347	0	440
RCS+fins	570	0	1	212	0	310

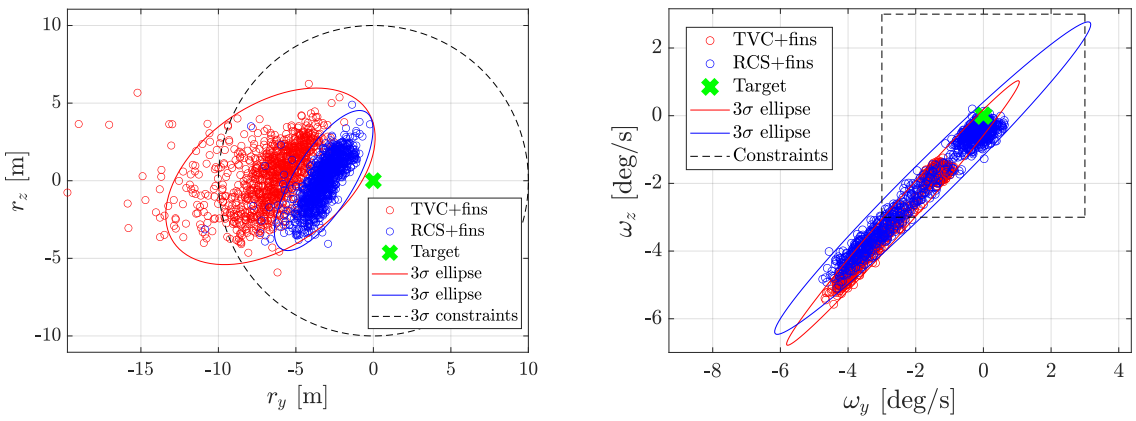
aggressive to prevent instabilities from interacting TVC deflections and thrust magnitude. However, with additional actuators to decouple the motions, the interactions are reduced, allowing the outer loop to correct for wind-induced position errors more strongly, while the RCS in the inner loop maintains a stable attitude, leading to more effective final corrections.

In this second scenario, the position constraint is always respected (except for one run), with a lower mean value (Table 6.3). Figure 6.10(e) shows that the vertical velocity is always tracked within its terminal constraint limit, with similar mean values in both cases. Figure 6.10(c) illustrates that the horizontal velocity constraint is sometimes violated, for the same reasons as explained for the first LQR architecture. However, the mean terminal value is slightly lower and fewer runs exceed this constraint, as shown in Table 6.3 and 6.4.

A larger separation between the inner and outer loop also helps the terminal accuracy of the rotational motion. Figure 6.10(d) and Table 6.4 show that the vertical angle is always below the threshold, while Table 6.3 reveals that the mean is reduced when the RCS is included.

The angular rates are also improved by adding the RCS, even if they do not fulfill completely the terminal constraint, for the same reason as explained for the first architecture. However, Fig. 6.10(c) and Table 6.4 indicate that fewer runs violate it. From Fig. 6.10(h) and Table 6.3, it can be seen that the inner loop better tracks the angular rates, when the RCS is used. As the terminal velocity increases, the norm of angular rates gets larger. Indeed, when the rocket lands with zero vertical velocity, at a few centimeters of altitude (Fig. 6.10(f)), the norm of angular rates is almost zero. This confirms the correlation previously hypothesized between terminal vertical velocity errors and angular rates' tracking.

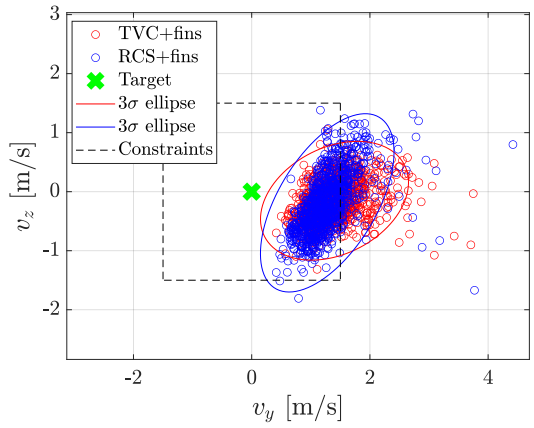
To conclude, it can be said that adding the RCS in the inner loop and moving the TVC to the outer loop helps in decoupling the translational and rotational motion. As a consequence, the controllers can be more aggressive, since less strong interactions are present between the states, and the performances of the closed loop simulations improve. The number of successful runs respecting the constraints is increased from 390 to 570, as shown in Table 6.4.



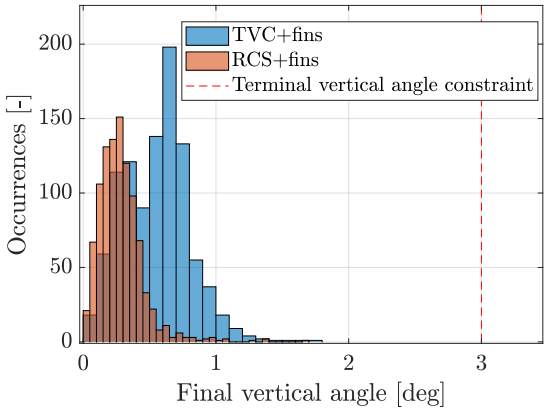
(a) Terminal horizontal position

(b) Terminal angular rates

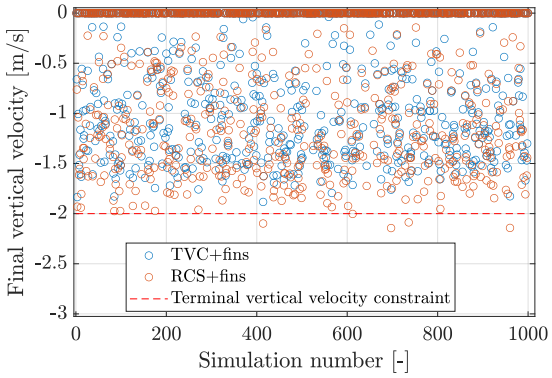
Figure 6.10: Comparison terminal constraints between the two sets of Monte Carlo simulations.



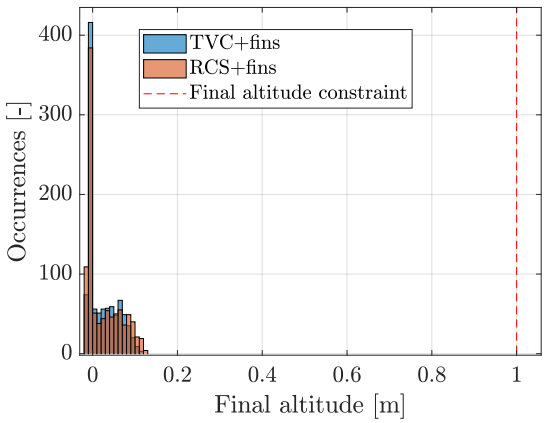
(c) Terminal horizontal velocity



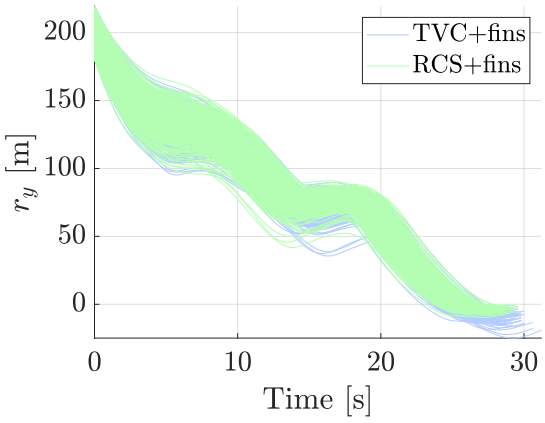
(d) Terminal vertical angle



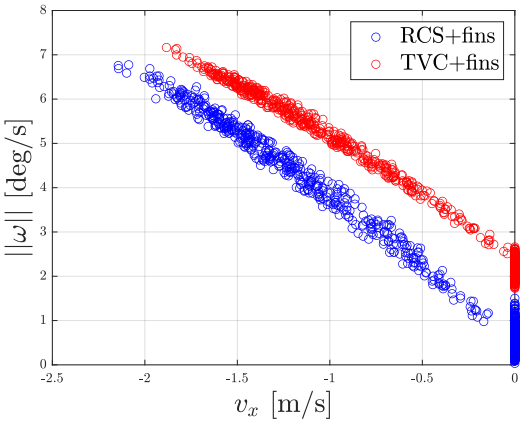
(e) Terminal vertical velocity



(f) Terminal final altitude



(g) Comparison r_y



(h) Relationship $\|\omega\|$-v_x

Figure 6.10: Comparison terminal constraints between the two sets of Monte Carlo simulations. - Concluded

7

Meta-Reinforcement Learning Guidance

This chapter starts with an explanation of the development and implementation of the meta-RL guidance policy in Section 7.1. Section 7.2 presents the reward function and how it is created. Section 7.3 illustrates all the results, starting from Section 7.3.1 with the baseline results on which the following sections are built. Section 7.3.2 assesses the sensitivity to the network's hyperparameters, while Section 7.3.3 compares GTrXL results with LSTM. The robustness of the policy to navigation and control errors is assessed in Section 7.3.4, while Section 7.3.5 presents the results adding RCS as controls. Furthermore, additional trainings are performed removing some inputs to the network in Section 7.3.6, while Section 7.3.7 describes a patch to improve the terminal vertical velocity. Finally, Section 7.3.8 illustrates the best training, including a thrust rate constraint to make the policy more realistic. Section 7.4 presents a preliminary budget analysis of the power drained by the actuators. Section 7.5 shows the Lyapunov numerical method used to assess the stability of the policy.

7.1. Meta-RL guidance algorithm

Development and implementation

The guidance system employs meta-reinforcement learning to determine the optimal closed-loop guidance strategy for landing a space vehicle. This method allows the agent to use its past experiences to select the best control action at each time step. The policy is represented as a Probability Distribution Function (PDF), based on the agent's current state. After being trained, the outputs during the deployment are the expected value of that PDF.

As for the conventional G&C strategy, the control vector output of the policy is composed of 7 elements: thrust magnitude, two TVC deflection rates, and four fin deflection rates:

$$\mathbf{u} = \left[T, \dot{\epsilon}_T, \dot{\psi}_T, \dot{\beta}_{fins,pitch_1}, \dot{\beta}_{fins,pitch_2}, \dot{\beta}_{fins,yaw_3}, \dot{\beta}_{fins,yaw_4}\right] \tag{7.1}$$

The choice of the rates instead of the deflection angles is driven by the necessity of enforcing constraints on both the rates and the deflection angles.

The inputs to the neural network are all the observations $\mathbf{x}$ available from the dynamics. If the states are not perturbed, the observations are coincident with the states.

$$\begin{aligned}\mathbf{x} =&[r_x, r_y, r_z, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, q_1, q_2, q_3, q_4,\\ &\epsilon_T, \psi_T, \beta_{fins,pitch_1}, \beta_{fins,pitch_2}, \beta_{fins,yaw_3}, \beta_{fins,yaw_4}, m, t]\end{aligned} \tag{7.2}$$

The guidance system uses meta-RL to train a NN, approximating the policy that relates states and actions. Recurrent or Transformer NNs introduce hidden states or attention, enabling the agent to consider both the current and previous states when making decisions.

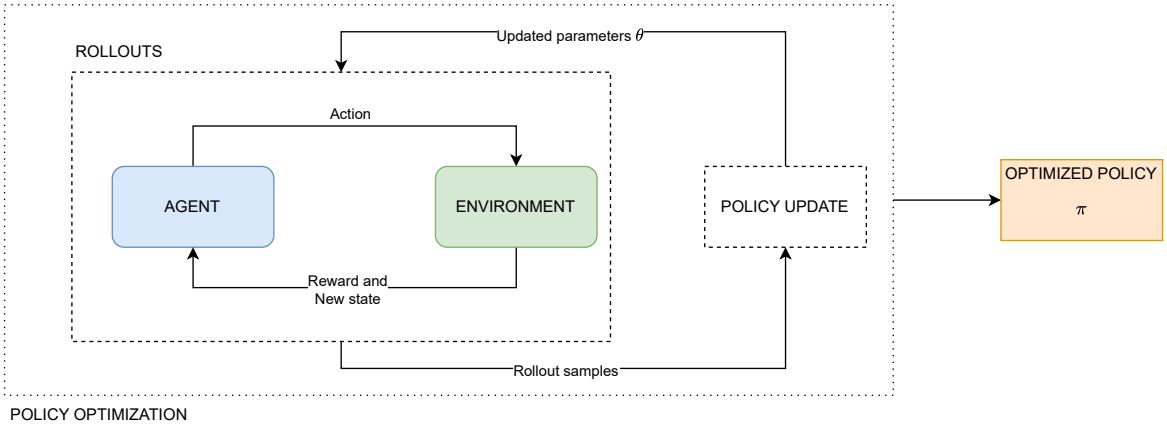


Figure 7.1: Training overview of the guidance system.

Training in uncertain environments allows the network to generalize its policy across different scenarios, also not encountered during training, enhancing the robustness of the algorithm.

The goal is to develop a guidance law that ensures precise pinpoint landing, aiming for efficient fuel consumption, using thrust vectoring and fin deflections. The fins reduce the TVC effort, conserving fuel, but they are effective only at high dynamic pressure.

A reward function, comprising a component (*shaping reward*) at each time step and a component (*terminal reward*) at the episode's end, is designed to guide the agent in learning the optimal policy while adhering to path and terminal constraints.

In Fig. 7.1, a schematic of the guidance system's training is presented. For every training iteration, using its current policy, the agent interacts with the environment, taking actions and receiving a reward at each time step. Based on the goodness of the actions performed in specific states, its network is updated, before interacting again with the environment, using the updated policy. After several iterations, the optimization terminates yielding the final policy.

The first step in developing a G&C policy using RL is to create a framework for training the NN. Implemented in Python, it uses open-source libraries. The implementation consists of three main components: the environment, the neural network, and the training algorithm.

The environment, which is the core of the simulation, is a custom setup based on OpenAI's `Gym` environment class. It includes the dynamic EOMs (Eq. (6.5)) and it manages the entire problem. At the start of each simulation (more often called *episode*), initial conditions and perturbations are sampled, and a wind profile is generated. At each time step, the `step()` method samples control actions based on current observations through a network forward propagation, propagates the states using the dynamics, and updates the observations. It also computes the reward function. This process is repeated until a termination condition is met.

NNs are built using `TensorFlow` and can range from simple fully connected networks to more complex architectures like Transformers and LSTMs. These architectures can be customized to meet specific needs and combined to enhance certain data patterns.

Since the goal of this thesis is not to develop new methods but to apply RL techniques to a space guidance problem, already available training algorithms are used. Moreover, implementing new algorithms could be very tricky and time-consuming, due to the necessary thorough testing needed. Multiple reliable libraries recognized as state-of-the-art, such as `stable-baseline` or `Ray-RLlib`, are available. `Ray-RLlib` is chosen, due to its ability to train custom networks using attention layers and recurrent layers with PPO. It is an industry-graded open-source library, which interfaces with a custom environment to train a network, handling the data management (Liang et al., 2018). It offers a highly parallelized framework, automatically distributing the tasks to the number of CPUs chosen by the user. Therefore, it can be

employed on clusters of CPUs, to speed up the training procedures. It is easy to use and it provides a high degree of personalization of parameters, algorithms, and networks.

Using `pyrlprob`, an open-source Python library that facilitates the training, evaluation, and post-processing of `Gym`-based environments using `Ray-RLlib`, custom configuration files are created to set the network's hyperparameters and the flight dynamics variables needed for the training and evaluation processes.

To ensure consistent magnitudes across state variables, the entire problem is normalized, selecting a minimal set of reference units and using them to normalize every variable. The normalization is done to facilitate the NN training, which is improved if it handles values close to the unit. Similarly to what is done by Sagliano et al. (2021a), the chosen reference variables are: nominal initial altitude (2000 m), nominal initial mass (4000 kg), gravitational acceleration (9.80665 m/s), ambient temperature (288.15 K) and pressure (101325 Pa). These last two are used only in the calculation of the normalized density and temperature profiles. Manipulating the first three variables, reference values for time, velocity, and acceleration are found. Finally, each state variable, vehicle, or environment parameter is divided by the reference values corresponding to its unit of measure.

Training a neural network to learn an optimal policy involves running numerous episodes over several iterations. In each iteration, new simulations are initialized with uncertainties and initial conditions sampled from distributions. During each episode, actions are performed based on the current network policy, and the EOMs are propagated. Initially, the NN parameters are random, leading to a random policy with unrealistic behavior, such as the vehicle rotating with extremely unrealistic angular rates and attitude. As training progresses, the agent learns to identify good and bad actions based on its current state, refining its policy over time.

Given the computationally intensive framework, the code is parallelized to run on multiple CPU threads, exploiting all the possible computer resources. Multiple trainings are run in parallel using three different machines with different hardware, preventing an exact comparison of the training times.

Verification

A complete verification of the RL models' correctness is not done, since these open-source libraries are largely used and already thoroughly tested and verified against many baseline cases. For example, the PPO algorithm implemented in `Ray-RLlib` is verified by its developers, since it is considered state-of-the-art. Also, the `Gym` class environment is developed and tested by OpenAI, which has verified its functionality in tons of problems. Similarly, the neural networks are created and built using `tensorflow` and `keras`, which are considered the standard in the RL community. In each GitHub repository of these libraries, there are tests folders, used to verify the implementation of every release [1]. What is verified in this thesis is that the composition of NN generated by the software is equivalent to the desired ones. It is done by printing the blocks that compose the NN and checking that they include all the elements that should theoretically be present, with the correct dimensions. Moreover, it is assessed that by increasing the dimensions of the network, the number of parameters rises accordingly. Another verification is done to check that the network's hyperparameters and flight dynamics variables indicated in the configuration files are correctly reported inside the `Gym` environment. Ultimately, the typical convergence profiles over a set of iterations, and the robustness to small hyperparameters variation shown in the following sections (Fig. 7.5), are hints to confirm the correctness of the RL models.

[1] `OpenAI`: `https://github.com/openai/gym/tree/master/tests`,
`Ray-Rllib`: `https://github.com/ray-project/ray/tree/master/rllib/tests`,
`tensorflow`: `https://github.com/tensorflow/tensorflow/tree/master/ci/official`

7.2. Reward function

One of the most crucial and challenging aspects of RL is defining the reward function. This scalar value communicates to the agent what needs to be achieved, indicating how good a specific action is at a given state in pursuit of the desired goal. The reward function is central to ensuring that the network learns the correct policy. A poorly designed reward function can lead the policy to diverge completely from the intended objective. Since the NN is trained to maximize the expected reward, if the reward function does not align with the user's goal, the agent will still learn a policy, but it may be entirely different from what was intended for. Thus, extreme care must be taken in developing the reward function.

Typically, the starting point involves understanding the underlying physical problem to identify its key features. It is also essential to define a clear goal that represents the policy the network should learn, similarly to what the objective function does in an optimization problem. There are different approaches for creating a reward function: it can be sparse, giving bonuses or penalties only at the end of each episode, or dense, providing rewards at each time step. However, relying solely on a sparse reward can result in slow or ineffective training, as the network may struggle to associate terminal rewards with the sequence of previous states-actions pairs. This is especially true when the problem involves a large number of time steps, states, and controls, making it difficult for the network to capture the cause-effect relationships between actions and delayed rewards. In such cases, it is often preferred to incorporate bonuses and penalties at each step to guide the agent toward the desired terminal states.

In the 6-DOF rocket landing problem, the primary goal is to land the vehicle with specific terminal properties, such as proximity to the landing site, minimal vertical and horizontal velocities, a controlled vertical angle, and acceptable residual angular velocity. Minimal fuel consumption is also a key factor. Therefore, the reward function should incorporate these five elements to ensure the agent meets the terminal constraints.

One approach is to design the reward function so that it penalizes mass consumption and assigns bonuses or penalties only at the end of each episode based on whether each constraint is met. In theory, this formulation encourages the agent to learn the most optimal policy, trading off the conflicting objectives of fuel consumption and terminal states precision. However, this reward function is not ideal in the complex and 6-DOF environment simulated in this thesis, where there are 14 states for position, velocity, attitude, angular rates, and mass, and 7 control inputs. Moreover, the closed-loop simulations (Fig. 6.8) show that, typically, each episode lasts about 25-35 seconds, which means about 250-350 steps, with 0.1 s as time step. Given problem complexity, broad design space, and long episodes, it is challenging for the neural network to discern cause-effect relationships between a single scalar reward received at the end of the episode and the entire sequence of observations and actions throughout the trajectory. The reward would condense the goodness of the actions of the entire episode, without diversification for each time step. This effect is exacerbated in the early stages of training when the policy is random, and the rocket may exhibit erratic behavior, such as rotating rapidly along each axis, landing far from the site, or having a very incorrect attitude. This makes it even harder for the agent to understand the cause of a poor trajectory from a single scalar reward.

Thus, providing dense rewards at each time step helps to guide the vehicle toward learning the correct policy. However, this approach can slightly reduce optimality because the trajectory becomes more "constrained" by the shape of the reward function. If the dense reward function emphasizes certain state behaviors, such as always moving toward the landing site or monotonically decreasing velocity, the rocket may learn a policy that deviates from the fuel-optimal one. Logarithmic and exponential functions are useful in guiding the vehicle towards "attractive points" (Bonasera et al., 2023) due to their steep changes near these points. Additionally, quadratic functions demonstrated to accelerate convergence (Fereoli et al., 2024).

Hence, creating a reward function is a complex, iterative process aimed at balancing the network's ability to learn with the solution's optimality. Even in simpler problems, like the OpenAI's two-dimensional Lunar Lander, a dense reward function is used to enhance the network's learning ability. In the scenario described in this thesis, using only a terminal reward would result in the network failing to learn anything meaningful. Therefore, both a shaping and a terminal reward are included.

Defining a suitable, efficient reward function for practical RL applications is a challenging task, without a single unique solution. It requires multiple iterations to find a correct reward function which makes the agent learn a meaningful policy. In this thesis, the reward function is formulated to capture the five pillars of the rocket landing scenario. To achieve this, a shaping reward term is formulated for the states related to the goal of fuel minimization and the terminal constraints derived from the requirements:

- Drive position components toward landing site (**GR-03**, **GR-04**)
- Guide velocity to be zero at landing site (**GR-05**, **GR-06**)
- Avoid large angular rates during the trajectory and touchdown (**GR-08**, **GR-09**)
- Keep the vehicle in a stable attitude (**GR-07**)
- Minimize mass consumption

From the initial to the final formulation, several refinements were made to adjust the learned policy. During training, various metrics were monitored using `TensorBoard` to track trends and infer which features the agent was learning. If performance was unsatisfactory, the reward function or its relative weights were adjusted based on what was observed. For example, these trends provide insights into whether the agent is minimizing errors in some state variables at the expense of others.

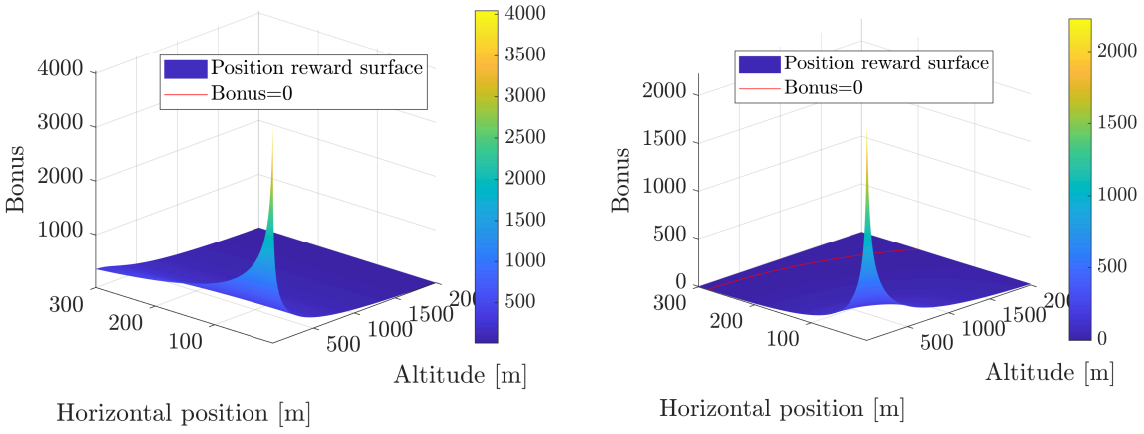
In the next paragraphs, each term of the reward function is shown and described. Moreover, their development and changes, throughout different iterations, are explained to understand how a wrong reward shaping can affect the final results.

It is remarked that all the states present in the reward function are normalized, as described in Section 7.1, to avoid different orders of magnitude.

Position

For the position component, a logarithmic dense reward function is defined, giving a bonus that increases as the rocket approaches the landing site. The reward is zero at each episode's initial position and theoretically infinite at the exact landing point $\mathbf{r}=\mathbf{0}_3$. As shown in Eq. (7.3), the logarithm's argument is the norm of the position components, each normalized by its initial value and divided by the square root of three. This setup ensures that only positions within the initial position boundaries yield a positive bonus. Therefore, the argument of the logarithm shall be one at the initial position and zero at the terminal position. To enhance convergence toward the "attractive" region close to the landing site, the square of this logarithm is used. However, squaring the expression could result in a positive bonus even for positions higher, or more horizontally, than the initial position. So, the expression is multiplied by its sign to ensure positive rewards only when within the initial boundaries, and negative penalties otherwise. This bonus is applied only if the rocket is descending. Conversely, if the rocket is hovering or ascending, a fixed penalty is applied to discourage this behavior.

$$R_{pos} = \begin{cases} -0.015 \left\{ \operatorname{sign}\left[\log\left(\frac{\left\| \frac{r_x}{r_{x0}}, \frac{r_y}{r_{y0}}, \frac{r_z}{r_{z0}} \right\|}{\sqrt{3}} \right) \right] \log\left(\frac{\left\| \frac{r_x}{r_{x0}}, \frac{r_y}{r_{y0}}, \frac{r_z}{r_{z0}} \right\|}{\sqrt{3}} \right)^2 \right\}, & \text{if } r_x(t) \leqslant r_x(t-1) \\ -0.2, & \text{otherwise} \end{cases} \tag{7.3}$$



(a) Wrong position reward function.

(b) Correct position reward function.

Figure 7.2: Comparison wrong and correct position reward.

This expression ensures that the area near the landing site offers a substantial bonus, driving the agent toward this region. The collected experiences of gaining large rewards near the landing site and large penalties for drifting away help the network to correctly update its internal parameters to learn the desired policy.

As a secondary effect, this position-related reward helps reduce vertical velocity near the landing site. Since proximity to the target yields large bonuses, the agent is motivated to remain in these favorable positions, rather than at a higher altitude or larger horizontal displacement, to maximize the cumulative reward. To prevent the rocket from hovering indefinitely at a low altitude with a large position bonus, a penalty for increasing altitude is introduced, as shown in Eq. (7.3). A maximum time horizon is set, along with a bonus when the rocket successfully lands. The combination of these three elements helps prevent the agent from "hacking" the reward function by hovering in favorable states indefinitely to accumulate bonuses. Staying for many steps in favorable states and then landing would give a bonus larger than hovering above the landing site until the time horizon expires.

In the first version of the reward function, dividing each non-dimensional position component by its initial value was not done. However, this approach had a critical drawback. In the initial "problem normalization" (the one explained in Section 7.1), an altitude of 2000 m becomes 1, while a horizontal position of 200 m becomes 0.1. Thus, a significant reward is given even when the rocket is close to the ground but still far horizontally. For example, landing 200 m from the site gives a normalized vector of (0, 0.1, 0.1), and its norm is close to 0, leading to a significant reward. This encourages the agent to exploit low-altitude states, regardless of the horizontal position, minimizing its importance and causing the rocket to drift away from the landing site while still earning a substantial reward.

In contrast, when scaling each position component by its initial value, landing 200 m from the target yields a normalized vector of (0, 1, 1). Hence, the argument of the logarithm is close to 1, giving an almost null reward, and encouraging the rocket to get closer to the landing site. This normalization prevents the agent from exploiting low-altitude states far from the target.

Figures 7.2(a-b) illustrate the difference between the two reward functions. In both the 3D plots, the altitude and one horizontal position are shown on the horizontal axes, while the value of the position reward is presented on the vertical axis. It can be immediately seen that in the correct reward expression (Fig. 7.2(b)) there are peak values near zero altitude and horizontal position, with a reward close to zero elsewhere, preventing the agent from "hacking" rewards by hovering and drifting away. By contrast, the wrong reward (Fig. 7.2(a)), gives substantial bonuses even at low altitudes with significant horizontal displacement. Moreover,

in the wrong formulation, the red line depicting where the position bonus is null is not even present within the altitude and horizontal position ranges shown.

Training with the wrong position reward led the rocket to hover at low altitudes for many seconds and drift away from the landing site. Adjusting the formulation with normalization corrected this issue, remarkably increasing the landing position precision.

Velocity

The velocity component of the reward function includes a penalty based on the deviation from a reference exponential velocity profile $\mathbf{v}_{ref}$ at each time step, as shown in Eq. (7.6) and Eq. (7.5). This suboptimal target profile guides the velocity components from their initial values to zero. The penalty is derived from the norm of the error between the rocket and the reference velocities. This approach is largely used in similar RL problems (Gaudet et al., 2020a), where it has proven to be effective, but also leading to suboptimal fuel consumption (Gaudet et al., 2020b). The profile is based on the rocket's position and the approximated time to go, which is estimated using the ratio between position and velocity vectors, as shown in (7.4).

$$t_{go} = \frac{\|\mathbf{r}\|}{\|\mathbf{v}\|} \tag{7.4}$$

$$\mathbf{v}_{ref} = -\|\mathbf{v}_0\| \frac{\mathbf{r}}{\|\mathbf{r}\|} \left(1 - e^{-\frac{t_{go}}{\tau}}\right) \tag{7.5}$$

$$R_{vel} = -0.01 \|\mathbf{v}_{ref} - \mathbf{v}\| \tag{7.6}$$

τ is set to 0.43, such that $\mathbf{v}_{ref}$ resembles the velocity profile from the optimal trajectory (Fig. 6.2(b)) when the position of the optimal trajectory is used as input. This velocity profile is more suitable for the vertical velocity component, rather than for the horizontal ones. Like the reference exponential profile, the vertical velocity is a monotonous decreasing variable, while the other two are not, as shown in the optimal trajectories plots in Fig. 6.2(b).

Given the reference velocity profile's suboptimal nature, its relative weight with respect to other terms in the total reward function is kept lower. The fact that it can give only negative values diminishes its importance as training progresses, because after a certain number of training iterations other reward terms (such as the position one) give positive bonuses that can be orders of magnitude larger. Therefore, the velocity reward term only partially influences the velocity to track a target reference profile, and it is probably the least influential term in the total reward function.

Moreover, as stated in the previous paragraph, the deceleration towards low-velocity states, at the end of the flight, is aided by the position reward component.

Angular rates

As set by requirement **GR-08**, the maximum angular rate on each axis is 15 deg/s (equivalent to $\frac{\pi}{12}$ rad/s) on each axis (roll, pitch, and yaw rate) as path constraint during the entire trajectory, This is chosen to avoid too large angular rates and to facilitate the decoupling of translational and rotational motion. Furthermore, if the angular rates are too large it is much harder to effectively use the fins and the TVC to control the trajectory of the vehicle, as the actuators' directions continuously change. Too large angular rates can also induce undesired effects not modeled in this problem, such as fuel sloshing.

At each time step, a penalty is applied if the angular rates exceed this limit, otherwise, a bonus is given. Both are proportional to the difference from the constraint, as shown in Eq. (7.7). This formulation discourages the agent from exceeding the maximum rates, stimulating it to respect the constraint and to minimize angular rates to avoid unwanted oscillations.

However, the requirement is not strictly enforced, meaning that the total reward can still be large if angular rates are kept low for a major portion of the flight, but the constraint of 15

deg/s is slightly exceeded for some time steps.

$$R_{\|\omega\|} = -0.0006\left[(\|\omega_x\| - \tilde{\omega}_{cstr}) + (\|\omega_y\| - \tilde{\omega}_{cstr}) + (\|\omega_z\| - \tilde{\omega}_{cstr})\right] \tag{7.7}$$

$\tilde{\omega}_{cstr}$ is the dimensionless maximum angular rate allowed by the requirement during flight.

Vertical angle

Maintaining the rocket's correct orientation throughout the flight is crucial for stability and to prevent catastrophic failures. The goal is to keep the rocket in a stable, near-vertical position so that the thrust counters gravity and downward velocity.

Given the **UEN** reference frame used, pitch and yaw are defined as the rotation around the horizontal y- and z-axes. From these, an auxiliary angle "vertical angle" (δ) is defined as the angle between the $X_{\mathcal{B}}$-axis of the rocket's body frame and the vertical X_{UEN}-axis of the landing site. Its mathematical definition is described in Eq. (6.9).

The reward function for this component is a piecewise function provided at each time step. To avoid the rocket points with the ogive downwards, a penalty is applied proportionally to how much the rocket's vertical angle exceeds 90 degrees. The more the rocket points downwards, the more the attitude is wrong and must be corrected. If the vertical angle is between 90 and 50 degrees, no reward is given. Below 50 degrees, a fixed bonus is awarded to avoid over-constraining the attitude. A constant bonus, instead of a proportional one, prevents giving an excessive reward for a fully vertical orientation, which could negatively affect other variables, such as giving less flexibility to correct the terminal landing position when starting from a horizontal displacement or a tilted attitude. Moreover, in the nominal OCP trajectory plots (Fig. 6.2(c)) the yaw and pitch angles go up to 50 deg on each axis.

Thus, this reward term guides the rocket to avoid pointing downwards while encouraging a pseudo-vertical attitude, allowing stability and necessary maneuvering during the entire flight. It is up to the terminal reward components to make sure that the rocket lands with a terminal vertical orientation, within the corresponding constraint.

$$R_\delta = \begin{cases} -0.02\left(\delta - \frac{\pi}{2}\right) & \text{if } \delta > \frac{\pi}{2} \\ 0 & \text{if } \frac{\pi}{2} > \delta > \frac{5\pi}{18} \\ +0.005, & \text{if } \delta < \frac{5\pi}{18} \end{cases} \tag{7.8}$$

Mass

The minimization of fuel consumption is the most straightforward term, where a penalty is given proportional to the mass difference at each time step, as shown in Eq. (7.9).

$$R_{mass} = 2\left(m(t_k) - m(t_{k-1})\right) \tag{7.9}$$

This is analogous to the formulation of the cost function in the OCP. However, while in the OCP one can choose to minimize the fuel consumption as the only mathematical objective, in the RL formulation this is just one of the many terms, all squeezed in a single scalar reward function. Therefore, RL cost is formulated similarly to single-objective optimization where multiple conflicting objectives are squeezed into one. For example, the minimization of fuel is conflicting with the reduction of vertical velocity, because conversion of fuel into thrust is needed to decelerate. Hence, the correct balance between the relative weights of mass consumption and velocity must be found.

In this work, a trial-and-error approach is used to tune the weights of each term in the reward function. They are iteratively updated after a partial or total evaluation of a training session's results. If one, or multiple variables are not satisfactory, their weights in the reward functions are tweaked, until the final version is found.

Terminal reward

While the five described components are provided at each time step, the terminal reward is given only when the termination conditions are met and the flight ends. Since the terminal constraints define the landing success, the goal is to give hints to the agent as to whether it landed correctly or not. The terminal states are not only a consequence of the controls at the last time step but also of the sequence of previous actions performed during the flight.

The requirements defined in Section 2.6 regarding terminal constraints are used in the terminal component of the reward function. If the threshold set by the requirement is exceeded, the agent gets a quadratic negative penalty, proportional to how much it is exceeded. Conversely, if the terminal variable fulfills the requirement, the agent receives a logarithmic positive bonus, based on the difference with respect to the threshold. The expression to assign the reward for each terminal state is:

$$R_x = \begin{cases} k_1(x - \epsilon\tilde{x})^2 & \text{if } x > \epsilon\tilde{x} \\ k_2 \log\left(\frac{x}{\epsilon\tilde{x}}\right) & \text{if } x < \epsilon\tilde{x} \end{cases} \tag{7.10}$$

x describes the normalized terminal variable considered, while $\tilde{x}$ represents the normalized terminal requirement corresponding to the considered variable. ϵ is a multiplicative factor that regulates the amplitude of the terminal constraints. It was first introduced by Federici and Zavoli (2024), Zavoli and Federici (2021) and it serves as a tolerance, decreasing exponentially through the training, as shown in Eq. (7.11). Its purpose is to allow larger terminal constraints at the beginning of the training when the rocket has a more exploratory behavior. As the training proceeds, ϵ decreases, slowly enforcing the actual terminal conditions, similar to an ϵ-constrained approach. Hence, allowing larger values in the beginning, when the agent has no clue on how to accomplish its tasks, helps to understand that it is advantageous to land close to the target states. After several iterations, fine-tuning takes place, reducing the allowable terminal constraints to those set by the requirements. In this way, the rocket is guided to progressively enforce the terminal values. This approach has proven successful in similar space applications (Federici and Furfaro, 2024).

$$\epsilon_i = \begin{cases} \epsilon_0 & \text{if } i < i_0 \\ \epsilon_0 \left(\frac{\epsilon_f}{\epsilon_i}\right)^{\frac{i-i_0}{i_f}} & \text{if } i_0 < i < i_f \\ \epsilon_f & \text{if } i > i_f \end{cases} \tag{7.11}$$

ϵ_0 and i_0 represent the initial value of the hyperparameter and the iteration when it starts to decrease, while ϵ_f and i_f are the final value and the corresponding iteration number when the decrease is stopped. The values used in the network's training of this thesis are presented in Table 7.1. In the initial training phase, where the agent explores the design space, it has been chosen to have fixed, larger, allowable constraints. They start to be reduced only when the agent has already learned a policy that is somehow aligned with the final one. Moreover, ϵ is constant again after about 70% of the training, such that the agent has fixed references to do the final refined tuning of its weights and biases.

The terminal states involved in the terminal reward function are those identified in the requirements. Hence, there is a term for the norm of the terminal horizontal position (**GR-04**), one for the final vertical velocity (**GR-06**), one for the norm of the final horizontal velocity (**GR-05**), one for the terminal vertical angle (**GR-07**), and one for the norm of the final angular rates (**GR-09**). They are all represented by expressions analogous to that in Eq. (7.10), where the threshold is the normalized value from the corresponding requirement.

The goal is to quadratically penalize when a terminal constraint is violated, guiding the agent to avoid repeating such errors. Conversely, when the requirement is met, a logarithmic

Table 7.1: Tuning parameters of the tolerance ϵ.

Variable	Value
ϵ_0	3
i_0	1500
ϵ_f	1
i_f	7000

function is used to suggest that achieving a terminal state of zero is the ideal outcome. The reason for using logarithmic functions for terminal bonuses is that they offer a theoretically infinite bonus when the value is exactly zero, unlike quadratic functions, which are capped. This gives a significant advantage to states closer to zero, motivating the agent to prioritize these values, which eventually are the desired ones.

Furthermore, there is a term for the landing requirement **GR-03**. It gives a fixed bonus if the rocket lands on the ground, while a penalty, proportional to the final altitude, is applied if the simulation terminates mid-air. Finally, an extra bonus is given if all the six requirements are fulfilled. Therefore, the final expressions for the terminal reward are:

$$R_{r_{hor},terminal} = \begin{cases} -32\left(\|r_y, r_z\| - \epsilon\tilde{r}_{hor}\right)^2 & \text{if } \|r_y, r_z\| > \epsilon\tilde{r}_{hor} \\ -1.6\log\left(\frac{\|r_y, r_z\|}{\epsilon\tilde{r}_{hor}}\right) & \text{if } \|r_y, r_z\| < \epsilon\tilde{r}_{hor} \end{cases} \tag{7.12}$$

$$R_{v_{vert},terminal} = \begin{cases} -24\left(\|v_x\| - \epsilon\tilde{v}_{vert}\right)^2 & \text{if } \|v_x\| > \epsilon\tilde{v}_{vert} \\ -0.24\log\left(\frac{\|v_x\|}{\epsilon\tilde{v}_{vert}}\right) & \text{if } \|v_x\| < \epsilon\tilde{v}_{vert} \end{cases} \tag{7.13}$$

$$R_{v_{hor},terminal} = \begin{cases} -16\left(\|v_y, v_z\| - \epsilon\tilde{v}_{hor}\right)^2 & \text{if } \|v_y, v_z\| > \epsilon\tilde{v}_{hor} \\ -0.16\log\left(\frac{\|v_y, v_z\|}{\epsilon\tilde{v}_{hor}}\right) & \text{if } \|v_y, v_z\| < \epsilon\tilde{v}_{hor} \end{cases} \tag{7.14}$$

$$R_{\delta,terminal} = \begin{cases} -0.8\left(\delta - \epsilon\tilde{\delta}\right)^2 & \text{if } \delta > \epsilon\tilde{\delta} \\ -0.12\log\left(\frac{\delta}{\epsilon\tilde{\delta}}\right) & \text{if } \delta < \epsilon\tilde{\delta} \end{cases} \tag{7.15}$$

$$R_{\|\omega\|,terminal} = \begin{cases} -4\cdot 10^{-5}\left(\|\omega_x, \omega_y, \omega_z\| - \epsilon\|\tilde{\omega}\|\right)^2 & \text{if } \|\omega_x, \omega_y, \omega_z\| > \epsilon\|\tilde{\omega}\| \\ -0.04\log\left(\frac{\|\omega_x, \omega_y, \omega_z\|}{\epsilon\|\tilde{\omega}\|}\right) & \text{if } \|\omega_x, \omega_y, \omega_z\| < \epsilon\|\tilde{\omega}\| \end{cases} \tag{7.16}$$

$$R_{landing,terminal} = \begin{cases} -8r_x & \text{if } r_x > \tilde{r_x} \\ 0.8 & \text{if } r_x < \tilde{r_x} \end{cases} \tag{7.17}$$

$$R_{bonus,terminal} = \begin{cases} 10 & \text{if all constraints respected} \\ 0 & \text{otherwise} \end{cases} \tag{7.18}$$

Equation (7.19) presents the final expression for the terminal component of the reward as sum of all the previously shown terms.

$$\begin{aligned} R_{terminal} = & R_{r_{hor},terminal} + R_{v_{vert},terminal} + R_{v_{hor},terminal} + R_{\delta,terminal} + R_{\|\omega\|,terminal} \\ & + R_{landing,terminal} + R_{bonus,terminal} \end{aligned} \tag{7.19}$$

Table 7.2: meta-RL hyperparameters setup.

Parameter	Value
Training iterations	9500
Batch size	8192
SGD minibatch size	128
SGD iterations	15
PPO clip parameter	0.01
Value function network	[256,128,128]
Number workers	8
Environment per worker	8
Activation function	tanh
Attention dimension	128
Time horizon	400 s
Number of heads	4
Memory length	250
Sequence length	128
Previous actions and rewards	250
Encoder layers	[256,128,128]

Summary

The final expression of the total reward function is the sum of the terms given at every time step and the ones given only at the end of the episode:

$$R = R_{pos} + R_{vel} + R_{\|\omega\|} + R_{\delta} + R_{mass} + \frac{\epsilon_0}{\epsilon} R_{terminal} \tag{7.20}$$

The term $\frac{\epsilon_0}{\epsilon}$ is used to incrementally increase the importance of the terminal constraints compared to the dense reward. The purpose of the dense reward is to initially lead the agent toward the correct direction to have a meaningful policy, while the terminal constraints are the real objectives that should be optimized at the end of the training.

It is reminded that the reward expression is calculated at each time step, where $R_{terminal}$ is always zero, apart from the last time step.

7.3. Results

After many trials and tuning iterations of the reward function, satisfactory results are found using the formulation presented in Section 7.2. The results in Section 7.3.1 are not the best, but they are adopted as the baseline, serving as the starting point for improvements, comparisons, sensitivity analysis, and robustness analysis presented in the following sections.

Table 7.2 presents all the relevant parameters used in the problem setup, in the neural network, and in the RL algorithm, to reproduce the results. Except for the last five elements of Table 7.2, all the other are fixed for every training done in this thesis. Furthermore, for each training, the learning rate is linearly interpolated between 10^{-4} and 10^{-6} between 0 and $8{\cdot}10^7$ time steps. The entropy coefficient is linearly interpolated between 10^{-4} and 0 between 0 and $4{\cdot}10^7$ time steps.

Since all the closed-loop simulations last no more than 35 seconds (Fig. 6.8), the maximum time horizon in RL training is set to 40 seconds. This prevents over-exploiting the reward function, by getting bonuses for being in favorable states for an infinite amount of time.

After training the agent for 9500 iterations (approximately five days), the solution is postprocessed and analyzed. As typically done in RL problems, once the training is finished and the network is deployed for analysis, the stochastic component is removed, making the policy

deterministic (Federici and Furfaro, 2024). Therefore, instead of outputting the probability of taking an action from a distribution, the network gives the expected value of the control action distribution. In PPO, since the action distributions are considered Gaussian, the output is its mean. A set of 1000 Monte Carlo simulations, considering all the uncertainties and initial dispersions, is run to evaluate the performances of the learned policy.

7.3.1. Baseline results

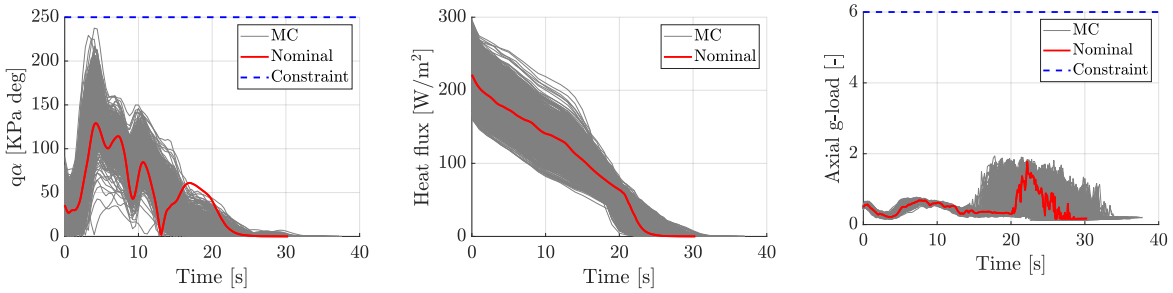
What can be seen in Table 7.3 is that the mean values of the terminal constraints of horizontal position, horizontal velocity, angular rates, and vertical angle are very close to zero. Also, the dispersions are not excessively wide, given that the standard deviations are small. This is confirmed by Fig. 7.4(a-e), where most simulations are very close to the target, with only a handful of runs that exceed some constraints. It is striking that the vertical velocity never respects the given requirement on its terminal value, consistently violating it, by a small margin. While the rocket should land with a vertical velocity comprised between -2 and 0 m/s, the mean value is about -5 m/s (Fig. 7.4(e)). However, there are some speculations that even SpaceX Falcon 9 lands with a vertical velocity of about -5 m/s, but, since no information is publicly available, this cannot be verified. Furthermore, if the constraints in the most similar RL 6-DOF atmospheric landing work (Rosa et al., 2023) are used for comparison, the vertical velocity would always fulfill its requirement. It means that the learned policy of this section is a good one, that probably only needs some fine refinement on the vertical velocity component.

The weight of the vertical velocity reward was increased to try to improve its final value. However, it was noticed that reducing vertical velocity violations led to deterioration in other terminal variables, especially in rotational motion. As illustrated in Fig. 7.4(f), it is clear that runs with the best vertical velocity often exceeded angular rate or vertical angle constraints.

This trade-off between vertical velocity and rotational motion variables is recurring across several training sessions, for two reasons. Firstly, wind speed (up to 15 m/s) makes the horizontal velocity larger than the vertical one, creating large aerodynamic angles and induced moments. It is reminded that for such large aerodynamic angles, the formulation of the aerodynamics for body and fins is much less accurate, limiting the correctness of the solution. Secondly, the agent sometimes hacks the reward function to exploit position bonuses closer to the landing site. Therefore, for a few particular combinations of initial states and dynamics uncertainties, the interaction between wind susceptibility and reward hacking makes the agent perform actions to lower the velocity and try to go as much as possible toward the landing site. In these few cases, large TVC deflections are commanded near the end of the flight, creating slightly too large angular rates and vertical angles than the terminal constraints.

Enforcing the vertical velocity constraint is challenging because it is the only variable that solely depends on thrust magnitude, creating a conflict in the cost function with mass minimization. All the other variables are influenced by both thrust, TVC, and fins, offering more control options for optimization. Unlike other states, the vertical velocity is "one-sided," only exceeding the target value of 0 m/s on the negative side. If it goes positive, the rocket skips upward, making landing impossible unless the engine is shut down. This limitation causes the neural network to conservatively prioritize other variables in such uncertain environments, avoiding the risk of skipping or worsening rotational states.

A "Partially Successful Runs" (PSR) index is introduced to evaluate performance. Since vertical velocity constraint is not met in most policies, a run is considered partially successful if all other variables (terminal altitude, position, horizontal velocity, angular rates, and vertical angle) are within the allowable limits of requirements **GR-03**, **GR-04**, **GR-06**, **GR-07**, and **GR-09**. In this training, as shown in Table 7.4, the number of partially successful runs is 987. Figures 7.3.1(b,d) show that the few failed runs exceed angular rates or vertical angle at most by 3 deg/s or 1 deg, respectively. Therefore, even if those are considered failed, they do



(a) Dynamic loads. **(b)** Heat flux. **(c)** Axial g-load.

Figure 7.3: Aero-thermal loads.

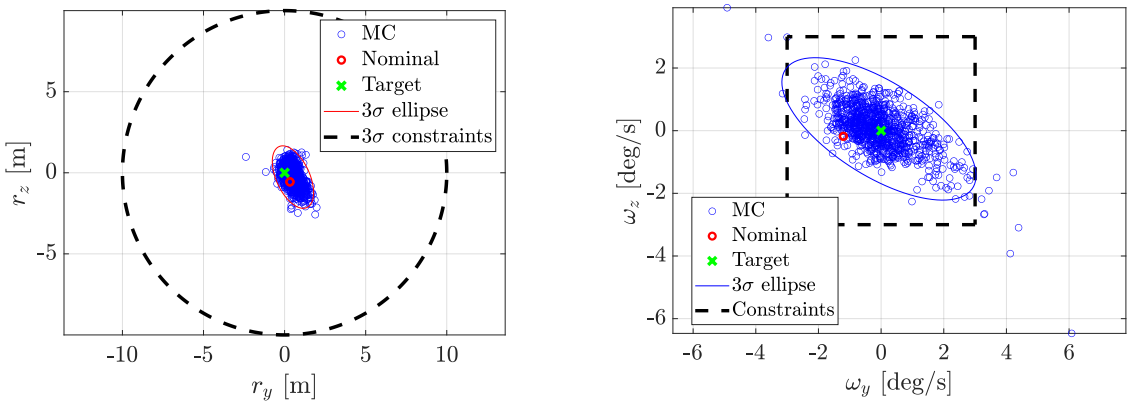
not completely diverge, nor violate the constraints by a huge margin. Position and horizontal velocity are respected in all 1000 cases, as shown in Fig. 7.3.1(a,c)

To assess whether the RL policy is suitable for real-time execution, the time required for a forward pass of the NN is evaluated. The network processes input observation states and outputs optimal actions in about 6 ms on average using an Intel® Core™ i7-8565U processor. This proves the compliance with requirements **GR-02**, showing potential to run online.

Figures 7.3(a-c) show the thermo-mechanical loads. Dynamic loads are within requirement, with a peak observed after a few seconds of flight. Since density barely changes and velocity monotonously decreases, also dynamic pressure is monotonous. Hence, this peak is mainly due to large aerodynamic angles caused by combinations of initial conditions and strong lateral winds that lead to significant angles of attack or sideslip.

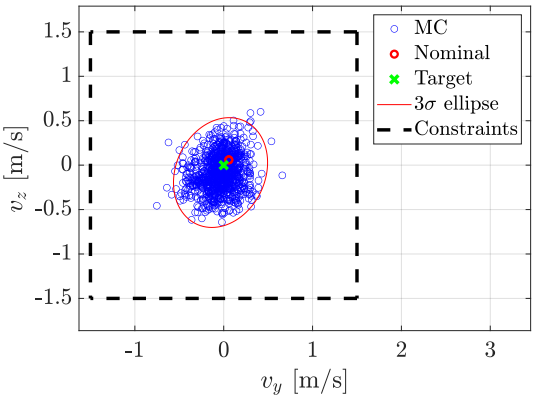
Heat flux is orders of magnitude lower than the constraint, as expected since the requirement is based on re-entry conditions, while terminal landing involves much lower velocities.

The axial g-load is smaller than the requirement, by a margin. For instance, considering the worst case, when the thrust is 100 kN and the mass is 3500 kg, the thrust acceleration is about 28.5 m/s^2. By subtracting the gravitational acceleration and dividing it by 9.81, it yields slightly less than 2 g. Only the axial component is considered because the lateral accelerations are orders of magnitude smaller, since the thrust force is by far the most influential one and it is mainly directed through the longitudinal axis of the rocket.

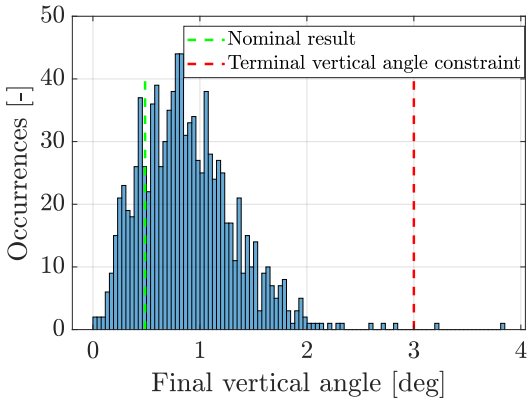


(a) Terminal horizontal position. **(b)** Terminal angular rates.

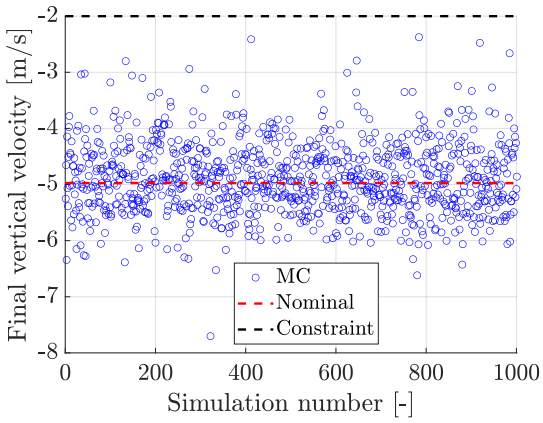
Figure 7.4: Plots baseline policy.



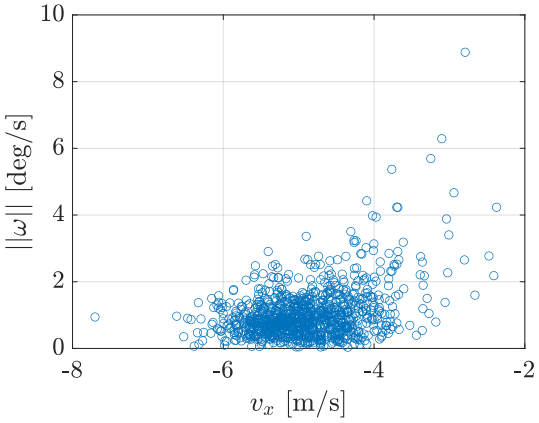
(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.



(f) Relationship v_x-ω

Figure 7.4: Plots baseline policy. - Concluded

7.3.2. Sensitivity analysis GTrXL hyperparameters

It is commonly known that RL problems have many tunable parameters, and the success of training often depends on finding the right values. Some typical parameters are the learning rate, the batch size, the number of layers, the entropy coefficient, or the clip parameters (for PPO). Introducing the GTrXL NNs, the number of tunable hyperparameters increases significantly, including attention dimension, number of heads, Transformer units, sequence length, memory, and encoder layers.

Given the novelty of GTrXL network, it is deemed to be interesting to analyze the effect of changing Transformer-specific parameters, rather than the more generic ones, for two reasons. First of all, no previous similar work analyzed the sensitivity of the solution to these parameters, while there are many RL studies dedicated to the effects of more traditional parameters, such as learning rate or batch size. Secondly, the goal is to find the striking factor such that future studies using this type of NN know what could strongly affect the quality of the solution.

The parameters analyzed are the number of attention heads, sequence and memory length, encoder depth, and attention dimension. A sensitivity analysis was conducted by varying them one-at-a-time while keeping the others constant, as in the baseline case (Table 7.2). Each training lasts for the same number of iterations (9500), to have fair comparisons.

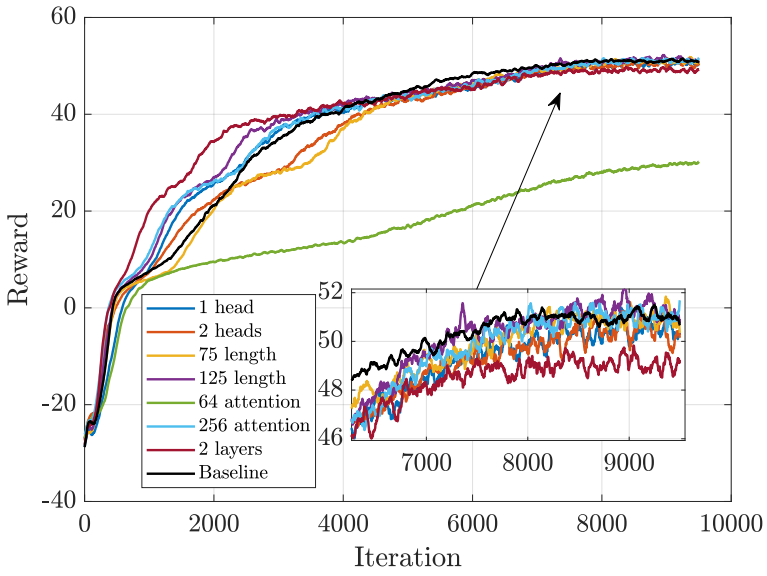


Figure 7.5: Moving average of rewards.

Sequence, memory, action and reward length

For instance, the length of the sequence and memory determines "how far" the network can look into the past, leveraging previous information to perform actions, during training. A sequence is a set of states from consecutive time steps, that are processed together by the network. The memory is a set of GTrXL outputs from previous sequences processing, that are concatenated to augment the network's inputs. Moreover, previous actions and rewards are also added as inputs. In theory, longer sequences and memories increase the network's ability to correlate distance events during the flight, at the expense of a longer training time. Since these four parameters are closely related to each other, they are varied together. Starting from memory, previous actions, and rewards 250 steps long and a sequence of 128 steps (the maximum length allowed by the mini-batch), they are first reduced to 125 steps and then to 75, to assess how the results are influenced.

As shown in Table 7.3, the mean values are very similar. There is a slight degradation trend as the lengths of sequence and memory are reduced, but it is not a drastic decline. When only 75 is used as length, a small subset of simulations do not reach the landing site altitude, terminating the episode with null vertical velocity at a few meters of altitude (Table 7.4). This is reflected in the mean value of the vertical velocity, which is improved due to these simulations terminating at 0 m/s. Therefore using any of the three lengths is almost equivalent since the small differences can be considered well within the variance of the problem variables.

The explanation behind these results is twofold: firstly, given that a single simulation lasts about 330 time steps, even the shortest sequence of 75 steps, and a memory 75 elements long are probably enough to capture any relevant information. While the sequence is limited to the last 75 steps, the memory allows to indirectly look much further in time, also from previous trajectories, to train the network accurately. Moreover, sequence and memory lengths do not influence the number of the network's internal parameters because the memory introduces only non-tunable parameters, not tweaked during the training. Only the length of previous actions and rewards changes the number of weights and biases, but not drastically in these cases.

Number of heads

The number of heads affects the network's ability to capture different features of a state. An example from Natural Language Processing can help to understand this. The word "*mine*" can represent the pronoun of something belonging to me, the hole in the ground where substances

like coal are extracted, or the type of bomb that explodes when vehicles pass over it. For a computer, these meanings are condensed into a single array of numbers. The attention mechanism is what uses the context to understand what is meant in a particular case. Similarly, the same use of the attention mechanism is done in rocket landing. Employing memory and sequence, the context is understood and the correct "meaning" of a state is inferred.

The number of heads also influences the ability of a network to catch different features because each head has its key, query, and value matrices. Therefore, using more heads augments the network's ability to capture multiple features. Given that the network's attention dimension is fixed, using more heads means that each head's attention dimension is smaller. For example, for a GTrXL with an attention dimension of 128, if four heads are used, each has a dimension of 32, while if two are used, each has a size of 64. Hence, if four heads are used, more nuances of states can be learned, focusing on simpler features, while if only one is used, it captures less diversity, but learns more complex features, due to the larger attention dimension.

The results in Table 7.3 show minimal differences when using a different number of heads, with mean and standard deviation very close to each other. Only the terminal vertical velocity has a noticeable trend: its absolute value increases as the number of heads is reduced. This is tightly related to the observation that when the rocket lands with lower vertical velocity (more heads), some runs exceed other terminal constraints, while if the landing velocity is larger (fewer heads), all the other constraints are met (Table 7.4). The reason is that a slower landing vehicle is more affected by wind and reward hacking, as already explained in Section 7.3.

The marginal differences in results can be attributed to the fixed attention dimension, meaning the total number of network weights and biases remains constant regardless of the number of heads. Although fewer heads may capture fewer features, the network's size is sufficient to learn the problem's critical aspects. Using different numbers of heads leads to different policies, resulting in different local minima, where rotational motion or vertical velocity is slightly worse. Although these policies yield similar cumulative rewards, as shown in Fig. 7.5, they perform different actions to achieve similar outcomes.

Number of encoder layers

As illustrated in Fig. 4.4(a) an encoder of fully connected layers transforms the raw input observation into a higher-dimensional input representation suitable for the Transformer. This operation is affine to what is done in Language Processing, where the words are first converted to a high dimensional numeric representation, before being fed to the attention-based NNs.

Initially, three fully connected layers were used in the encoder. Since the results were

Table 7.3: Statistics of different trainings changing GTrXL hyperparameters.

Training	**Norm Horizontal Position [m]**		**Norm Horizontal Velocity [m/s]**		**Vertical Velocity [m/s]**		**Norm Angular rates [deg/s]**		**Vertical angle [deg]**		**Mass Consumption [kg]**	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Baseline	0.84	0.48	0.25	0.14	-4.89	0.64	1.03	0.76	0.88	0.45	503.3	19.5
2 heads	1.16	0.68	0.25	0.20	-4.90	1.64	0.48	0.43	0.59	0.57	526.3	28.0
1 head	1.54	0.67	0.33	0.19	-7.64	1.10	0.73	0.34	1.45	0.48	506.0	14.1
125 length	1.22	0.65	0.29	0.15	-6.06	0.96	0.60	0.32	0.67	0.37	525.6	19.3
75 length	1.29	0.77	0.29	0.18	-5.37	1.63	0.69	0.40	0.61	0.34	534.6	23.1
64 attention	14.47	15.06	5.02	4.05	-6.19	7.3	12.96	7.11	12.98	9.52	590.0	60.4
256 attention	1.00	0.80	0.45	0.26	-8.78	1.27	0.90	0.49	0.93	0.56	500.1	14.1
2 layers	1.96	0.91	0.31	0.21	-7.80	0.82	0.57	0.27	0.75	0.47	509.3	15.8

Table 7.4: Success rates and failure causes.

Training	PSR (out of 1000)	Altitude	Horizontal position	Horizontal velocity	Vertical angle	Angular rates
Baseline	987	0	0	0	2	13
2 heads	982	7	1	5	12	2
1 head	1000	0	0	0	0	0
125 length	1000	0	0	0	0	0
75 length	980	20	0	0	0	0
64 attention	28	466	349	819	918	850
256 attention	993	0	0	6	4	0
2 layers	1000	0	0	0	0	0

satisfactory, they were reduced to two to assess what effect this change would have on the final policy results. By reducing from three (of size 256-128-128) to two layers (of size 256-128) the number of encoder internal parameters is reduced from 110,209 to 77,185.

From Fig. 7.5 it stands out that, using only two layers in the encoder, the agent learns faster, but in a less accurate way. It can be seen that the reward function has the fastest initial rise, eventually stabilizing at a slightly lower terminal value, compared to the Baseline. This behavior happens because a smaller encoder reduces, even if not drastically, the number of internal parameters so that the network can tune them quickly. However, this comes at the cost of a slightly worse final performance on the vertical velocity, as shown in Table 7.3. Using a smaller encoder with fewer parameters introduces the drawback of having an initial transformation, from the observation to the Transformer input, that gives a smaller high-dimensional representation, capturing fewer aspects of the states.

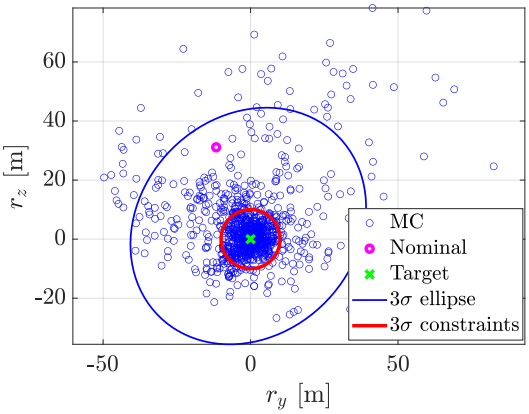
Attention dimension

The attention dimension is the parameter with the largest impact on the size of the network. A GTrXL with an attention dimension of 128 has 585,991 weights and biases while reducing it to 64 lowers the number to 215,175. Larger networks with more parameters generally better approximate the function mapping inputs to outputs. Too small networks do not have enough parameters to accurately capture the underlying function, similar to how a linear function poorly approximates a cubic dataset. In contrast, increasing the network size demands more episodes to properly tune all internal parameters, leading to longer training times. Thus, a balance between these characteristics is essential.

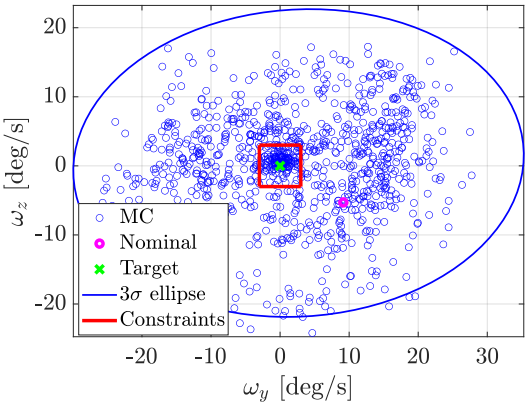
While the baseline attention dimension of 128 shows good performance, varying it to 64 or 256 was explored to see if a smaller value disrupts results or if a larger value improves vertical velocity performance. Table 7.3 presents a dramatic degradation when the attention dimension is reduced to 64. Most terminal variables fall outside their constraints, with large dispersions, and even a large fuel consumption. Figures 7.6(a-e) and Table 7.4 clearly illustrate what was just mentioned: 972 failed simulations, with several exceeding the constraints by a large margin. For instance, some trajectories end up almost 100 m from the target, with horizontal and vertical velocities of 20 m/s and -50 m/s, respectively. Also, the rotation motion is strongly affected, with runs that terminate with a vertical angle of 60 degrees or 30 deg/s of angular rates. This indicates that 64 is too small for this hyperparameter, likely due to an insufficient amount of internal parameters to approximate the underlying function.

Conversely, increasing the attention dimension to 256 does not give the expected advantages. While most metrics, such as terminal position, horizontal velocity, and attitude, show values close to the baseline, the vertical velocity worsens, with a mean value of almost -9 m/s, as described in Table 7.3. One speculation about this behavior is that a larger network would

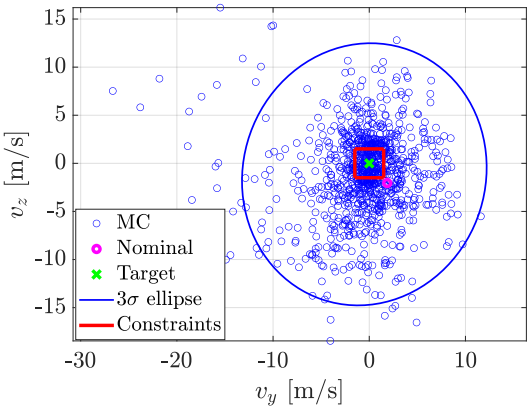
require more training iterations to adjust its internal parameters and learn an optimal policy. When increasing the attention dimension from 128 to 256, the number of weights and biases surges from 585,991 to 1,794,567, raising the training time from 5 to 7.5 days approximately.



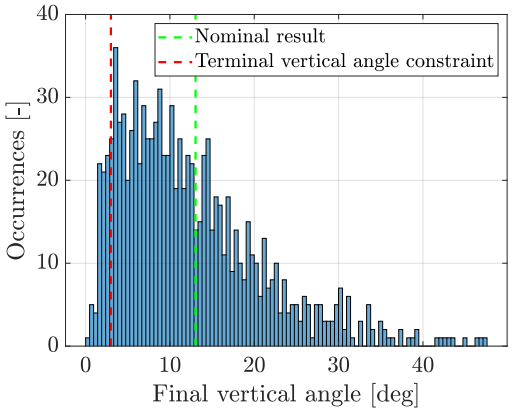
(a) Terminal horizontal position.



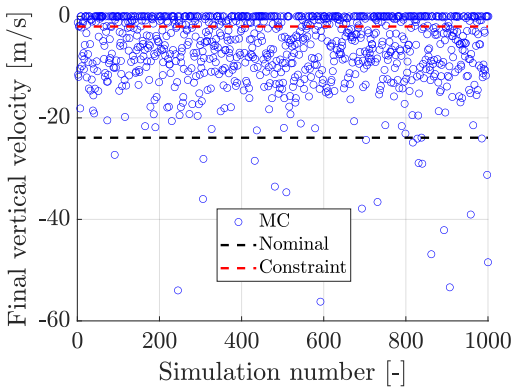
(b) Terminal angular rates.



(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.

Figure 7.6: Plots policy with attention dimension equal to 64.

Table 7.5: Statistics of different trainings using LSTM

Training	Norm Horizontal Position [m]		Norm Horizontal Velocity [m/s]		Vertical Velocity [m/s]		Norm Angular rates [deg/s]		Vertical angle [deg]		Mass Consumption [kg]	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
128 size	1.90	1.17	0.82	0.66	-5.20	1.27	6.77	5.57	3.13	3.51	505.45	21.41
256 size	0.79	0.76	0.55	0.42	-3.92	2.31	4.39	3.49	2.11	2.06	506.36	24.06

7.3.3. Comparison with LSTM

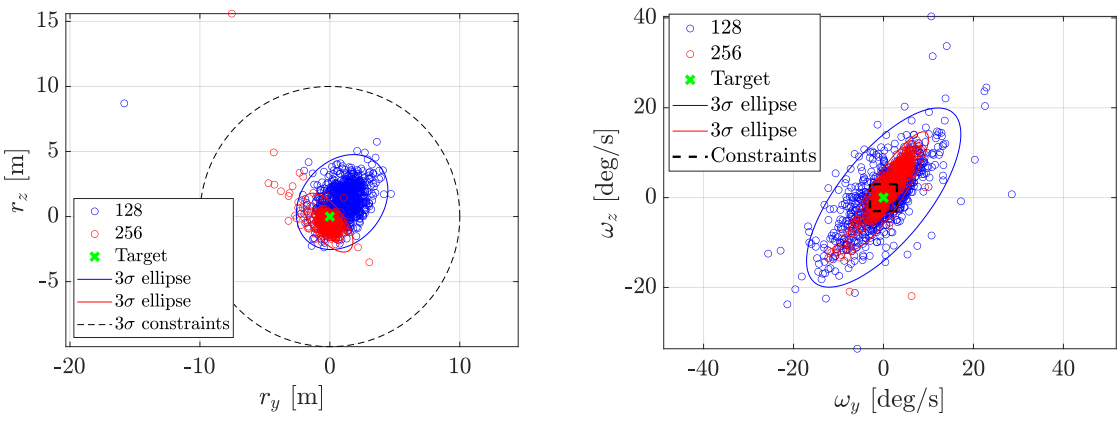
In Section 4.5 it was hypothesized that attention-based NNs, such as GTrXL, would perform better in complex problems, due to their enhanced ability to capture long-term dependencies in time series. To test this, LSTM NNs are also trained for comparison. Both architectures have similar sizes for a fair comparison in terms of performance and training time. The LSTM network uses a 3-layer encoder and a single LSTM cell of size 128. Since the attention dimension is found to be the most influential hyperparameter for GTrXL networks, another training is done by doubling the LSTM size, to have a broader overview of its influence on results.

The results presented in Fig. 7.7(a-e) and Table 7.5 reveal that the terminal dispersions with LSTM are significantly larger than with GTrXL, especially affecting angular rates and vertical angle, with mean values two to seven times worse than the Transformer results. Even more eye-catching are the large dispersions of these terminal states, giving a huge number of failed simulations, with substantial terminal errors, as shown in Fig. 7.7(b,c,d).

This clearly shows that LSTM-based NNs struggle to effectively capture the different aspects and relationships of such a complex problem. It is an evident limit of these kinds of networks, where previous information is compressed in only two internal hidden states, that are not enough to build a secondary policy that handles all the uncertainties and dispersions.

Doubling the LSTM cell size improves results somewhat, but not sufficiently to get closer to Transformers-based policies. Additionally, it also increases the training time from five to nine days, compared to Transformers. Therefore, it becomes a very slow, inefficient process, without producing high-quality performances. Using the index previously presented, only 278 runs are partially successful when the cell size is 128, rising to 489 when the cell size is 256.

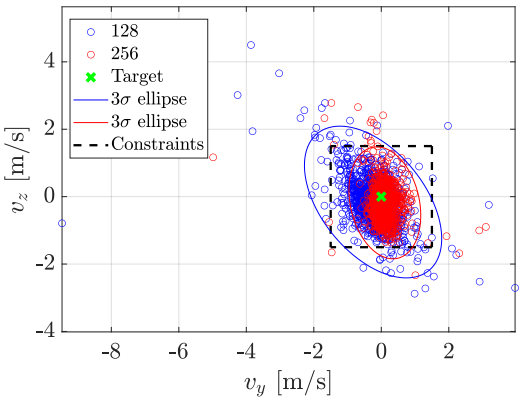
Thus, the low success rate, the larger mean, and the dispersion of terminal variables prove that LSTM networks are an unsuitable choice for such a problem, largely underperforming compared to Transformers-based neural networks.



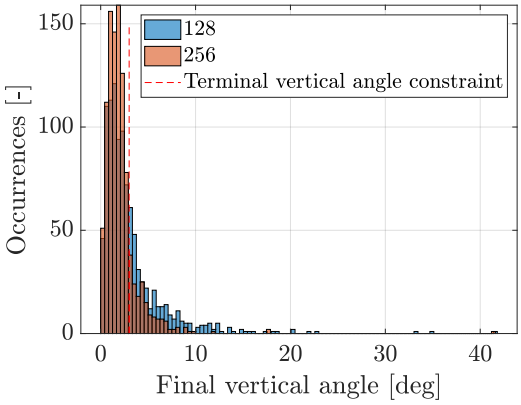
(a) Terminal horizontal position.

(b) Terminal angular rates.

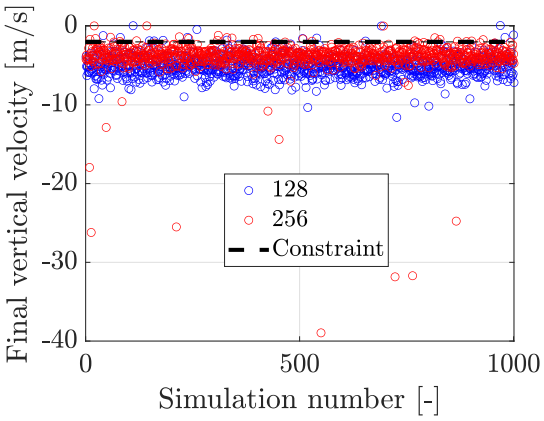
Figure 7.7: Plots comparison LSTM policies with cell size 128 and 256.



(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.

Figure 7.7: Plots comparison LSTM policies with cell size 128 and 256. - Concluded

7.3.4. Robustness analysis

So far, the network has effectively managed complex dynamics, and large state and control spaces, showing robustness to variations in initial conditions and dynamic uncertainties in density and aerodynamic coefficients. These are known to be some of the most relevant perturbations present in the system. Indeed, the accuracy of initial states depends on the errors accumulated during previous flight phases, while density and aerodynamics are parameters difficult to measure and with significant variations due to many external factors.

However, other error sources affect G&C algorithms, primarily related to hardware, such as the errors introduced by the navigation and control modules.

The navigation module introduces errors due to biases, misalignment, and intrinsic sensor inaccuracies. A Kalman filter is typically used to estimate states by combining multiple sensor measurements to reduce noise and improve accuracy. Despite this, estimated states are not

Table 7.6: Navigation uncertainties.

State variable	1σ
σ_{r_x},σ_{r_y},σ_{r_z}	0.15 m
σ_{v_x},σ_{v_y},σ_{v_z}	0.1 m/s
σ_{ω_x},σ_{ω_y},σ_{ω_z}	0.15 deg/s
σ_ϕ, σ_θ, σ_ψ	0.15 deg

Table 7.7: Control uncertainties.

Control variable	1σ
$\sigma_{T_{bias}}$	1%
$\sigma_{T_{error}}$	1%
$\sigma_{\dot{\epsilon}_T}$, $\sigma_{\dot{\psi}_T}$	0.25 deg/s
$\sigma_{\dot{\beta}_{fin,P1}}$, $\sigma_{\dot{\beta}_{fin,P2}}$	0.25 deg/s
$\sigma_{\dot{\beta}_{fin,Y3}}$, $\sigma_{\dot{\beta}_{fin,Y4}}$	0.25 deg/s

error-free, so the G&C algorithm must handle these errors effectively. The more inaccurate the estimated values are, the more the G&C policy has to be robust. Given the time constraints of this thesis, a Kalman filter could not be implemented and tested thoroughly, but errors in the navigation are introduced, to recreate the estimated states. Hence, to simulate navigation errors, Gaussian noise is added to the actual states, generating the estimated states used by the G&C system to compute optimal control actions. In this way, errors are sampled from predefined distributions at each time step, and added to the actual states, to avoid accumulation. While a Kalman filter would compensate for errors, the method used here can cause errors to accumulate over time. Therefore, to avoid this effect, the estimated states are used only to calculate the control actions, while the actual states propagate the dynamics one step forward in time. In Table 7.6, the standard deviation for each error is shown. They all have a mean value equal to zero, and they are assumed to be uncorrelated with each other.

Similarly, hardware errors in the control module arise because actual actuators' performance deviates from commanded values. Moreover, in this thesis, actuator dynamics are not considered, missing the modeling of transients due to friction, inertia, and damping effects. Hence, even if the G&C algorithm commands thrust magnitude, TVC, and fin deflection rates, in reality, those values are never exactly replicated. To model these real-world effects, errors are added to the commanded control variables output by the neural network. TVC and fin deflection rates errors are sampled from Gaussian distributions at each time step, and summed to the commanded values. Thrust magnitude error combines two terms. A constant bias, sampled from a Gaussian distribution at the start of each episode, multiplies the nominal thrust value along the entire trajectory. An additional multiplicative factor is added, sampled from a Gaussian distribution at each time step, to simulate uncertain thrust values. In Table 7.7 the standard deviation for each error is presented. They all have a mean equal to zero and are assumed uncorrelated between each other.

Figures 7.8(a-e) show plots regarding the terminal results on position, velocity, vertical angle, and angular rates for this training, compared to the one using RCS. At first impact, it can be seen that the terminal results of this training are all concentrated around the target, with very little dispersion, fulfilling the required constraints with a considerable margin.

The results illustrate that all the 1000 Monte Carlo simulations meet the imposed terminal requirements, except for the vertical velocity, as experienced in the Baseline case. As pointed out in Table 7.8, the mean and standard deviations values are in line with what was found out in the policies without control and navigation errors (Table 7.3). Only the vertical velocity is slightly worsened compared to the Baseline. The results suggest that adding more uncertainties barely affects the performance of the NN. Therefore, it is clear that the learned G&C policy is robust to many sources of uncertainties in the dynamics, as well as in the navigation and control modules. These results confirm the initial hypothesis that using attention-based neural networks would give a significant advantage in terms of robustness of the policy.

Table 7.8: Statistics of trainings including control and navigation errors and RCS.

Training	**Norm Horizontal Position [m]**		**Norm Horizontal Velocity [m/s]**		**Vertical Velocity [m/s]**		**Norm Angular rates [deg/s]**		**Vertical angle [deg]**		**Mass Consumption [kg]**	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Ctrl/Nav errors	1.74	0.89	0.37	0.22	-6.30	0.92	0.85	0.44	0.91	0.45	511.4	15.5
RCS	1.00	0.49	0.54	0.25	-6.76	0.76	1.72	0.88	1.81	0.80	494.5	17.2

7.3.5. Addition of RCS

To ensure a fair comparison with the second set of closed-loop Monte Carlo simulations using classical feedback methods, the RCS is also incorporated into the RL problem formulation. It is included in the dynamics identically: a torque is added around each axis, ignoring thruster effects on translational motion, and limiting the maximum moment to 9000 Nm on pitch and yaw and 1350 Nm on roll. Thus, the network's outputs increase from 7 to 10 control actions.

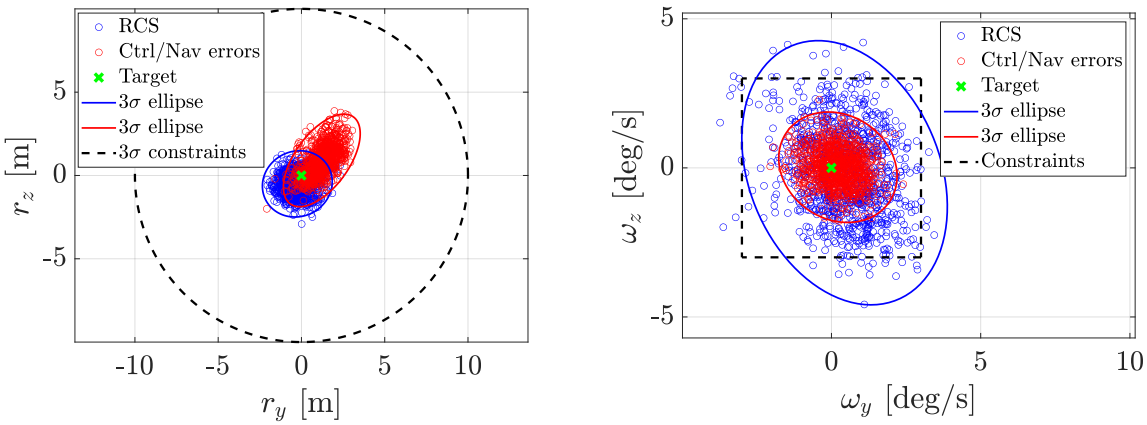
The results show that the performances of this network are slightly worse than the ones previously presented. Comparing Table 7.8 with Table 7.3 and Table 7.8, it can be noted that the mean terminal vertical angle is about one degree larger than the Baseline and the GTrXL training with additional control and navigation errors. The norm of the angular rates is also approximately one deg/s higher. Figures 7.8(b,d) illustrate what is presented by the numbers, showing that there are several simulations exceeding terminal constraints on the rotational motion. Figure 7.8(a,c) shows that the terminal position and horizontal velocity constraints are always respected, even if the latter is sometimes close to its limit.

A likely reason for this decline is that the number of control actions increases while the network size remains the same. Consequently, the network must capture more features of the problem without additional internal parameters, leading to degraded performances.

This issue is particularly evident in rotational motion since the RCS impacts directly this aspect of the problem. Using the experiences collected during training, the agent has to understand how to manage and balance the fins and RCS actuators to have the correct attitude and angular rates. Since they both affect the same variables, a larger dimension of the network is required to understand the relationships and couplings between them.

7.3.6. Removal of inputs

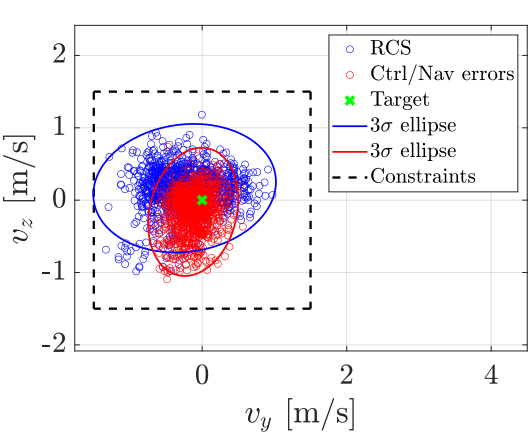
Initially, it was deemed appropriate to feed the network with as much information as available during flight, so that it could exploit all the data to execute the most optimal control actions. Therefore, not only classical states such as position, velocity, attitude, and angular rates, but also information on mass, time, and position of the actuators were given as inputs. The rationale behind it is that all these variables influence the dynamics, hence their knowledge aids in determining the correct actions. For example, rocket mass affects thrust acceleration. Thus, knowing it would give hints to the network about how much thrust it needs to output to have a certain acceleration to perform maneuvers for pinpoint landing. Similarly, knowing the current TVC or fin deflections helps the network to choose more accurate actuator deflection rates to create the desired forces and torques. It is somehow similar to the fact that in the



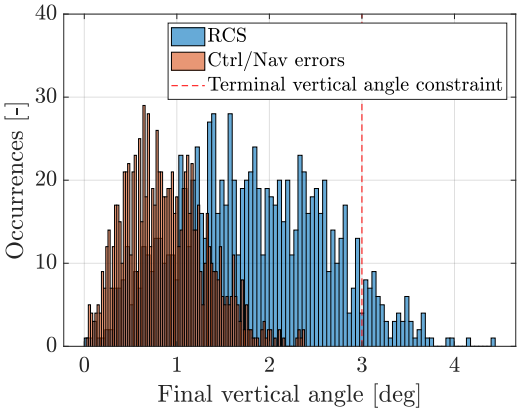
(a) Terminal horizontal position.

(b) Terminal angular rates.

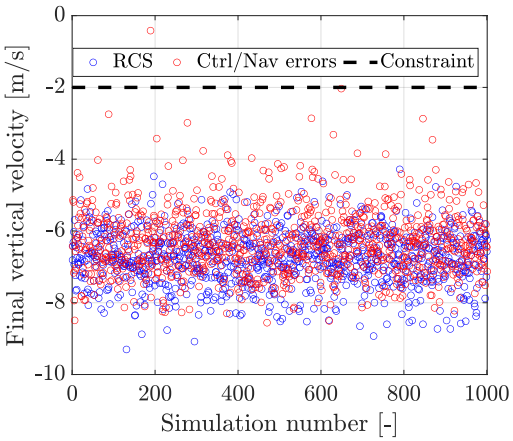
Figure 7.8: Plots policy including control and navigation errors and RCS.



(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.

Figure 7.8: Plots policy including control and navigation errors and RCS. - Concluded

traditional feedback G&C in Chapter 6, the TVC and fin deflections are given as incremental values with respect to a nominal profile.

However, it is considered interesting to understand how the meta-RL policy would perform in a scenario where a only subset of these observations is available to the network. Therefore, two additional trainings are conducted. The first removes only the mass from the observations presented in Eq. (7.2), while the second one excludes both mass and actuator deflections.

To enhance training, the GTrXL block's inputs can be expanded to include actions and rewards from previous time steps. This approach is intended to improve the network's ability to grasp the cause-and-effect relationships between control actions and subsequent states. Nevertheless, it is valuable to evaluate the network's performance without these previous actions and rewards to determine how crucial this information is for the learning process. To test this, previous actions and rewards are excluded, leaving only state observations as inputs to the GTrXL. Removing these components significantly reduces the number of weights and biases. Considering the disastrous effects of having a smaller network from Section 7.3.2, the attention dimension can be increased to maintain a network size comparable to the Baseline.

For instance, when removing previous actions and rewards, if the attention dimension is 128, the number of internal parameters drops from 585,991 to 329,991, while if it is increased to 256, the number rises to 1,282,567. Hence, training with 128 and 256 as attention dimensions are performed without actions and rewards, to assess again the network's size influence.

Table 7.9: Statistics of different trainings removing network's inputs.

Training	Norm Horizontal Position [m]		Norm Horizontal Velocity [m/s]		Vertical Velocity [m/s]		Norm Angular rates [deg/s]		Vertical angle [deg]		Mass Consumption [kg]	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
No mass	0.98	0.40	0.27	0.14	-5.99	0.88	0.63	0.31	1.08	0.42	520.3	12.3
No mass /TVC/fins	1.26	0.76	0.40	0.33	-7.60	2.23	2.20	1.28	1.21	0.87	521.5	29.3
No actions /rewards (256 attn)	1.30	0.66	0.31	0.16	-6.75	0.83	0.74	0.38	0.86	0.43	523.6	19.6
No actions /rewards (128 attn)	2.57	3.08	1.57	2.36	-5.32	3.54	8.08	10.9	5.89	8.94	544.5	29.2

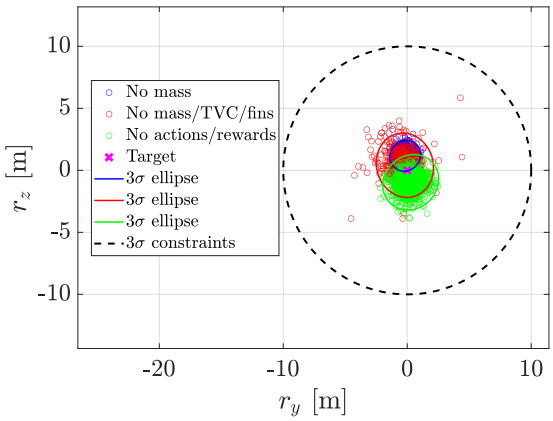
From Fig. 7.9 it can be observed that removing mass (Case "No Mass") alone still meets all terminal constraints, except for vertical velocity, whereas removing both mass and actuator deflections (Case "No mass/TVC/fins") leads to 186 failed simulations, many more than the Baseline. Figures 7.9(b,c,d,f) point out that failures are not only due to violations of terminal angular rates or vertical angle constraints but, in some instances, also because of excessive horizontal velocity or terminal altitude. Moreover, Fig. 7.9(e) highlights that the terminal vertical velocity in this policy is even larger than in the other cases.

The high number of failed runs with violations across all variables suggests that these issues are not due to factors like wind or reward hacking. Instead, the policy appears poorly optimized, likely because the network lacks information about actuator deflections. Commanding six actuators without knowing their deflections strongly degrades the network's ability to make accurate decisions, as it only partially understands the inputs that affect dynamics. Even a small difference of just a few degrees in TVC or fin deflections can strongly affect the thrust and aerodynamic forces and torques generated, potentially compromising the rocket's motion.

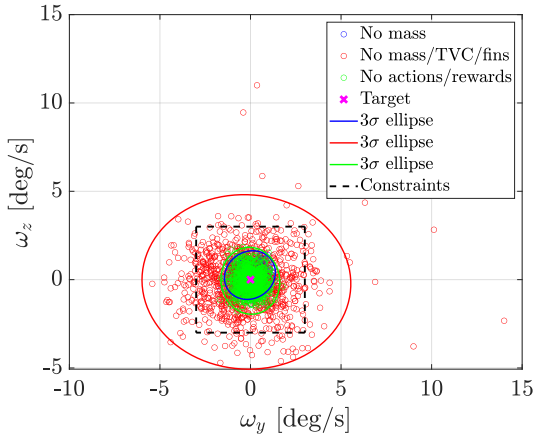
In contrast, when only the current mass is unknown, the network still effectively guides the rocket to the desired final states, as shown in Fig. 7.9(a-f) and Table 7.9. The mass observation directly affects the translation of thrust force into acceleration, and, indirectly, also the torques generated by the controls, through the center of mass and inertia. However, the mass changes by approximately 500 kg (only ∼12.5% of its initial value) throughout the entire flight. Considering minimum thrust (40 kN) and a 500 kg mass difference (from 4000 to 3500 kg) the change in acceleration is only about 1.5 m/s^2. By contrast, a 10-degree change in TVC deflection completely alters the force direction and magnitude. Hence, it can be concluded that knowing the mass is less crucial than the control deflections for the problem's dynamics.

Figures 7.9(a-e) and Table 7.9 also present the results when actions and rewards are removed. Since Table 7.9 already shows that the learned policy is not a good one, results from the training with a dimension of 128 are not illustrated in the plots to allow a clearer visualization of the other three trainings.

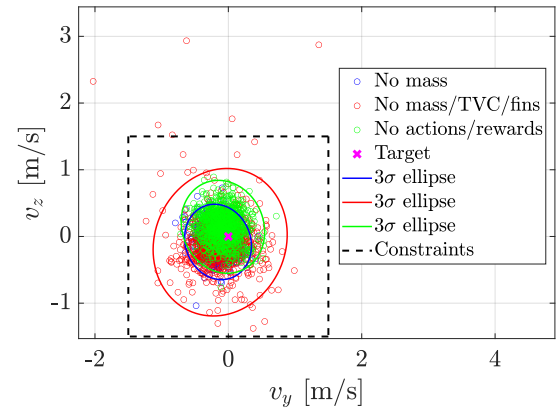
For an attention dimension of 256, excluding previous actions and rewards from the input does not significantly degrade performance. Table 7.9 shows that the mean values are close to the case without mass, and Fig. 7.9(a-e) illustrates that no outliers are present. This suggests that knowledge of previous actions and rewards is not a key factor for training success, as long as the network's size is sufficiently large. The network is able to reconstruct the cause-effect relationships happening throughout the flight, even without previous actions and rewards,



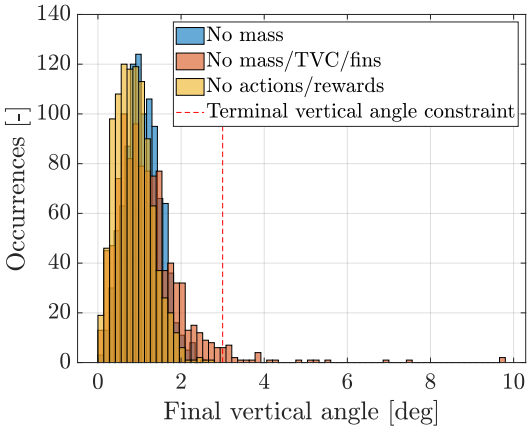
(a) Terminal horizontal position.



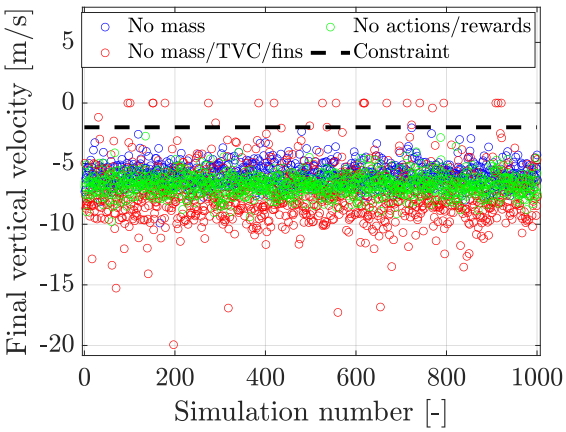
(b) Terminal angular rates.



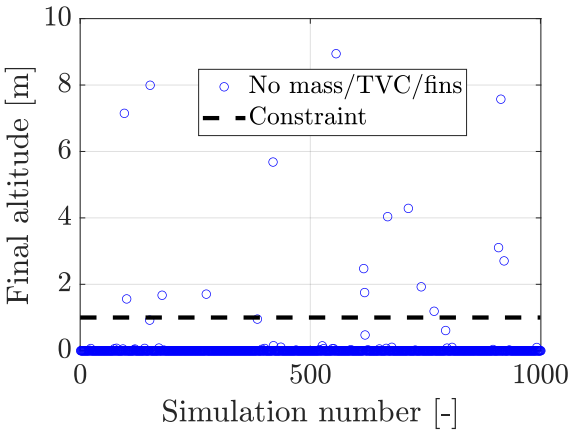
(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.



(f) Terminal altitude.

Figure 7.9: Plots comparison policies without mass, without mass/TVC/fins and actions/rewards.

but it has to be large enough to capture all the features. Indeed, when removing actions and rewards keeping the attention dimension to 128, the network size is too small, and the terminal performances are not sufficiently good. In Table 7.9, this training presents greater mean and standard deviation values for all the terminal variables, as well as a considerable number of failed runs (about 550 out of 1000), and a larger mass consumption. These results reinforce the hypothesis made in Section 7.3.2, stating that the size of GTrXL is crucial.

7.3.7. Improvement terminal velocity

As shown in the previous policies, all the terminal constraints are almost always met, except for the vertical velocity. Additionally, the thrust magnitude at the end of each simulation always terminates to its minimum value, or close to it. To improve the vertical velocity constraint, increasing the thrust in the final meters of flight could be effective. Also in real flights, a specific thrust profile is often enforced during the final phase to ensure precise motion.

The approach implemented in this section is to vary the thrust magnitude, by solving the equations of the uniform accelerated motion at each time step (Eq. (7.21), (7.22) and (7.23)), targeting a certain final velocity and altitude x_{target} and v_{target}. Since the vertical velocity is low (<5 m/s), the aerodynamic forces are assumed to be negligible. Therefore, the rocket acceleration is considered only the sum of the gravitational and thrust accelerations.

$$x_{target} = x_0 + v_0 t + \frac{1}{2} a t^2 \tag{7.21}$$

$$v_{target} = v_0 + at \tag{7.22}$$

$$a = -g + \frac{T}{m} \tag{7.23}$$

Solving this set of equations for time and desired acceleration yield Eq. (7.24), (7.25).

$$t = \frac{-v_0 \pm \sqrt{v_0^2 - 2a(x_0 - x_{target})}}{a} \tag{7.24}$$

$$a = \frac{v_0^2 - v_{target}^2}{2(x_0 - x_{target})} \tag{7.25}$$

Once the vehicle acceleration is found, the commanded thrust magnitude is determined.

$$T = \text{clip}\left(ma + g, 40, 100\right) \text{kN} \tag{7.26}$$

The target altitude is set at 0 meters, but the target velocity must be chosen wisely. The choice of the starting altitude of the terminal guidance is also crucial. Targeting a too low velocity or starting from a too high altitude can make the rocket more susceptible to wind, causing it to drift or tilt in some simulations.

The TVC deflection angles are fixed at zero during the thrust increase for two reasons. First, this thrust patch is applied just a few meters above the ground when the vehicle's angular rates, vertical angle, position, and horizontal velocity are already within final constraints. Since these variables are small and the remaining flight time is just fractions of a second, the rotational motion is barely impacted by having null TVC deflections. Fixing them maintains the vehicle's rotational rates constant from before the application of this patch since the fins have negligible control authority at such low velocities. Hence, if the rocket's rates are sufficiently small before the thrust increases, also the attitude is almost not affected. Secondly, similarly to the issue with LQR closed-loop, the network outputs specific thrust and actuator deflections to generate desired forces and moments. If only the thrust is externally increased they are largely amplified, potentially degrading the rotational motion performance.

Figures 7.10(a-f) illustrate the application of the described patch to the policy trained without mass observation in Section 7.3.6. While several policies were tested, this one is highlighted because it already met all terminal constraints, but the vertical velocity, across all 1000 runs before the patch was applied. If a simulation exceeds one of the other terminal constraints without the patch, it will likely remain a failed run since the patch does not actively control rotational motion. Additionally, this training exhibited a high terminal vertical velocity, making it an ideal choice to demonstrate the patch's effectiveness in challenging scenarios.

Table 7.10: Statistics of patched policy and thrust rate policy.

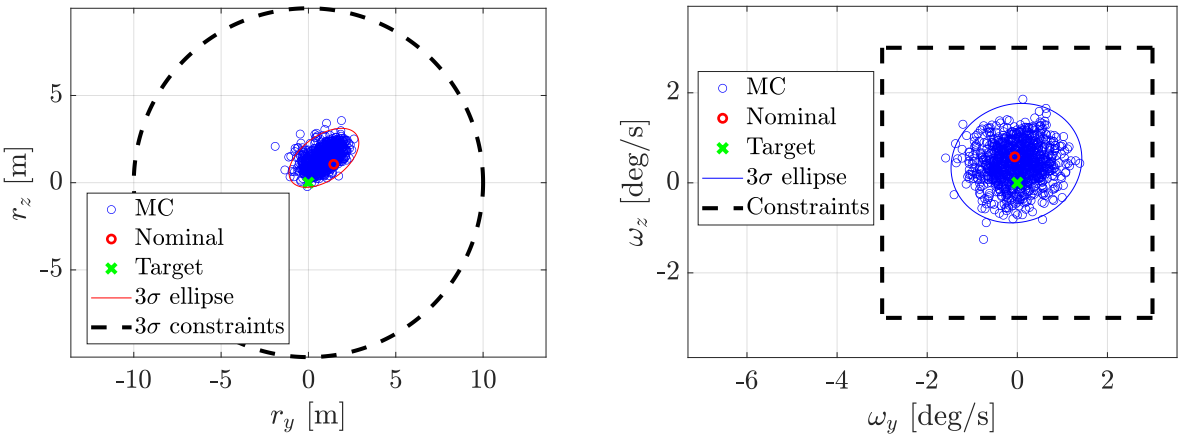
Training	Norm Horizontal Position [m]		Norm Horizontal Velocity [m/s]		Vertical Velocity [m/s]		Norm Angular rates [deg/s]		Vertical angle [deg]		Mass Consumption [kg]	
	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std	Mean	Std
Patched	1.76	0.68	0.34	0.16	-0.75	0.38	0.71	0.34	0.88	0.43	531.3	13.6
Thrust rate	1.01	0.55	0.23	0.12	-3.39	0.80	0.55	0.28	0.82	0.37	536.4	21.7

The correction is applied directly in the Monte Carlo campaign, not requiring additional training. When it is applied, the network's output is substituted by this terminal patch.

Carefully choosing the target velocity and initial altitude is crucial to avoid significant rotational motion degradation. The target vertical velocity is set at -1 m/s, providing a margin below the terminal constraint. A small analysis can determine the correct altitude initiation to ensure all simulations are corrected. With a maximum thrust of 100 kN and an end-of-flight mass of about 3500 kg, the maximum acceleration (thrust minus gravity) is around 18.5 m/s^2. Considering that the policy without mass has a mean landing vertical velocity of about -6 m/s, with a maximum value of -9.5 m/s, Eq. (7.25) can help in finding out a reasonable altitude to correct the entire spectrum of vertical velocities. For instance, if the vehicle has a vertical velocity of -10 m/s at 3 m of altitude, targeting -1 m/s at 0 m requires an acceleration of 16.5 m/s^2, which is within the thrust limit of 18.5 m/s^2. Therefore, for this scenario, 3 m is sufficient to correct a broad range of vertical velocities without affecting other state variables.

Figures 7.10(a-e) illustrate that all 1000 simulations from the Monte Carlo campaign are successful, meeting all terminal requirements, including vertical velocity. Figure 7.10(f) demonstrates the thrust increase in the final seconds of each simulation, thanks to the new patched algorithm, something the neural network could not learn independently. Table 7.10 shows that while all the other mean values are very close to the Baseline, the vertical velocity is significantly improved, staying within the constraint with a margin.

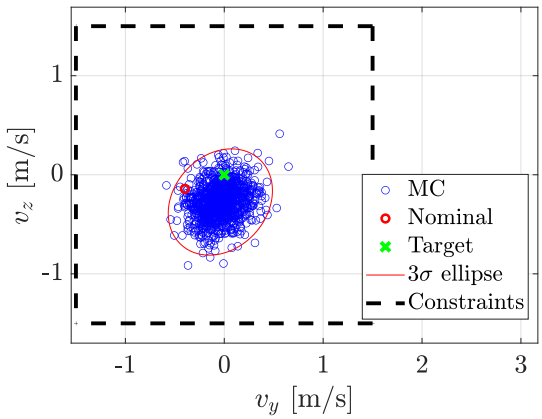
This terminal patch is a preliminary, rudimental solution to the issue of excessive vertical velocity. It is not optimal, and more refined strategies can be implemented. Nevertheless, it proves to be effective in reducing the vertical velocity within the terminal constraint, without worsening other variables. Furthermore, for each training result, the initiation altitude and the target velocity should be finely tuned to avoid negative impacts on other variables.



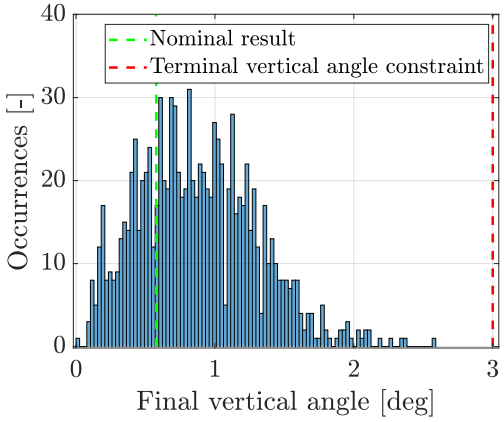
(a) Terminal horizontal position.

(b) Terminal angular rates.

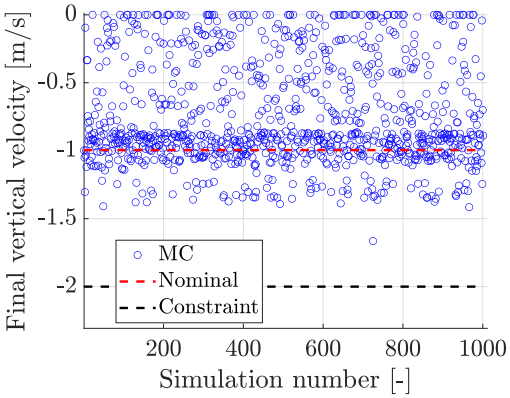
Figure 7.10: Plots patched policy.



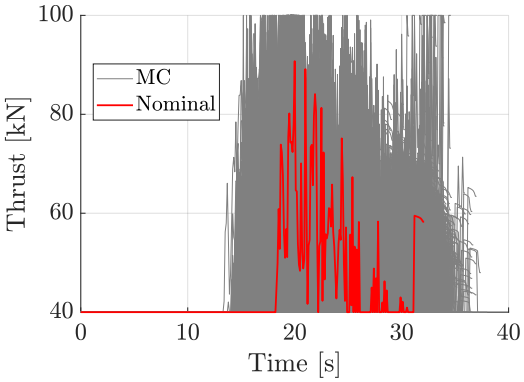
(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.



(f) Thrust magnitude.

Figure 7.10: Plots patched policy. - Concluded

7.3.8. Thrust rate

From the thrust profile presented in Fig. 7.10(f), it can be clearly noticed that it has a high-frequency oscillating trend, from the middle of the flight onward. While this behavior is certainly suboptimal under nominal conditions, it represents what the network considers the best strategy to cope with uncertainties and initial condition dispersions. From the network's point of view, this thrust profile is acceptable, since no constraints are given on how quickly the thrust can change between two time steps. In the problem formulation, no requirement is set on the maximum thrust rate. However, this kind of profile would be extremely challenging to be executed by a real rocket engine due to the totally unrealistic thrust rates required.

Nevertheless, one objective of this thesis is to develop a guidance policy feasible to be implemented in an operational scenario. To address this issue, the thrust magnitude is converted to an additional state, similar to what was done for the other actuators (TVC and fins), and the thrust rate becomes a new control variable. The new solution is implemented by adding Eq. (7.27) to the EOMs, clipping the thrust between 40 kN and 100 kN, and limiting the thrust rate between -10 kN/s and 10 kN/s, which is a reasonable range for this class of engines.

$$\dot{T} = u_{\dot{T}} \tag{7.27}$$

With this new formulation, the learning process now includes a thrust magnitude rate constraint, encouraging the agent to develop a smoother, more realistic thrust profile. It is trained considering all uncertainties, including also the control and navigation ones.

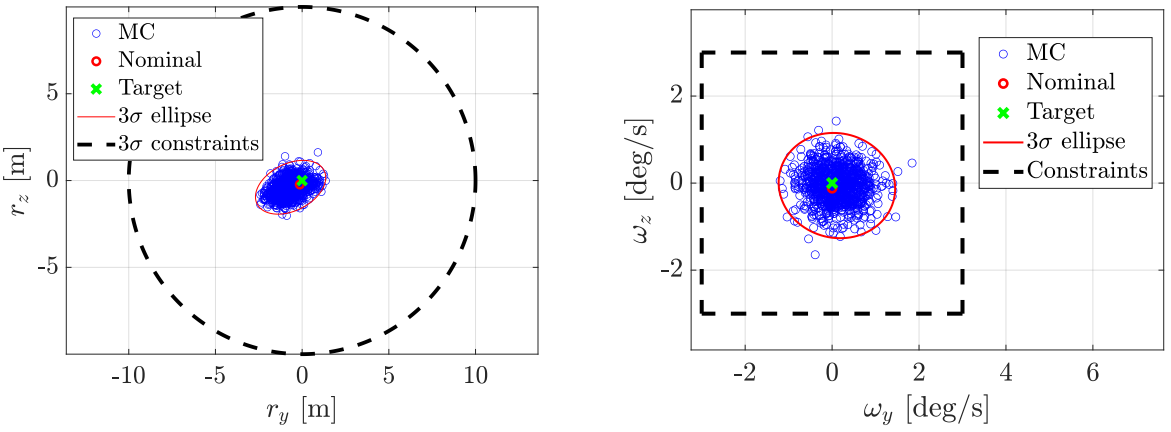
In Fig. 7.11(a-e) and Table 7.10, it can be clearly seen that all the 1000 simulations meet

by far the terminal constraints for position, horizontal velocity, angular rates, and vertical angle, with mean close to zero and small standard deviations indicating the absence of outliers. Hence, it stands out that this is by far the best training done. Even the vertical velocity is significantly improved with respect to any other training, with a mean value slightly below -3 m/s, compared to -5 m/s of the Baseline. Figure 7.11(e) shows that there are also about 70 simulations that land with a vertical velocity between 0 and -2 m/s, thus meeting all the terminal constraints, without the need for any terminal patch. This policy consumes slightly more fuel, primarily due to the extra terminal velocity reduction compared to other trainings.

The exact reason why the network performs better with the inclusion of the thrust rate is not trivial to infer. Previously, the rocket could instantly adjust thrust levels to correct state variables over short periods. Introducing the thrust rate constraint spreads the correction maneuvers over a longer time, making actions have prolonged consequences. The GTrXL architecture effectively captures these relationships during training, optimizing the final policy.

For instance, when a specific thrust level is needed for maneuvers, it takes longer to reach it and return to the minimum level. This potentially helps the reduction of the terminal vertical velocity, because spreads the thrust acceleration over a longer time. Additionally, not having thrust spikes also assists during the training phase. When enforcing the thrust rate constraint, if the thrust is increased too late during the flight, it may happen that the rocket lands with an excessively large vertical velocity, because it does not have enough time to increase the thrust magnitude to an acceptable level. In such a case, less time would be spent in favorable states to take advantage of the position reward, decreasing the final return value. Similarly, if thrust is increased prematurely, the vehicle might achieve zero vertical velocity several meters above the landing site, missing the touchdown reward. Since the agent experiences all these situations during the training, it learns to manage them effectively.

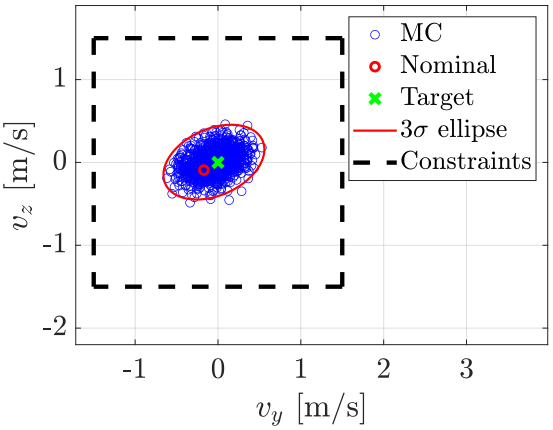
Comparing the nominal thrust profile from this training (Fig. 7.11(f)) to the one from the optimal trajectory, it is evident that they are significantly different. In the optimal trajectory, thrust remains at the minimum value and switches to the maximum for the last 3 seconds of flight, consuming a total of 501.45 kg of fuel. Conversely, in the nominal scenario, the RL policy has a thrust magnitude that never reaches the maximum value, varying between 40 kN and 60 kN for about 15 seconds of flight, consuming a total of 531.77 kg (6% more than OCP). If the rocket would land with 0 m/s the mass consumption would be even higher. Comparing the thrust profile including rate constraint (Fig. 7.11(f)), to the one without it Fig. 7.10(f), it is clear that the latter has some unfeasible local spikes, while the former has a much smoother realistic, never reaching the upper limit.



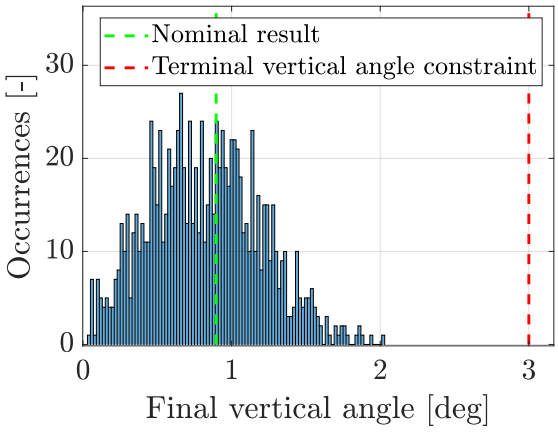
(a) Terminal horizontal position.

(b) Terminal angular rates.

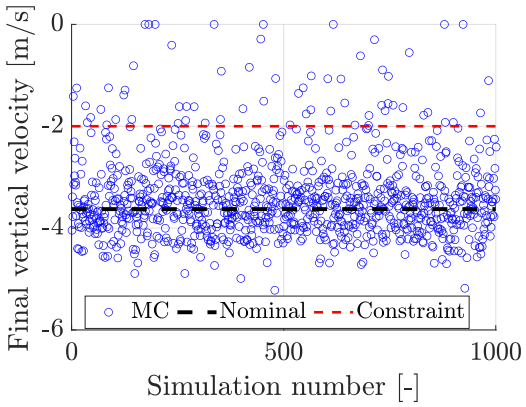
Figure 7.11: Plots thrust rate policy.



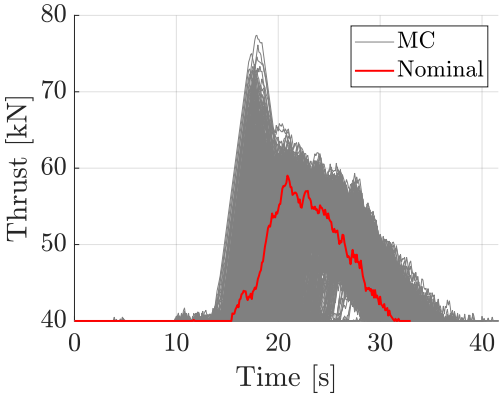
(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.



(f) Thrust profile

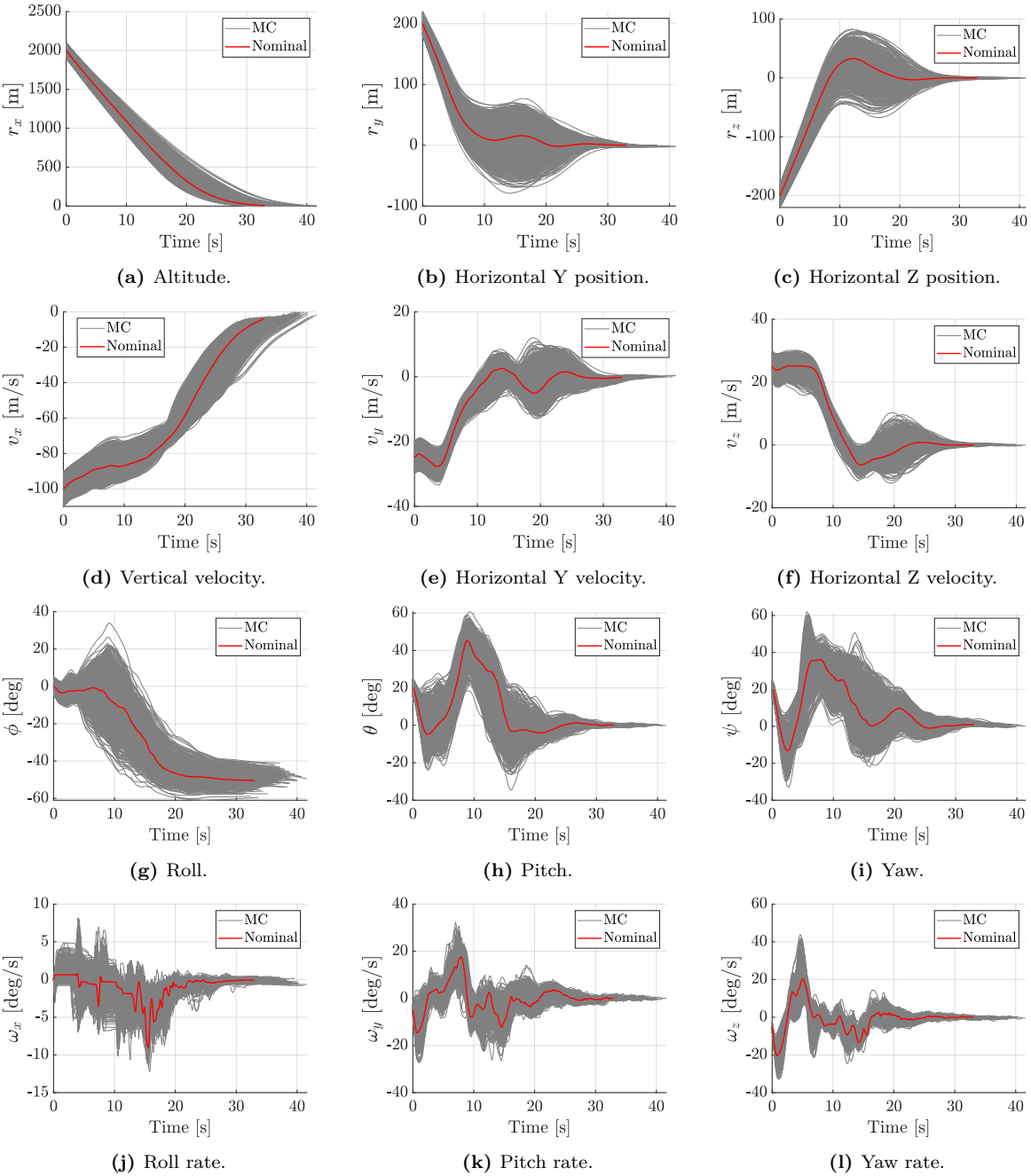
Figure 7.11: Plots thrust rate policy. - Concluded

As presented in Fig. 7.12(b,c,e,f), due to the position reward component, the rocket initially reduces its horizontal position and velocity by positioning itself directly above the landing site, then it descends almost vertically. The 3D trajectories (Fig. 7.12(m)) illustrate this, also highlighting the terminal maneuvers to drive the terminal position close to zero. Such a flight profile is different from the optimal one (Fig. 6.8), but it enables the rocket to use less TVC deflections once it is vertically above the landing site. Hence, instead of largely reducing the velocity at a given point during the flight, the rocket gradually reduces its velocity while descending (Fig. 7.12(d)), to spend more time in regions where it can gain advantages for the cumulative reward. Since the weight of the position component is larger than the mass one, actions are performed to prefer the former to the latter. On the other hand, in the optimal trajectory, there is only the mass objective, so the rocket goes as quickly as possible toward the landing site, increasing the thrust only at the last instant possible to meet terminal constraints.

In Fig. 7.12(j,k,l), it can be observed that the angular rate path constraints of 15 deg/s are locally exceeded, for certain simulations. However, as mentioned in Section 6.3.3, this is not a hard constraint, and local violations are accepted, as long as they are correctly handled by the G&C algorithm. Moreover, the violation is less pronounced than in the traditional closed-loop solution (Fig. 6.8(j,k,i)). Figures 7.12(h,i) illustrate how the rocket, starting from a tilted attitude, initially oscillates around zero, eventually stabilizing into a vertical orientation at about half of the flight time. This behavior is driven by the reward function that gives a bonus any time that the vertical angle is lower than 50 degrees, regardless of the oscillations.

Finally, this policy is tested on a larger set of initial conditions, compared to the ones where it was trained, doubling (or more) the initial dispersion from the nominal trajectory. The exact

set of initial conditions and the plots illustrating the results are shown in Appendix D. 947 out of 1000 simulations are partially successful. The vertical velocity is the only constraint violated, even if typically only by a couple of m/s. Therefore, the RL-based policy can adapt to a significantly extended scenario outside the training conditions. Since the agent has never experienced these states during training, it does not have a consolidated input-output mapping for them, trying to extrapolate the actions to execute. It may take random actions due to the lack of knowledge. Nevertheless, once it enters the known portion of state space, it is able to recover, landing with correct terminal states in most cases. For certain sets of initial states, the discrepancy is too large that the simulations completely diverge. As expected, the more the initial states are enlarged with respect to the training set, the more simulations fail.



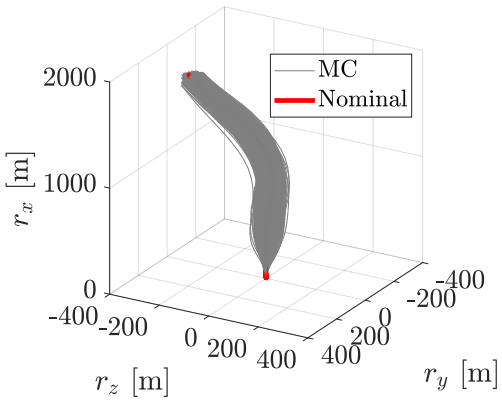
(a) Altitude. **(b)** Horizontal Y position. **(c)** Horizontal Z position.

(d) Vertical velocity. **(e)** Horizontal Y velocity. **(f)** Horizontal Z velocity.

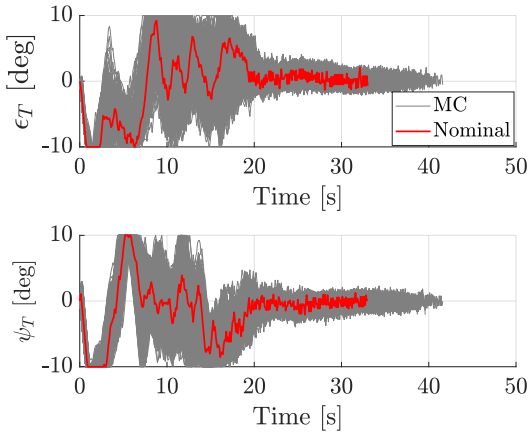
(g) Roll. **(h)** Pitch. **(i)** Yaw.

(j) Roll rate. **(k)** Pitch rate. **(l)** Yaw rate.

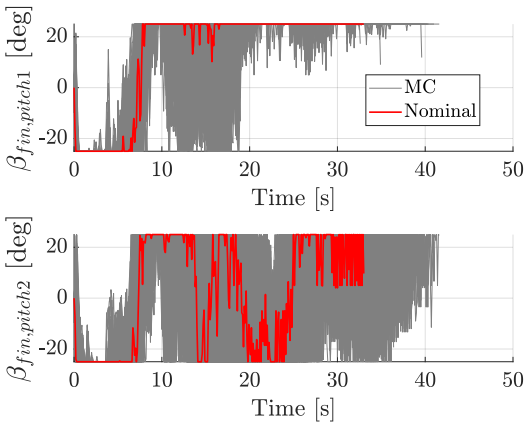
Figure 7.12: RL 1000 Monte Carlo simulations results including thrust rate constraint.



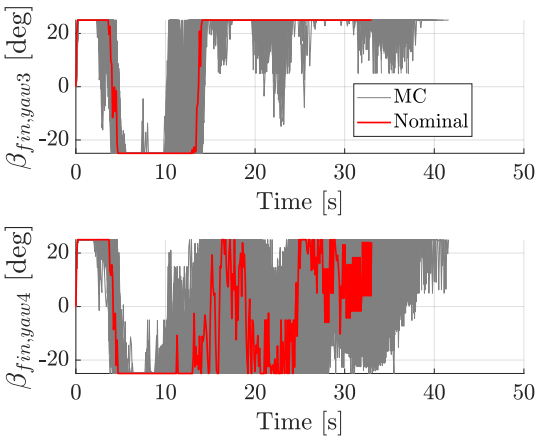
(m) 3D trajectory.



(n) TVC deflections.



(o) Fins pitch pair deflections.



(p) Fins yaw pair deflections.

Figure 7.12: RL 1000 Monte Carlo simulations results including thrust rate constraint. - Concluded

7.4. Power consumption analysis

Figures 7.12(n,o,p) show the nominal profile for TVC and fins deflections. Compared to 6.8(n,o,p), they present a high-frequency oscillation, because the network always tries to correct the rocket's state to achieve a better reward. Since there is no term penalizing the control effort in the reward function, the RL policy respects the deflections and rates constraints, but it does not enforce any additional smoothness. Therefore, a preliminary analysis of the power that they use is performed, to assess the feasibility of these quick and frequent deflections on a real vehicle. The power consumption of each actuator is calculated by multiplying the inertia by the actuator's angular velocity and acceleration rate:

$$P = I\dot{\omega}\ddot{\omega} \tag{7.28}$$

The fins are modeled as thin plates, while the engine as a hollow thin-wall cylinder, calculating their inertia as in Eq. (7.29). Table 7.11 summaries the values used to calculate the inertia, estimated on the vehicle taken in consideration, and similar hardware currently available.

$$I_{fin} = \frac{1}{12} m_{fin} L_y^2 \quad \text{and} \quad I_{engine} = \frac{1}{12} m_{engine} \left[3\left(r_1^2 + r_2^2\right) + h^2\right] \tag{7.29}$$

To understand the total energy consumption (i.e., work) throughout the entire flight, the power consumption (Eq. (7.28)) of two TVC and four fins actuators are integrated over time and

Table 7.11: Fins' and engine's dimensions.

Variable	Value	Unit
m_{fin}	40	kg
L_y	1	m
m_{engine}	250	kg
r_1	0.5	m
r_2	0.48	m
h	1.5	m

summed up. Taking the mean value of the Monte Carlo simulations for the thrust rate RL policy, it yields 21.5 kJ, which is equivalent to about 6 Wh. Out of 1000 simulations, the maximum peak power consumption is about 3.5 kW.

It is reminded that given the modeling of the inertia, the power consumption is only a first estimation of the real energy drained by the actuators. By changing the value of the inertia of the fins and TVC, also the power consumption changes linearly. However, as first approximation is representative of the orders of magnitude of the power consumption. Nevertheless, it is very hard to find real-world missions to compare the numbers. Only the metrics for the Space Shuttle missions are found, which is a significantly larger vehicle. The Space Shuttle can handle power peaks up to 36 kW, and a total energy available larger than 3000 MJ (i.e., 850 kWh) for a 7-day mission (NASA, 1981). Reconverting the total energy to the 40-second simulation of the terminal rocket landing, it yields about 56 Wh available. The Space Shuttle and the vehicle analyzed in this thesis are different, considering the missions and the dimensions. However, the power consumption of the rocket landing is about one order of magnitude less than the Space Shuttle for the peak power, as well as for the total energy. Therefore, it is considered appropriate and available in flight. It could be reduced, by lowering the maximum deflection rate of the fins actuators, which is what drains the most power.

The closed loop solutions, where the LQR tracks an optimal trajectory, drain significantly less energy. While the maximum peak power out of 1000 Monte Carlo simulations is about 3.5 kW (similar to the NN-based results), the total energy consumed is approximately 1 kJ (i.e., 0.3 Wh), which is about 20 times less than the RL solution. The reason for such lower values is due to smoother nominal control profiles and lower effort done by the LQR controller, compared to the frequent corrections applied by the RL policy.

7.5. Lyapunov stability

In G&C algorithms ensuring the stability of the controlled system is crucial for mission success, especially when uncertainties are present. Stability guarantees predictable performance and prevents deviations from the intended trajectory. Classical controllers are typically analyzed for asymptotic stability, often by assessing the negativity of the real part of the eigenvalues of the linearized closed-loop system, as in Lyapunov's Indirect Method (Khalil, 2002).

DRL-based strategies are really attractive in controlling complex non-linear systems, in presence of uncertainties, which are difficult to control with conventional techniques. However, they lack explainability and formal proof of the stability of the policy they generate. To date, it has not been developed yet a rigorous formulation to prove the stability of such a system. Therefore, RL-based controllers are not yet widespread in real-world applications, due to the lack of safety guarantees of the stability of their behavior once deployed. What has been done so far to implement safe reinforcement learning policies, is to train an additional neural, in parallel to the policy network, to synthesize a Lyapunov function, providing a stability guarantee for the system (Manfredi et al., 2022). A less rigorous, but quicker approach, employed by Fereoli et al. (2024), involves checking post-training that a set of trajectories from a Monte Carlo

campaign are asymptotically stable with respect to a given Lyapunov function. In both cases, candidate Lyapunov functions are used, and Lyapunov's Direct Method is used to demonstrate the asymptotic stability of the system (Khalil, 2002).

Differently from the Indirect Method, which applies only to linearized systems, the Direct Method proves whether a nonlinear system is asymptotically stable, without the need of linearizing it around an equilibrium point. This method can be used in the synthesis of Lyapunov-based controllers, as well as to numerically assess the stability of a large set of Monte Carlo simulations. It is remarked that, in the second case, it is not a mathematical proof of the stability of the policy. Nevertheless, if the Monte Carlo set is large enough and representative of real-world scenarios, it can serve as a practical proxy for policy stability.

The definition of Lyapunov stability is: *Considering an autonomous non-linear dynamic system* $\dot{\mathbf{x}} = f(\mathbf{x})$*, and an equilibrium point* $\bar{\mathbf{x}}$ *of the system, for which* $f(\bar{\mathbf{x}}) = \mathbf{0}$*, the system, around that equilibrium point, can be:*

- **Stable**: for any $\epsilon > 0$, there exists $\delta > 0$ such that $\|\mathbf{x}(t_0) - \bar{\mathbf{x}}\| < \delta$ then $\|\mathbf{x}(t) - \bar{\mathbf{x}}\| < \epsilon$ for all $t > t_0$.
- **Asymptotically stable**: if the equilibrium point $\bar{\mathbf{x}}$ is stable, and $\|\mathbf{x}(t) - \bar{\mathbf{x}}\| \to 0$.

The definition describes a local stability around the equilibrium point, but if the limit to zero holds true for the entire state space, the stability is global.

Based on the Lyapunov stability definition, Lyapunov's Direct Method provides a formulation to assess the asymptotic stability of a non-linear system. Given an autonomous non-linear dynamic system $\dot{\mathbf{x}} = f(\mathbf{x})$ and considering an equilibrium point $\bar{\mathbf{x}}$, a system is said to be asymptotically stable if there exists, in a finite neighborhood D of the equilibrium point, a differentiable scalar function $V(\mathbf{x})$ for which the following conditions hold:

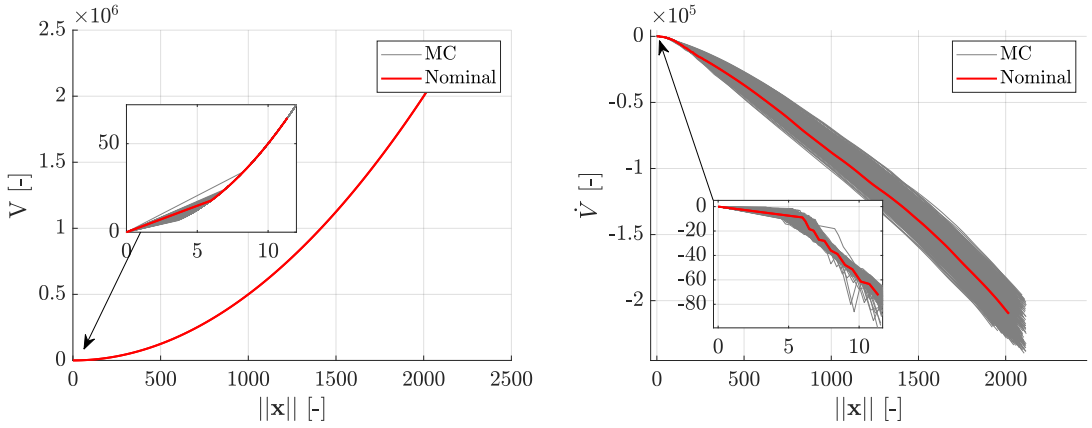
1. $V(\mathbf{x}) > 0, \forall \mathbf{x} \neq \bar{\mathbf{x}} \in D$ and $V(\bar{\mathbf{x}}) = 0$
2. $\dot{V}(\mathbf{x}) < 0, \forall \mathbf{x} \neq \bar{\mathbf{x}} \in D$ and $\dot{V}(\bar{\mathbf{x}}) \leqslant 0$

If D includes all the possible states, then the system is globally asymptotically stable.

For the problem at hand, the equations considered are the EOMs as shown in Eq. 6.5, and the equilibrium point is made of the terminal states.

The candidate Lyapunov function is a positive-definite quadratic function $V(\mathbf{x}) = \mathbf{x}P\mathbf{x}$, where P is a matrix with all zeros, except for the diagonal, which is made of all 0.5.

Figures 7.13(a,b) show the Lyapunov function and its derivative for 1000 simulations run with the Baseline GTrXL-based policy. It stands out that for all simulations the function



(a) Lyapunov candidate function.

(b) Derivative Lyapunov candidate function.

Figure 7.13: Numerical Lyapunov stability results.

is always positive, and it reaches zero in the equilibrium point. Moreover, the derivative is always negative, ensuring the asymptotic stability of the policy. This analysis, combined with the results of the Monte Carlo campaign, indicates that the system is stable even under varying uncertainties in the dynamics and initial conditions, avoiding catastrophic outcomes or instabilities. However, it does not constitute theoretical proof of the global asymptotic stability of the system, as there could still be untested combinations of uncertainties that might lead the rocket to an unstable state. Nonetheless, 1000 uncertain runs are considered a representative set of all the possible combinations encountered in the real world.

7.6. Summary

From the previous sections, it stems out that including the thrust rate constraint yields the best RL policy, fulfilling all the terminal constraints, except for a slight violation of vertical velocity of 1 m/s on average (Section 7.3.8). Nevertheless, it was also demonstrated that the vertical velocity constraint can be met, reaching 100% success rate, if thrust is enforced to increase in the last 3 meters of flight, using uniform accelerated motion (Section 7.3.7).

The RL policies showed excellent robustness to a large set of dispersed initial conditions, uncertainties in density and aerodynamic coefficients, and navigation and control errors, managing to drive the rocket to the landing site (Section 7.3.4). Moreover, the NN presented also robustness to variations in its internal hyperparameters. The only key factor strongly affecting the accuracy of the solution is the attention dimension, which directly regulates the size of the network. Using a small network with not enough internal weights and biases drastically increased the number of failed simulation (Section 7.3.2). This effect was also experienced when removing the previous actions and rewards as input to the network, without increasing the attention dimension accordingly, provoking a reduction of the network's size. It was demonstrated that the GTrXL-based policies perform better than the LSTM-based ones, confirming the theoretical better performance of the first one in complex problems due to its particular architecture (Section 7.3.3).

RCS was introduced to compare with the second controller architecture from the conventional guidance and control strategy. It was discovered that the performance declined compared to the policies without RCS. Likely, the increase in complexity without a corresponding enlargement of the network's size makes it unable to capture all the features (Section 7.3.5).

It was found that the mass observation can be removed from the network's input, without affecting the performance. In contrast, the TVC and fins deflections are essential knowledge for the network to understand the underlying dynamics (Section 7.3.6).

8

Conclusions and recommendations

Section 8.1 gives insights about the findings of this thesis, drawing conclusions on the Reinforcement Learning (RL) approach used for rocket landing, directly addressing the four subquestions stemming from the main one. Hence, an overview of the advantages of using Neural Networks (NNs) and RL is presented. In Section 8.2 the recommendations for future works are laid out, explaining how to tackle the still partially unresolved issues.

8.1. Conclusions

The work developed in this thesis aims to answer the following research question:

Main Research Question

How can Machine Learning improve the robustness of a guidance system with real-time capability, for the powered landing phase in an uncertain (atmospheric) environment?

A direct answer to the research question is that embedding dispersed initial conditions, dynamics, control, and navigation uncertainties into the training of a NN using Meta Reinforcement Learning (meta-RL) yields potentially better performance in rocket landing guidance compared to a conventional feedback Guidance and Control (G&C) method, composed of a Linear Quadratic Regulator (LQR) controller tracking an optimal trajectory.

The key advantage of RL is that the models and the uncertainties are part of the training, directly optimizing a robust policy. Furthermore, the computational burden is only in the training process, which can be very long. However, once deployed in real-time, the forward pass of the NN takes just a few milliseconds, because it is a sort of mapping between states and controls. Therefore, very detailed guidance policies that use complex models close to reality can be run online, meeting onboard computer requirements. Other traditional methods that may outperform RL in simulations, such as model predictive control or predictor-corrector methods, typically introduce simplifications and/or assumptions to achieve fast enough convergence, possibly leading to a degradation of their performance when used in real scenarios.

Research Subquestion 1

What is the best Machine Learning method to be used in a guidance system for an atmospheric powered landing mission scenario?

As found out during the literature study, a Machine Learning (ML) method stands out to be used for atmospheric powered landing guidance. By learning from a potentially unlim-

ited experience, rather than on a limited dataset, RL outperforms Supervised Learning (SL), enabling better generalization and the ability to cope with situations unseen during training.

Meta-RL is the method used to embed uncertainties and dispersed initial conditions in the training process to create a robust policy, while Proximity Policy Optimization (PPO) is the state-of-the-art RL algorithm that successfully trains the network to learn an optimal policy.

Recurrent and attention-based NNs are employed to output the optimal control actions, leveraging current states and a history of past observations. It was initially hypothesized that the latter would perform better than the former due to their ability to learn longer-term dependencies in time series. Indeed, results confirm that attention-based networks like Gated Transformer XL (GTrXL) outperformed recurrent ones like Long-Short Term Memory (LSTM), because they do not compress all the past information in just two internal states, but they represent it with a much larger amount of internal parameters.

For instance, LSTM policy shows consistently higher mean values and many more simulations violating the terminal constraints. While the best GTrXL-based policy reaches 1000/1000 successful runs, the best LSTM-based network stops at 489/1000. Failed runs in LSTM-based architectures exceed the terminal constraints by a huge margin on all the state variables, while the ones failing in GTrXL policies typically deviate by just a few degrees or degrees per second in vertical angle or angular rates.

Research Subquestion 2

How can the training process of a Machine Learning algorithm be improved for a rocket landing guidance system?

Training a NN for optimal landing guidance involves several challenges, with the reward function being the most critical, as it determines what the network learns. Creating a correct reward function for complex problems with many states and controls is hard. Balancing multiple, often conflicting, objectives within a single scalar reward requires extensive trial and error. For instance, emphasizing vertical velocity might improve its accuracy but degrade angular rates. Moreover, even a slightly wrong formulation of a single term can lead to catastrophic situations where the agent maximizes a reward function, which is not the intended one.

A successful approach is to use dense rewards (bonuses/penalties at each step), based on the quality of the observed states, and episodic rewards (given at the episode's end), based on terminal constraint violations. Logarithmic expressions for dense rewards are effective when a state has an attraction point, such as guiding a rocket's position to the target landing site. For episodic rewards, quadratic penalties for terminal violations and logarithmic bonuses for compliance are effective, pushing the network to stay within limits with a margin.

Tuning the GTrXL hyperparameters is another major challenge affecting results quality. Sensitivity analysis shows that the attention dimension significantly impacts the ability to learn an optimal policy. A smaller attention size, such as 64, drastically reduces the partially successful runs to 28/1000, resulting in poor performance and large error dispersions. Similarly, removing previous actions and rewards with an attention dimension of 128 gives a meaningless policy. Both scenarios experience a reduction in network weights and biases: 215,175 and 329,991 compared to 585,991 in the Baseline case, respectively. This suggests that network size (number of weights and biases) is more crucial than attention dimension or previous actions and rewards. For instance, increasing the attention dimension to 256, while excluding previous rewards and actions, makes the internal parameters surge to 1,282,567, yielding good performance with 1000/1000 partially successful runs, where only vertical velocity is violated. Thus, the network size is the most important factor affecting solution quality, rather than other hyperparameters like the number of heads, encoder layers, or sequence and memory length.

Research Subquestion 3

How accurate are the performances of the developed guidance system in terms of mass consumption and terminal pose?

98.7% of the simulations from the Baseline Monte Carlo campaign meet all terminal constraints, except for vertical velocity. The horizontal position has about 1-meter precision, and horizontal velocity is accurate to submeters per second on the y- and z-components. The average landing vertical velocity is around -5 m/s, slightly exceeding the -2 m/s constraint. The vertical angle has a mean value of around 1 degree, with only two simulations violating the 3-degree limit. Angular rates are 1 deg/s on average, with just 13 runs exceeding the 3 deg/s limit. As vertical velocity approaches zero, angular rates increase due to wind sensitivity and efforts to maximize rewards by spending time in high position bonus states. The average mass consumption is about 500 kg, but reducing vertical velocity to zero would increase fuel depletion.

The hyperparameter sensitivity analysis reveals not only that reducing network size negatively impacts performance, but also that tweaking certain values leads to different learned policies that capture various features. By varying the number of heads, sequence and memory length, or encoder layers, the NN focuses on different aspects of the problem, converging to different local minima despite achieving similar cumulative rewards. For instance, some policies, like the one using a sequence and memory length of 125, result in higher terminal vertical velocity while meeting all the rotational constraints. In contrast, the policy using two heads yields lower vertical velocity but occasionally exceeds angular rate and vertical constraints.

Remarkably, the RL policy demonstrates robustness not only to dispersed initial conditions and uncertainties in density, aerodynamic coefficients, and winds but also to additional errors in navigation and control variables. Results highlight that the NN effectively handles these disturbances, confirming that embedding all uncertainties in the training process produces a robust final policy. Similarly to the Baseline, all the terminal variables have mean values close to the target of zero, respecting the terminal constraints in all the 1000 Monte Carlo simulations, except for the vertical velocity, which is -6 m/s on average.

Since mass is typically not measured by onboard sensors, training a NN without it as input demonstrates no performance degradation compared to the Baseline. However, removing inputs like Thrust Vector Control (TVC) and fin deflections leads to a decline in performance, as the NN requires knowledge of the current deflections to accurately command the rates required to generate the desired forces and torques.

To address the high-frequency oscillatory behavior in the thrust profile, which is incompatible with typical rocket engines, a new NN is trained using thrust rate as the control input to limit these oscillations. The results show that position, horizontal velocity, angular rate, and vertical angle performances remain excellent, fulfilling these constraints across 1000 Monte Carlo runs. Remarkably, 67 simulations respect all the constraints, vertical velocity included. Its mean value is improved to approximately -3 m/s, very close to the -2 m/s constraint. The fuel consumption is increased by about 30 kg, reaching a mean value of 536 kg.

To improve the consistently exceeded vertical velocity constraint, a terminal patch is developed by fixing TVC deflections to zero and increasing thrust magnitude in the final meters above ground using uniformly accelerated motion. This simple strategy works very well, effectively reducing the vertical velocity within its constraint when it is the only one violated. This patch does not address other violations, such as those related to rotational motion if they were already present. Results illustrate that all the terminal constraints, including the terminal velocity, are met for all the 1000 runs. This means that 100% success is reached, enhancing the NN policy with terminal guidance in the last 3 meters above ground.

Thus, the meta-RL policy, validated through extensive Monte Carlo campaigns, has proven to be robust to many uncertainties on winds (**GR-12**, **GR-13**), aerodynamic coefficients (**GR-11**), atmospheric density (**GR-10**), as well as navigation and control errors (**GR-14**, **GR-15**), and dispersed initial conditions on every state variable (**MR-01**, **MR-02**, **MR-03**, **MR-04**, **MR-05**). It meets terminal constraints on position (**GR-03**, **GR-04**), horizontal velocity (**GR-06**), angular rates (**GR-09**), and attitude (**GR-07**). While the vertical velocity is slightly violated, a correction can be implemented and successfully applied to fulfill also this requirement (**GR-05**). The policy shows good potential to run in real-time since a NN forward pass outputs the control actions in only 6 ms on average (**GR-01**). This thesis demonstrates that a meta-RL approach can develop a robust G&C policy for rocket landing. This work advances existing literature by combining RL with the innovative GTrXL and incorporating a more complex environment with 6-Degrees Of Freedom (DOF) uncertain dynamics, dispersed initial conditions, wind, aerodynamics, and more accurate vehicle and environment models. Additionally, it leverages a broader range of controls, like thrust magnitude, engine deflections, and aerodynamic fins, as used in real rocket landings.

The results are compared with the most similar RL-based atmospheric rocket landing guidance paper. Rosa et al. (2023) use a 6-DOF high-fidelity environment, using only TVC and thrust magnitude, and starting from 3 km of altitude. Training a fully connected network with Deep Deterministic Policy Gradient (DDPG), they show a 97% success rate in 1000 Monte Carlo runs, considering a 5% and 3% initial dispersion on position and velocity. However, they allow larger terminal constraints, 10 m/s and 3 m/s on vertical and horizontal velocity, or 5 deg on vertical angle. This thesis also employs a similar 6-DOF high-fidelity simulator, including a variable Mass, Center of Mass and Inertia (MCI) model, largely dispersed initial conditions on every state variable, as well as dynamics, control, and navigation uncertainties. Including control rate constraints and additional actuators such as aerodynamic fins, further increases the model's complexity, making it more realistic. If the same terminal constraints as in Rosa et al. (2023) were considered in this thesis, the success rate for many Monte Carlo campaigns would be 100%. Despite stricter constraints defined in this thesis, the results are remarkable, meeting them all in 1000/1000 simulations, except for the vertical velocity, which is resolved by applying a terminal patch, ultimately achieving 100% success rate.

Research Subquestion 4

How does the Machine Learning-based guidance system perform compared to the state-of-the-art solutions in a strongly perturbed environment?

Comparing the RL Baseline policy to the one obtained with the optimal trajectory tracked by the LQR controller, it can be clearly seen that the former outperforms the latter in every single terminal variable, except the vertical velocity. For example, the mean norm of the horizontal position is about 1 m for RL, while it is close to 7 m for the other case. Similarly, the norm of horizontal velocity and angular rates are, on average, about 1.5 m/s and 3.75 deg/s in conventional feedback guidance and control, compared to 0.25 m/s and 1 deg/s in RL Baseline case. Only the vertical angle is similar, while the vertical velocity is better using traditional methods. However, the vertical velocity can be corrected in the RL approach using a terminal guidance patch, leading to excellent performance in all terminal constraints, with RL's best result achieving 1000 successful runs, compared to 390 with LQR.

Mass consumption is similar between the two policies, with a slightly larger value for the RL case. This is expected since the nominal trajectory is optimized only for fuel consumption, while the cost function in RL includes many more terms. For example, comparing the best RL results (i.e. training including thrust rate) with the closed-loop simulations, the former

consumes approximately 6% more fuel.

The conventional approach performs poorly due to the strong interactions between translational and rotational motions, which are controlled separately by two 3-DOF loops. This separation proved to be suboptimal for tightly coupled interactions. For instance, TVC deflections, meant for rotational control, create moments that are also influenced by thrust magnitude, which is intended only for translational control. In contrast, RL effectively accounts for these interactions during training, allowing the NN to learn how specific actions impact both motions simultaneously and to generate policies that consider their combined effects.

Moreover, LQR is not the best control technique to handle so many uncertainties. It uses a linearized model around reference states and controls from a nominal trajectory, without consideration for uncertainties during the synthesis process. Conversely, the RL problem is already trained on the non-linear dynamics, incorporating all the uncertainties and initial dispersions, so that the NN can experience the possible effects and learn how to be robust.

To increase the separation between translational and rotational motion, Reaction Control System (RCS) is added, and TVC deflections are moved to the outer loop. The performances increase, with mean values closer to the target of zero, but 430 simulations still fail. The results are still far from those achieved by RL because the interactions between the inner and outer loop are still present, requiring careful tuning to avoid catastrophic instabilities.

The tuning of the LQR controller is a time-consuming process, primarily due to the need to iteratively adjust the **Q** and **R** matrices to balance state and control deviations. This trial-and-error process often involves many cycles to achieve a successful Monte Carlo campaign, with outcomes highly sensitive to even minor changes in the weight matrices.

Generating an optimal trajectory is also a very challenging part. Using software such as ICLOCS is not as straightforward as it seems, requiring a careful understanding of the effect of many parameters such as tolerances, number of nodes, and transcription methods. The correct settings have to be found and the generation of a single trajectory takes up to several hours.

8.2. Recommendations for future work

The conclusions show that the RL policy is successful and robust to many types of uncertainties, improving the results with respect to an optimal trajectory tracked by an LQR controller. This section focuses on recommendations directly related to improving the current RL policy:

- **Reward function optimization:** As presented in the majority of the policies, the terminal vertical velocity typically violates its constraint, even if by only a few m/s. Even if by adding a terminal patch to correct this issue 100% success rate was achieved, future research could try to integrate the fulfillment of this constraint into the policy generated by the network. To address this, a different expression for the velocity in the reward function could be developed, eliminating the tracking of the exponential profile. For example, a logarithmic bonus could combine the advantages of reducing the velocity while the vehicle gets closer to the landing site.
- **Low-altitude wind modeling:** When the rocket has a low velocity at low altitude it is highly sensitive to the winds. This also happens because the wind in these states is still significantly high. However, it is known that close to the ground, the wind intensity usually decreases. Therefore, implementing an exponential profile that goes from zero velocity at ground level to a steady-state value at a certain altitude could mitigate wind effects during the final descent.
- **Penalization control effort:** In the RL policies, the control profiles for TVC and fin deflection angles present frequent oscillations, due to continuous maneuvers to generate forces and moments to increase the total reward. These profiles respect all the constraints initially set, but a penalization of the control effort could be introduced in the reward

function to encourage the agent to have a smoother control profile, beneficial for real applications.

- **Alternative reward formulation:** It could be beneficial to test a new reward function, using a piecewise formulation made of two terms. The first one would be identical to the one used in this thesis, but instead of the landing site, it would target a position 10 meters above it. Then, the second term would give bonuses for a vertical terminal motion. In this way, the agent would learn to meet all the constraints on rotational and translational motion at the end of the first leg, while it would learn to enforce a vertical motion to drive the vertical velocity to zero in the second part.
- **Stability of the policy:** Lyapunov Direct Method was used to numerically assess the stability of the policy, analyzing results of Monte Carlo simulations. Even if it suggests a stable policy, it is not a theoretical proof. More refined approaches could be explored, such as using an additional network to generate a Lyapunov function to embed stability already during training.

Additional recommendations for further expansion of this research are presented, to enhance the realisticity of the RL policy in an operational scenario.

- **Implementation of Kalman filter:** The navigation errors are introduced as Gaussian noises, but future research could include state estimators like a Kalman filter, to increase the fidelity of the Guidance, Navigation, and Control (GNC) system and RL policy.
- **GPU training:** Implementing the use of GPU in the training process, may be beneficial to speed it up, reducing it from the approximately five days necessary when only the CPU is used.
- **Testing different NN architectures:** It would be interesting to compare the performance of RL with Transformers-based NNs to other NN architectures. For example, any SL technique is discarded in this thesis due to the very long time to generate the optimal trajectories. However, pre-training the network with behavioral cloning may speed up the RL training time. Another interesting option could be the use of Physics-Informed Neural Networks, imitating trajectories while enforcing the dynamics constraints.
- **Comparison with other conventional G&C strategies:** Given the coupling issues arosen with the two 3-DOF controllers, using a unified 6-DOF controller for translational and rotational motion, as done in Sagliano et al. (2023) could improve the performance of the LQR controller. It would be interesting to compare the RL policy with more sophisticated conventional G&C strategies that are more suitable for robustness to uncertainties by design, such as H_∞. Also exploring different guidance methods like Convex Optimization could provide a meaningful comparison with RL.
- **Expansion mission scenario:** Future research could enlarge the initial conditions, including the aerodynamic descent, before the powered landing. It would be interesting to understand whether, using meta-RL, the agent would learn the optimal switch between the aerodynamic and propulsive phases, to achieve a successful landing.
- **Flexible vehicle and fuel sloshing:** Including more complex vehicle models, such as structural bending and fuel sloshing would increase the accuracy of the solution. Therefore, the learned policy would be more realistic, and closer to the one needed in an operational scenario.

Bibliography

Acikmese, B. and Ploen, S. R. "Convex Programming Approach to Powered Descent Guidance for Mars Landing". *Journal of Guidance, Control, and Dynamics*, Vol. 30, No. 5, pp. 1353–1366, Sept. 2007. doi: 10.2514/1.27553. URL: https://arc.aiaa.org/doi/10.2514/1.27553.

Acikmese, B., Casoliva, J., and Carson, J. M. "G-FOLD: A Real-Time Implementable Fuel Optimal Large Divert Guidance Algorithm for Planetary Pinpoint Landing".

ArianeGroup. "1N, 20N, 400N and Heritage Thrusters, Hydrazine Thruster", 2020. [Online; accessed July 29, 2024].

Arulkumaran, K., Deisenroth, M. P., Brundage, M., and Bharath, A. A. "A Brief Survey of Deep Reinforcement Learning". *IEEE Signal Processing Magazine*, Vol. 34, No. 6, pp. 26–38, Nov. 2017. doi: 10.1109/MSP.2017.2743240. URL: http://arxiv.org/abs/1708.05866.

Açıkmeşe, B. and Blackmore, L. "Lossless convexification of a class of optimal control problems with non-convex control constraints". *Automatica*, Vol. 47, No. 2, pp. 341–347, Feb. 2011. doi: 10.1016/j.automatica.2010.10.037. URL: https://linkinghub.elsevier.com/retrieve/pii/S0005109810004516.

Ba, J. L., Kiros, J. R., and Hinton, G. E. "Layer Normalization", 2016. URL: https://arxiv.org/abs/1607.06450.

Beck, J., Vuorio, R., Liu, E. Z., Xiong, Z., Zintgraf, L., Finn, C., and Whiteson, S. "A Survey of Meta-Reinforcement Learning", Jan. 2023. URL: http://arxiv.org/abs/2301.08028.

Bellman, R. *Dynamic programming*. Princeton Univ. Pr, Princeton, NJ, 1984. ISBN 978-0-691-07951-6.

Betts, J. T. *Practical Methods for Optimal Control and Estimation Using Nonlinear Programming*. Society for Industrial and Applied Mathematics, second edition, Jan. 2010. ISBN 978-0-89871-688-7 978-0-89871-857-7. doi: 10.1137/1.9780898718577. URL: https://epubs.siam.org/doi/book/10.1137/1.9780898718577.

Bieniawski, S. R., Lewis, B., Friia, B., Mahajan, A., Somervill, K., Mamidipudi, P., and Dakin, D. "New Shepard Flight Test Results from Blue Origin De-Orbit Descent and Landing Tipping Point". In: *AIAA SCITECH 2022 Forum*, Number 1829, San Diego, CA & Virtual, Jan. 2022. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-631-6. doi: 10.2514/6.2022-1829. URL: https://arc.aiaa.org/doi/10.2514/6.2022-1829.

Blackmore, L. "Autonomous precision landing of space rockets". *Winter Bridge on Frontiers of Engineering*, Vol. 4, No. 46, pp. 15–20, Jan. 2016.

Blackmore, L., Açıkmeşe, B., and Carson, J. M. "Lossless convexification of control constraints for a class of nonlinear optimal control problems". *Systems & Control Letters*, Vol. 61, No. 8, pp. 863–870, Aug. 2012. doi: 10.1016/j.sysconle.2012.04.010. URL: https://linkinghub.elsevier.com/retrieve/pii/S0167691112000874.

Blevins, D. and Hohmann, C. "Description of the Space Shuttle Reaction Control System". In: *11th Propulsion Conference*, Number 1299, Anaheim,CA,U.S.A., Sept. 1975. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.1975-1299. URL: https://arc.aiaa.org/doi/10.2514/6.1975-1299.

Bonasera, S., Bosanac, N., Sullivan, C. J., Elliott, I., Ahmed, N., and McMahon, J. W. "Designing Sun–Earth L2 Halo Orbit Stationkeeping Maneuvers via Reinforcement Learning". *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 2, pp. 301–311, Feb. 2023. doi: 10.2514/1.G006783. URL: https://arc.aiaa.org/doi/10.2514/1.G006783.

Boyd, S. P. and Vandenberghe, L. *Convex optimization*. Cambridge University Press, Cambridge, UK, 2004. ISBN 978-0-521-83378-3.

Briden, J., Gurga, T., Johnson, B. J., Cauligi, A., and Linares, R. "Improving Computational Efficiency for Powered Descent Guidance via Transformer-based Tight Constraint Prediction". In: *AIAA SCITECH 2024 Forum*, Number 1760, Orlando, FL, Jan. 2024. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-711-5. doi: 10.2514/6.2024-1760. URL: https://arc.aiaa.org/doi/10.2514/6.2024-1760.

Bryson, A. "Applied optimal control: Optimization, estimation and control". *ZAMM - Journal of Applied Mathematics and Mechanics / Zeitschrift für Angewandte Mathematik und Mechanik*, Vol. 59, No. 8, p. 402, 1979. doi: https://doi.org/10.1002/zamm.19790590826. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/zamm.19790590826.

Bulirsch, R., Nerz, E., Pesch, H. J., and von Stryk, O. *Combining Direct and Indirect Methods in Optimal Control: Range Maximization of a Hang Glider*, pp. 273–288. Birkhäuser Basel, 1993. ISBN 9783034875394. doi: 10.1007/978-3-0348-7539-4\20.

Chen, Y. and Ma, L. "Rocket Powered Landing Guidance Using Proximal Policy Optimization". In: *Proceedings of the 2019 4th International Conference on Automation, Control and Robotics Engineering*, pp. 1–6, Shenzhen China, July 2019. ACM. ISBN 978-1-4503-7186-5. doi: 10.1145/3351917.3351935. URL: https://dl.acm.org/doi/10.1145/3351917.3351935.

Cheng, L., Wang, Z., Jiang, F., and Li, J. "Fast Generation of Optimal Asteroid Landing Trajectories Using Deep Neural Networks". *IEEE Transactions on Aerospace and Electronic Systems*, Vol. 56, No. 4, pp. 2642–2655, Aug. 2020. doi: 10.1109/TAES.2019.2952700. URL: https://ieeexplore.ieee.org/document/8894850/.

De Oliveira, A. and Lavagna, M. "Reusable launchers re-entry controlled dynamics simulator". In: *Proceedings of the 9th European Conference for Aeronautics and Space Sciences (EUCASS)*, Number 6193, Lille, France, 2022. doi: 10.13009/EUCASS2022-6193. URL: https://www.eucass.eu/doi/EUCASS2022-6193.pdf.

De Oliveira, A. and Lavagna, M. "Development of a Controlled Dynamics Simulator for Reusable Launcher Descent and Precise Landing". *Aerospace*, Vol. 10, No. 12, pp. 993–1022, Nov. 2023. doi: 10.3390/aerospace10120993. URL: https://www.mdpi.com/2226-4310/10/12/993.

Delft High Performance Computing Centre (DHPC). "DelftBlue Supercomputer (Phase 2)". https://www.tudelft.nl/dhpc/ark:/44463/DelftBluePhase2, 2024.

DLR. "Callisto illustration". URL: https://www.dlr.de/en/irs/research-transfer/missions-and-projects/callisto. [Online; accessed June 6, 2024].

Drob, D. P., Emmert, J. T., Meriwether, J. W., Makela, J. J., Doornbos, E., Conde, M., Hernandez, G., Noto, J., Zawdie, K. A., McDonald, S. E., Huba, J. D., and Klenzing, J. H. "An update to the Horizontal Wind Model (HWM): The quiet time thermosphere". *Earth and Space Science*, Vol. 2, No. 7, pp. 301–319, July 2015. doi: 10.1002/2014EA000089. URL: https://agupubs.onlinelibrary.wiley.com/doi/10.1002/2014EA000089.

D'Souza, C. "An optimal guidance law for planetary landing". In: *Guidance, Navigation, and Control Conference*, Number 3709, New Orleans,LA,U.S.A., Aug. 1997. American Institute of Aeronautics and Astronautics. doi: 10.2514/6.1997-3709. URL: https://arc.aiaa.org/doi/10.2514/6.1997-3709.

Dumke, M., Trigo, G. F., Sagliano, M., Saranrittichai, P., and Theil, S. "Design, development, and flight testing of the vertical take-off and landing GNC testbed EAGLE". *CEAS Space Journal*, Vol. 12, No. 1, pp. 97–113, Jan. 2020. doi: 10.1007/s12567-019-00269-5. URL: http://link.springer.com/10.1007/s12567-019-00269-5.

Dumont, E., Ishimoto, S., Tatiossian, P., Klevanski, J., Reimann, B., Ecker, T., Witte, L., Riehmer, J., Sagliano, M., Giagkozoglou Vincenzino, S., Petkov, I., Rotärmel, W., Schwarz, R., Seelbinder, D., Markgraf, M., Sommer, J., Pfau, D., and Martens, H. "CALLISTO: A Demonstrator for Reusable Launcher Key Technologies". *TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES, AEROSPACE TECHNOLOGY JAPAN*, Vol. 19, No. 1, pp. 106–115, 2021. doi: 10.2322/tastj.19.106. URL: https://www.jstage.jst.go.jp/article/tastj/19/1/19_19.106/_article.

Federici, L. *Deep Reinforcement Learning for Robust Spacecraft Guidance and Control*. PhD thesis, Sapienza, Università di Roma, 2022. URL: https://rgdoi.net/10.13140/RG.2.2.32807.93608.

Federici, L. and Furfaro, R. "Meta-Reinforcement Learning with Transformer Networks for Space Guidance Applications". In: *AIAA SCITECH 2024 Forum*, Number 2061, Orlando, FL, Jan. 2024. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-711-5. doi: 10.2514/6.2024-2061. URL: https://arc.aiaa.org/doi/10.2514/6.2024-2061.

Federici, L. and Zavoli, A. "Robust interplanetary trajectory design under multiple uncertainties via meta-reinforcement learning". *Acta Astronautica*, Vol. 214 pp. 147–158, Jan. 2024. doi: 10.1016/j.actaastro.2023.10.018. URL: https://linkinghub.elsevier.com/retrieve/pii/S0094576523005258.

Federici, L., Benedikter, B., and Zavoli, A. "Deep Learning Techniques for Autonomous Spacecraft Guidance During Proximity Operations". *Journal of Spacecraft and Rockets*, Vol. 58, No. 6, pp. 1774–1785, Nov. 2021. doi: 10.2514/1.A35076. URL: https://arc.aiaa.org/doi/10.2514/1.A35076.

Federici, L., Scorsoglio, A., Ghilardi, L., D'Ambrosio, A., Benedikter, B., Zavoli, A., and Furfaro, R. "Image-Based Meta-Reinforcement Learning for Autonomous Guidance of an Asteroid Impactor". *Journal of Guidance, Control, and Dynamics*, Vol. 45, No. 11, pp. 2013–2028, Nov. 2022. doi: 10.2514/1.G006832. URL: https://arc.aiaa.org/doi/10.2514/1.G006832.

Federici, L., Scorsoglio, A., Zavoli, A., and Furfaro, R. "Autonomous Guidance Between Quasiperiodic Orbits in Cislunar Space via Deep Reinforcement Learning". *Journal of Spacecraft and Rockets*, pp. 1–12, Aug. 2023. doi: 10.2514/1.A35747. URL: https://arc.aiaa.org/doi/10.2514/1.A35747.

Fereoli, G., Schaub, H., and Di Lizia, P. "Meta-Reinforcement Learning for Spacecraft Proximity Operations Guidance and Control in Cislunar Space". Feb. 2024. MSc thesis, Politecnico di Milano.

Ferlin, M. "FROG project: small demonstrator for big ambitions in RLV European objectives. Status quo and results". In: *Proceedings of the 9th European Conference for Aeronautics and Space Sciences (EUCASS)*, Number 9718, 2022. doi: 10.13009/EUCASS2022-9718. URL: https://www.eucass.eu/doi/EUCASS2022-9718.pdf. Artwork Size: 20 pages Medium: PDF Publisher: [object Object].

Fujimoto, S., van Hoof, H., and Meger, D. "Addressing Function Approximation Error in Actor-Critic Methods", Oct. 2018. URL: http://arxiv.org/abs/1802.09477. arXiv:1802.09477.

Gaudet, B. and Furfaro, R. "Adaptive pinpoint and fuel efficient mars landing using reinforcement learning". *IEEE/CAA Journal of Automatica Sinica*, Vol. 1, No. 4, pp. 397–411, Oct. 2014. doi: 10.1109/JAS.2014.7004667. URL: http://ieeexplore.ieee.org/document/7004667/.

Gaudet, B., Linares, R., and Furfaro, R. "Adaptive guidance and integrated navigation with reinforcement meta-learning". *Acta Astronautica*, Vol. 169 pp. 180–190, Apr. 2020a. doi: 10.1016/j.actaastro.2020.01.007. URL: https://linkinghub.elsevier.com/retrieve/pii/S0094576520300072.

Gaudet, B., Linares, R., and Furfaro, R. "Deep reinforcement learning for six degree-of-freedom planetary landing". *Advances in Space Research*, Vol. 65, No. 7, pp. 1723–1741, Apr. 2020b. doi: 10.1016/j.asr.2019.12.030. URL: https://linkinghub.elsevier.com/retrieve/pii/S0273117719309305.

Gaudet, B., Drozd, K., and Furfaro, R. "Adaptive Approach Phase Guidance for a Hypersonic Glider via Reinforcement Meta Learning". In: *AIAA SCITECH 2022 Forum*, Number 2214, San Diego, CA & Virtual, Jan. 2022. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-631-6. doi: 10.2514/6.2022-2214. URL: https://arc.aiaa.org/doi/10.2514/6.2022-2214.

Goodfellow, I., Bengio, Y., and Courville, A. *Deep learning*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, 2016. ISBN 978-0-262-03561-3.

Guo, Y., Hawkins, M., and Wie, B. "Optimal feedback guidance algorithms for planetary landing and asteroid intercept". Jan. 2012. URL: https://www.researchgate.net/publication/265352837_Optimal_feedback_guidance_algorithms_for_planetary_landing_and_asteroid_intercept.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", 2018. URL: https://arxiv.org/abs/1801.01290. doi: 10.48550/ARXIV.1801.01290.

Hochreiter, S. and Schmidhuber, J. "Long Short-Term Memory". *Neural Computation*, Vol. 9, No. 8, pp. 1735–1780, Nov. 1997. doi: 10.1162/neco.1997.9.8.1735. URL: https://direct.mit.edu/neco/article/9/8/1735-1780/6109.

Huneker, L., Sagliano, M., and Arslantas, Y. "Spartan: An improved global pseudospectral algorithm for high-fidelity entry-descent-landing guidance analysis". In: *30th International Symposium on Space Technology and Science, Kobe, Japan, 2015*, 2015.

Izzo, D., Märtens, M., and Pan, B. "A survey on artificial intelligence trends in spacecraft guidance dynamics and control". *Astrodynamics*, Vol. 3, No. 4, pp. 287–299, Dec. 2019. doi: 10.1007/s42064-018-0053-6. URL: https://link.springer.com/10.1007/s42064-018-0053-6.

JAXA. "RVT-9 JAXA Flight Experiment Successful", 2007. URL: https://www.youtube.com/watch?v=Ogm5w6RxS3A. [Online; accessed April 1, 2024].

Justus, C. G., Campbell, C. W., Doubleday, M. K., and Johnson, D. L. "New Atmospheric Turbolence Model for Shuttle Applications", Jan. 1990.

Kelly, M. "An Introduction to Trajectory Optimization: How to Do Your Own Direct Collocation". *SIAM Review*, Vol. 59, No. 4, pp. 849–904, Jan. 2017. doi: 10.1137/16M1062569. URL: https://epubs.siam.org/doi/10.1137/16M1062569.

Khalil, H. K. *Nonlinear systems. Hauptbd.* Prentice Hall, Upper Saddle River, NJ, 3. ed edition, 2002. ISBN 978-0-13-067389-3.

Klumpp, A. R. "Apollo lunar descent guidance". *Automatica*, Vol. 10, No. 2, pp. 133–146, Mar. 1974. doi: 10.1016/0005-1098(74)90019-3. URL: https://linkinghub.elsevier.com/retrieve/pii/0005109874900193.

Kruger, J., Wallace, K., Koenig, A. W., and D'Amico, S. "Autonomous Angles-Only Navigation for Spacecraft Swarms around Planetary Bodies". In: *2021 IEEE Aerospace Conference (50100)*, pp. 1–20, Big Sky, MT, USA, Mar. 2021. IEEE. ISBN 978-1-72817-436-5. doi: 10.1109/AERO50100.2021.9438363. URL: https://ieeexplore.ieee.org/document/9438363/.

Launius, D. R. and Jenkins, R. D. *Coming Home: Reentry and Recovery from Space*. NASA Aeronautics Book Series, 2012.

Leahy, F. "Discrete Gust Model for Launch Vehicle Assessments". In: *AMS Annual Meeting 12th Conference on Aviation, Range and Aerospace Meteorology*, Jan. 2008.

Lee, S.-D. and Lee, C.-H. "Multi-phase and dual aero/propulsive rocket landing guidance using successive convex programming". *Proceedings of the Institution of Mechanical Engineers, Part G: Journal of Aerospace Engineering*, Vol. 237, No. 8, pp. 1816–1834, Nov. 2022. doi: 10.1177/09544100221138350. URL: http://journals.sagepub.com/doi/10.1177/09544100221138350.

Liang, E., Liaw, R., Moritz, P., Nishihara, R., Fox, R., Goldberg, K., Gonzalez, J. E., Jordan, M. I., and Stoica, I. "RLlib: Abstractions for Distributed Reinforcement Learning", June 2018. URL: http://arxiv.org/abs/1712.09381. doi: arXiv:1712.09381.

Lillicrap, T. P., Hunt, J. J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D. "Continuous control with deep reinforcement learning", July 2019. URL: http://arxiv.org/abs/1509.02971. doi: arXiv:1509.02971.

Lu, P. "Propellant-Optimal Powered Descent Guidance". *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 4, pp. 813–826, Apr. 2018. doi: 10.2514/1.G003243. URL: https://arc.aiaa.org/doi/10.2514/1.G003243.

Lu, P., Forbes, S., and Baldwin, M. "A Versatile Powered Guidance Algorithm". In: *AIAA Guidance, Navigation, and Control Conference*, Number 4843, Minneapolis, Minnesota, Aug.

2012. American Institute of Aeronautics and Astronautics. ISBN 978-1-60086-938-9. doi: 10.2514/6.2012-4843. URL: https://arc.aiaa.org/doi/10.2514/6.2012-4843.

Malyuta, D., Reynolds, T. P., Szmuk, M., Lew, T., Bonalli, R., Pavone, M., and Açıkmeşe, B. "Convex Optimization for Trajectory Generation: A Tutorial on Generating Dynamically Feasible Trajectories Reliably and Efficiently". *IEEE Control Systems*, Vol. 42, No. 5, pp. 40–113, Oct. 2022. doi: 10.1109/MCS.2022.3187542. URL: https://ieeexplore.ieee.org/document/9905530/.

Manfredi, G., De Cicco, L., and Mascolo, S. "On Asymptotic Stability of Nonlinear Systems with Deep Reinforcement Learning Controllers". In: *2022 30th Mediterranean Conference on Control and Automation (MED)*, pp. 306–311, Vouliagmeni, Greece, June 2022. IEEE. ISBN 978-1-66540-673-4. doi: 10.1109/MED54222.2022.9837155. URL: https://ieeexplore.ieee.org/document/9837155/.

Markley, F. L. and Crassidis, J. L. *Fundamentals of spacecraft attitude determination and control*. Number 33 in Space technology library. Springer, New York, 2014. ISBN 978-1-4939-0801-1. OCLC: ocn882605422.

Marwege, A., Gülhan, A., Klevanski, J., Hantz, C., Karl, S., Laureti, M., De Zaiacomo, G., Vos, J., Jevons, M., Thies, C., Krammer, A., Lichtenberger, M., Carvalho, J., and Paixão, S. "Retalt: review of technologies and overview of design changes". *CEAS Space Journal*, Vol. 14 pp. 433–445, 06 2022. doi: 10.1007/s12567-022-00458-9.

McDonough, G. *Wind effects on launch vehicles*. NATO, 1970.

Mendeck, G. and Craig, L. "Entry Guidance for the 2011 Mars Science Laboratory Mission". In: *AIAA Atmospheric Flight Mechanics Conference*, Number 6639, Portland, Oregon, Aug. 2011. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-153-3. doi: 10.2514/6.2011-6639. URL: https://arc.aiaa.org/doi/10.2514/6.2011-6639.

Mooij, E. *The motion of a vehicle in a planetary atmosphere*. Delft University of Technology, Faculty of Aerospace Engineering, Delft, 1994. ISBN 978-90-5623-003-6. OCLC: 69026039.

Mooij, E. *Linear quadratic regulator design for an unpowered, winged re-entry vehicle*. Delft University Press, Delft, 1998. ISBN 978-90-407-1597-6. OCLC: 41476493.

Mooij, E. "Re-entry Guidance for Path-Constraint Tracking". In: *AIAA Guidance, Navigation, and Control Conference*, Number 1265, Grapevine, Texas, Jan. 2017. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-450-3. doi: 10.2514/6.2017-1265. URL: https://arc.aiaa.org/doi/10.2514/6.2017-1265.

Mooij, E. *Re-entry System, Lecture Notes*. Delft University of Technology, Faculty of Aerospace Engineering, 2019.

NASA. "Space Shuttle news reference". Technical report, NASA, 1981.

Nelessen, A., Sackier, C., Clark, I., Brugarolas, P., Villar, G., Chen, A., Stehura, A., Otero, R., Stilley, E., Way, D., Edquist, K., Mohan, S., Giovingo, C., and Lefland, M. "Mars 2020 Entry, Descent, and Landing System Overview". In: *2019 IEEE Aerospace Conference*, pp. pp. 1–20, Big Sky, MT, USA, Mar. 2019. IEEE. ISBN 978-1-5386-6854-2. doi: 10.1109/AERO.2019.8742167. URL: https://ieeexplore.ieee.org/document/8742167/.

Nie, Y., Faqir, O., and Kerrigan, E. "ICLOCS2: Solve your optimal control problems with less pain", 2018.

NOOA, USAF, and NASA. "US. Standard Atmosphere, 1976". Technical report, NOOA and USAF and NASA, 1976.

Parisotto, E., Song, H. F., Rae, J. W., Pascanu, R., Gulcehre, C., Jayakumar, S. M., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., Botvinick, M. M., Heess, N., and Hadsell, R. "Stabilizing Transformers for Reinforcement Learning", Oct. 2019. URL: http://arxiv.org/abs/1910.06764. doi: arXiv:1910.06764.

Patterson, M. A. and Rao, A. V. "GPOPS-II: A MATLAB Software for Solving Multiple-Phase Optimal Control Problems Using hp-Adaptive Gaussian Quadrature Collocation Methods and Sparse Nonlinear Programming". *ACM Transactions on Mathematical Software*, Vol. 41 , No. 1, pp. 1–37, Oct. 2014. doi: 10.1145/2558904. URL: https://dl.acm.org/doi/10.1145/2558904.

Petit, G. and Luzum, B. "IERS Conventions (2010)". *Tech. Rep. DTIC Document*, Vol. 36, Jan. 2010.

Rao, A. "A Survey of Numerical Methods for Optimal Control". Jan. 2010.

Rosa, P., Fernandes Vasconcelos, J., Somma, N., Botelho, A., Tofanelli, G., Ignacio Bravo, J., Hinz, R., Belhadj, J., Casasco, M., and Bennani, S. "Deep Reinforcement Learning based Integrated Guidance and Control for a Launcher Landing Problem". In: *Papers of ESA GNC-ICATT 2023*. ESA, July 2023. doi: 10.5270/esa-gnc-icatt-2023-145. URL: https://www.esa-gnc.eu/paper?paper_id=23145.

Sagliano, M. "Performance analysis of linear and nonlinear techniques for automatic scaling of discretized control problems". *Operations Research Letters*, Vol.42, No. 3, pp. 213–216, 2014. doi: 10.1016/j.orl.2014.03.003.

Sagliano, M. *Development of a Novel Algorithm for High Performance Reentry Guidance*. PhD thesis, Fachbereich 4, ProduktionsTechnik, Bremen Universit!"at, 2016. URL: http://elib.suub.uni-bremen.de/edocs/00105082-1.pdf.

Sagliano, M. "Pseudospectral Convex Optimization for Powered Descent and Landing". *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 2, pp. 320–334, Feb. 2018. doi: 10.2514/1.G002818. URL: https://arc.aiaa.org/doi/10.2514/1.G002818.

Sagliano, M., Theil, S., Bergsma, M., D'Onofrio, V., Whittle, L., and Viavattene, G. "On the Radau pseudospectral method: theoretical and implementation advances". *CEAS Space Journal*, Vol. 9, No. 3, pp. 313–331, Sept. 2017. doi: 10.1007/s12567-017-0165-5. URL: http://link.springer.com/10.1007/s12567-017-0165-5.

Sagliano, M., Tsukamoto, T., Maces-Hernandez, J. A., Seelbinder, D., Ishimoto, S., and Dumont, E. "Guidance and Control Strategy for the CALLISTO Flight Experiment". In: *8th European Conference for Aeronautics and Space Sciences (EUCASS).*, Number 284, Madrid, Spain, 2019. doi: 10.13009/EUCASS2019-284. URL: https://www.eucass.eu/doi/EUCASS2019-0284.pdf.

Sagliano, M., Heidecker, A., Macés Hernández, J., Farì, S., Schlotterer, M., Woicke, S., Seelbinder, D., and Dumont, E. "Onboard Guidance for Reusable Rockets: Aerodynamic Descent and Powered Landing". In: *AIAA Scitech 2021 Forum*, Number 0862, VIRTUAL EVENT, Jan. 2021a. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-609-5. doi: 10.2514/6.2021-0862. URL: https://arc.aiaa.org/doi/10.2514/6.2021-0862.

Sagliano, M., Seelbinder, D., and Theil, S. "SPARTAN: Rapid Trajectory Analysis via Pseudospectral Methods". Virtual Event, June 2021b.

Sagliano, M., Seelbinder, D., Theil, S., Im, S., Lee, J., and Lee, K. "Booster Dispersion Area Management through Aerodynamic Guidance and Control". In: *AIAA SCITECH 2022 Forum*, Number 0759, San Diego, CA & Virtual, Jan. 2022. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-631-6. doi: 10.2514/6.2022-0759. URL: https://arc.aiaa.org/doi/10.2514/6.2022-0759.

Sagliano, M., Hernández, J. A. M., Farì, S., Heidecker, A., Schlotterer, M., Woicke, S., Seelbinder, D., Krummen, S., and Dumont, E. "Unified-Loop Structured H-Infinity Control for Aerodynamic Steering of Reusable Rockets". *Journal of Guidance, Control, and Dynamics*, Vol. 46, No. 5, pp. 815–837, May 2023. doi: 10.2514/1.G007077. URL: https://arc.aiaa.org/doi/10.2514/1.G007077.

Sagliano, M., Heidecker, A., Farì, S., Jose Alfredo, M. H., Schlotterer, M., Woicke, S., Seelbinder, D., and Dumont, E. "Powered Atmospheric Landing Guidance for Reusable Rockets: the CALLISTO studies". In: *AIAA SCITECH 2024 Forum*, Number 1761, Orlando, FL, Jan. 2024a. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-711-5. doi: 10.2514/6.2024-1761. URL: https://arc.aiaa.org/doi/10.2514/6.2024-1761.

Sagliano, M., Lu, P., Johnson, B., Seelbinder, D., and Theil, S. "Six-degrees-of-freedom aeropropulsive entry trajectory optimization". In: *AIAA SCITECH 2024 Forum*, Number 1171. American Institute of Aeronautics and Astronautics, Jan. 2024b. doi: 10.2514/6.2024-1171.

Sagliano, M., Seelbinder, D., Theil, S., and Lu, P. "Six-degree-of-freedom rocket landing optimization via augmented convex–concave decomposition". *Journal of Guidance, Control, and Dynamics*, 47, No. 1, pp. 20–35, Jan. 2024c. doi: 10.2514/1.g007570.

Scharf, D. P., Açıkmeşe, B., Dueri, D., Benito, J., and Casoliva, J. "Implementation and Experimental Demonstration of Onboard Powered-Descent Guidance". *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, pp. 213–229, Feb. 2017. doi: 10.2514/1.G000399. URL: https://arc.aiaa.org/doi/10.2514/1.G000399.

Schulman, J., Levine, S., Moritz, P., Jordan, M. I., and Abbeel, P. "Trust Region Policy Optimization", 2015. URL: https://arxiv.org/abs/1502.05477.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. "Proximal Policy Optimization Algorithms", Aug. 2017. URL: http://arxiv.org/abs/1707.06347. doi: arXiv:1707.06347.

Schwarz, R., Solari, M., Razgus, B., Dumke, M., Markgraf, M., Körner, M., Pfau, D., Reigenborn, M., Braun, B., and Sommer, J. "Preliminary Design of the Hybrid Navigation System (HNS) for the CALLISTO RLV Demonstrator". In: *8th European Conference for Aeronautics and Space Sciences (EUCASS)*., Madrid, Spain, July 2019.

Scorsoglio, A., D'Ambrosio, A., Ghilardi, L., Gaudet, B., Curti, F., and Furfaro, R. "Image-Based Deep Reinforcement Meta-Learning for Autonomous Lunar Landing". *Journal of Spacecraft and Rockets*, Vol. 59, No. 1, pp. 153–165, Jan. 2022. doi: 10.2514/1.A35072. URL: https://arc.aiaa.org/doi/10.2514/1.A35072.

Shirobokov, M., Trofimov, S., and Ovchinnikov, M. "Survey of machine learning techniques in spacecraft control design". *Acta Astronautica*, Vol. 186 pp. 87–97, Sept. 2021. doi: 10.

1016/j.actaastro.2021.05.018. URL: https://linkinghub.elsevier.com/retrieve/pii/S0094576521002514.

Silvestrini, S. and Lavagna, M. "Deep Learning and Artificial Neural Networks for Spacecraft Dynamics, Navigation and Control". *Drones*, Vol. 6, No. 10, p. 270, Sept. 2022. doi: 10.3390/drones6100270. URL: https://www.mdpi.com/2504-446X/6/10/270.

Simplício, P., Marcos, A., Joffre, E., Zamaro, M., and Silva, N. "Review of guidance techniques for landing on small bodies". *Progress in Aerospace Sciences*, Vol. 103 pp. 69–83, Nov. 2018. doi: 10.1016/j.paerosci.2018.10.005. URL: https://linkinghub.elsevier.com/retrieve/pii/S0376042118300101.

Simplício, P., Marcos, A., and Bennani, S. "Guidance of Reusable Launchers: Improving Descent and Landing Performance". *Journal of Guidance, Control, and Dynamics*, Vol. 42, No. 10, pp. 2206–2219, Oct. 2019. doi: 10.2514/1.G004155. URL: https://arc.aiaa.org/doi/10.2514/1.G004155.

Simplício, P., Marcos, A., and Bennani, S. "Reusable Launchers: Development of a Coupled Flight Mechanics, Guidance, and Control Benchmark". *Journal of Spacecraft and Rockets*, Vol. 57, No. 1, pp. 74–89, Jan. 2020. doi: 10.2514/1.A34429. URL: https://arc.aiaa.org/doi/10.2514/1.A34429.

Song, Z.-y., Wang, C., Theil, S., Seelbinder, D., Sagliano, M., Liu, X.-f., and Shao, Z.-j. "Survey of autonomous guidance methods for powered planetary landing". *Frontiers of Information Technology & Electronic Engineering*, Vol. 21, No. 5, pp. 652–674, May 2020. doi: 10.1631/FITEE.1900458. URL: http://link.springer.com/10.1631/FITEE.1900458.

SpaceX. "Spacex, dragon capsule", a. URL: https://www.spacex.com/vehicles/dragon/. [Online; accessed July 25, 2024].

SpaceX. "Falcon 9 first stage trajectory", b. URL: https://www.spacex.com/mission/. [Online; accessed August 12, 2024].

SpaceX. "Crs-13 mission, falcon 9 landing", 2017. URL: https://www.flickr.com/photos/spacex/39051469552/. [Online; accessed February 18, 2024].

SpaceX. "Falcon 9 user guide". Technical report, SpaceX, 2021. URL: https://www.spacex.com/media/falcon-users-guide-2021-09.pdf. [Online; accessed July 19, 2024].

SpaceX. "Starship takeoff", 2023. URL: https://x.com/SpaceX/status/1726386449170501783/photo/1. [Online; accessed May 25, 2024].

Spada, F., Sagliano, M., and Topputo, F. "Direct–indirect hybrid strategy for optimal powered descent and landing". *Journal of Spacecraft and Rockets*, Vol. 60, No. 6, pp. 1–18, July 2023. doi: 10.2514/1.a35650. URL: https://doi.org/10.2514/1.A35650.

Steinfeldt, B. A., Grant, M. J., Matz, D. A., Braun, R. D., and Barton, G. H. "Guidance, Navigation, and Control System Performance Trades for Mars Pinpoint Landing". *Journal of Spacecraft and Rockets*, Vol. 47, No. 1, pp. 188–198, Jan. 2010. doi: 10.2514/1.45779. URL: https://arc.aiaa.org/doi/10.2514/1.45779.

Su, L., Wang, J., Ma, Z., and Chen, H. "Real-time Guidance for Powered Landing of Reusable Rockets via Deep Reinforcement Learning". In: *2022 IEEE International Conference on*

Unmanned Systems (ICUS), pp. 214–219, Guangzhou, China, Oct. 2022. IEEE. ISBN 978-1-66548-456-5. doi: 10.1109/ICUS55513.2022.9986540. URL: https://ieeexplore.ieee.org/document/9986540/.

Subirana, J. S., Zornoza, J., and Hernández-Pajares, M. "Conventional Celestial Reference System". Technical report, Technical University of Catalonia, Spain., 2011. URL: https://gssc.esa.int/navipedia/index.php/Conventional_Celestial_Reference_System.

Sutton, R. S. and Barto, A. *Reinforcement learning: an introduction*. Adaptive computation and machine learning. The MIT Press, Cambridge, Massachusetts, nachdruck edition, 2018. ISBN 978-0-262-19398-6.

Szmuk, M. and Acikmese, B. "Successive Convexification for 6-DoF Mars Rocket Powered Landing with Free-Final-Time". In: *2018 AIAA Guidance, Navigation, and Control Conference*, Number 0617, Kissimmee, Florida, Jan. 2018. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-526-5. doi: 10.2514/6.2018-0617. URL: https://arc.aiaa.org/doi/10.2514/6.2018-0617.

Szmuk, M., Acikmese, B., and Berning, A. W. "Successive Convexification for Fuel-Optimal Powered Landing with Aerodynamic Drag and Non-Convex Constraints". In: *AIAA Guidance, Navigation, and Control Conference*, Number 0378, San Diego, California, USA, Jan. 2016. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-389-6. doi: 10.2514/6.2016-0378. URL: https://arc.aiaa.org/doi/10.2514/6.2016-0378.

Szmuk, M., Eren, U., and Acikmese, B. "Successive Convexification for Mars 6-DoF Powered Descent Landing Guidance". In: *AIAA Guidance, Navigation, and Control Conference*, Number 1500, Grapevine, Texas, Jan. 2017. American Institute of Aeronautics and Astronautics. ISBN 978-1-62410-450-3. doi: 10.2514/6.2017-1500. URL: https://arc.aiaa.org/doi/10.2514/6.2017-1500.

Szmuk, M., Reynolds, T. P., and Açıkmeşe, B. "Successive Convexification for Real-Time Six-Degree-of-Freedom Powered Descent Guidance with State-Triggered Constraints". *Journal of Guidance, Control, and Dynamics*, Vol. 43, No. 8, pp. 1399–1413, Aug. 2020. doi: 10.2514/1.G004549. URL: https://arc.aiaa.org/doi/10.2514/1.G004549.

Sánchez-Sánchez, C. and Izzo, D. "Real-Time Optimal Control via Deep Neural Networks: Study on Landing Problems". *Journal of Guidance, Control, and Dynamics*, Vol. 41, No. 5, pp. 1122–1135, May 2018. doi: 10.2514/1.G002357. URL: https://arc.aiaa.org/doi/10.2514/1.G002357.

Tipaldi, M., Iervolino, R., and Massenio, P. R. "Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges". *Annual Reviews in Control*, Vol. 54 pp. 1–23, 2022. doi: 10.1016/j.arcontrol.2022.07.004. URL: https://linkinghub.elsevier.com/retrieve/pii/S136757882200089X.

Tony Landis, NASA. "Space shuttle atlantis touchdown", 2007. URL: https://www.dfrc.nasa.gov/Gallery/Photo/STS-117/HTML/ED07-0137-01.html. [Online; accessed May 27, 2024].

Topputo, F. and Zhang, C. "Survey of direct transcription for low-thrust space trajectory optimization with applications". *Abstract and Applied Analysis*, Vol. 2014 pp. 1–15, 06 2014. doi: 10.1155/2014/851720.

Tsiotras, P. and Mesbah, A. "Introducing Computational Guidance and Control". *Journal of Guidance, Control, and Dynamics*, Vol. 40, No. 2, p. 193, Feb. 2017. doi: 10.2514/1.G002745. URL: https://arc.aiaa.org/doi/10.2514/1.G002745.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. "Attention Is All You Need", Aug. 2017. URL: http://arxiv.org/abs/1706.03762. doi: arXiv:1706.03762.

Von Stryk, O. and Bulirsch, R. "Direct and indirect methods for trajectory optimization". *Annals of Operations Research*, Vol. 37, No. 1, pp. 357–373, Dec. 1992. doi: 10.1007/BF02071065. URL: http://link.springer.com/10.1007/BF02071065.

Wakker, K. F. *Fundamentals of astrodynamics*. Aerospace Engineering (Aerospace Engineering) (TU Delft), Jan. 2015. ISBN 978-94-6186-419-2. URL: https://repository.tudelft.nl/record/uuid:3fc91471-8e47-4215-af43-718740e6694e.

Wang, J. X., Kurth-Nelson, Z., Tirumala, D., Soyer, H., Leibo, J. Z., Munos, R., Blundell, C., Kumaran, D., and Botvinick, M. "Learning to reinforcement learn", Jan. 2017. URL: http://arxiv.org/abs/1611.05763. doi: arXiv:1611.05763.

Wang, Z. "A survey on convex optimization for guidance and control of vehicular systems". *Annual Reviews in Control*, Vol. 57 p. 100957, 2024. doi: 10.1016/j.arcontrol.2024.100957. URL: http://dx.doi.org/10.1016/j.arcontrol.2024.100957.

White, P. W. and Hoffman, J. "Earth Global Reference Atmospheric Model (Earth-GRAM): User Guide". Technical report, NASA, Sept. 2021.

Wilson, C. and Riccardi, A. "Improving the efficiency of reinforcement learning for a spacecraft powered descent with Q-learning". *Optimization and Engineering*, Vol. 24 pp. 223–255, Oct. 2021. doi: 10.1007/s11081-021-09687-z. URL: https://link.springer.com/10.1007/s11081-021-09687-z.

Wächter, A. and Biegler, L. T. "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming". *Mathematical Programming*, Vol. 106, No. 1, pp. 25–57, Mar. 2006. doi: 10.1007/s10107-004-0559-y. URL: http://link.springer.com/10.1007/s10107-004-0559-y.

Xue, S., Bai, H., Zhao, D., and Zhou, J. "Research on Intelligent Control Method of Launch Vehicle Landing Based on Deep Reinforcement Learning". *Mathematics*, Vol. 11, No. 20, p. 4276, Oct. 2023. doi: 10.3390/math11204276. URL: https://www.mdpi.com/2227-7390/11/20/4276.

Zavoli, A. and Federici, L. "Reinforcement Learning for Robust Trajectory Design of Interplanetary Missions". *Journal of Guidance, Control, and Dynamics*, Vol. 44, No. 8, pp. 1440–1453, Aug. 2021. doi: 10.2514/1.G005794. URL: https://arc.aiaa.org/doi/10.2514/1.G005794.

A

Research plan

A.1. Work packages division and Gantt Chart

In the context of evolving space exploration demands and requests for cost-effective satellite launches, the focus has shifted towards achieving reusability in space vehicles, notably through precise powered vertical landings. SpaceX's success with Falcon 9 first-stage landings has been groundbreaking, introducing a significant improvement in the space launch industry. Similarly, NASA's Moon to Mars Architecture highlights the importance of pinpoint landing accuracy for establishing a future sustained human presence on these two bodies. However, achieving such precision poses challenges, as existing landing ellipses are typically kilometers wide due to many uncertainties, such as those on vehicle performances and environmental conditions. The powered descent and landing phase, which is the terminal stage of a mission, heavily relies on Guidance, Navigation, and Control (GNC) systems to enhance accuracy. While traditional analytical guidance algorithms have been used, recent advancements in mathematics computational power have enabled the exploration of Computational Guidance methods, such as convex optimization techniques. These methods offer real-time capabilities and fast convergence, even if they introduce simplifications and assumptions in the problem formulation. After the recent technological surge of Machine Learning (ML) applications, techniques such as Supervised Learning and Deep Reinforcement Learning (DRL) have also been applied to aerospace research and studies, showing promising results in spacecraft landing guidance. DRL, in particular, offers real-time capability and robustness to uncertainties, presenting a novel approach for powered landing guidance. Moreover, Transformer-based and recurrent neural networks are the most suitable architecture to provide the capabilities of capturing long-term dependencies. This ability is at the core of meta-RL and it is extremely useful when sequential optimal actions have to be taken in uncertain environments.

The objective of this research is to develop an ML-based guidance algorithm to enhance adaptability and robustness in powered descent and landing scenarios, particularly in Earth-like environments. The problem formulation includes 6-DOF dynamics and accurate vehicle and environment models to enhance the application of the solution in a real scenario. The reference scenario is a rocket, equipped with four fins and one gimballed engine, during its powered descent landing phase. The proposed algorithm will be trained to optimize spacecraft trajectories while satisfying path and control constraints, with performance comparable (or superior) to traditional optimal control solutions. Therefore, the research question is:

How can Machine Learning improve the robustness of a guidance system with real-time capability, for the powered landing phase in an uncertain (atmospheric) environment?

The main research question is then broken down into multiple subquestions that will be addressed during the thesis, in a, more or less, chronological order.

- **[RQ 1] What is the best Machine Learning method to be used in a guidance system for an atmospheric powered landing mission scenario?**
- **[RQ 2] How can the training process of a Machine Learning algorithm be improved for a rocket landing guidance system?**
- **[RQ 3] How accurate is the performance of the developed guidance system in terms of mass consumption and terminal pose?**
- **[RQ 4] How does the Machine Learning-based guidance system perform compared to the state-of-the-art solutions in a strongly perturbed environment?**

The first one is already partially answered in the Literature Study, where Supervised and Unsupervised Learning are ruled out with respect to RL, which is the chosen method. This choice stems from the fact that the first two methods are inherently limited by their training dataset. Learning from a potentially unlimited experience with RL, rather than on a limited dataset with Supervised Learning, gives the advantage of generalization and the ability to the network to cope with situations unseen in the training.

In particular, meta-RL has shown better performance in similar space guidance problems, where the optimal action does not depend only on the current observation, but also on the history of previous states. This method is used to train specific neural networks that are able to leverage previous information to learn an internal policy so that they can cope with a broad set of uncertainties. PPO is the state-of-the-art RL algorithm, which is used in the large majority of continuous control problems, and it is used also in this thesis. The neural networks chosen to be used in this thesis are LSTM and GTrXL. The first is a type of Recurrent Neural Network, largely used in meta-RL and in similar space guidance applications, while the second is an attention-based Transformer neural network, broadly used in Large Language Models, such as ChatGPT. It is not widely employed in RL problems, but it is chosen because it can potentially improve the state-of-the-art solutions due to its ability to capture dependencies in long sequences. The two neural network architectures are then compared to understand the differences in performance.

The work of this thesis is divided into work packages:

A **Benchmark simulator** (week 7-10)
Objective(s): The first objective is to develop the 6-DOF flight simulator with the models used by the benchmark guidance and to test each module alone, before a system verification. The second objective is to generate the benchmark using direct optimization software (ICLOCS). After that, the control output will be used to perform the system verification and to compare the results with the Benchmark paper.
Required inputs to start: Mathematical formulation of the simulator models.
Tasks:

1. Define input-output interfaces between simulator blocks (1 day)
2. Set up simulator high-level architecture (1 day)
3. Implement problem equations of motion (0.5 days)
4. Implement transformations between reference frames (0.5 days)
5. Implement environment models (atmosphere, gravity, aerodynamics) (1 day)
6. Test each simulator module with unit testing against the expected outcome (1 day)
7. Report on models implementation and verification (2 days)
8. Set up mathematical model with direct optimization software (ICLOCS) (2 days)
9. Implement mathematical formulation of the problem with direct optimization (3 days)
10. Generate optimal solution (2 days)

11. Report results of benchmark solution (3 days)
12. System verification of flight simulator using control outputs from benchmark (1 day)

Expected outcome: The first expected outcome of this work package is a verified 6-DOF flight simulator. The simulator is developed in Python. Concurrently, the same simulator is developed also in MATLAB to be used by the direct optimization software. Another outcome is the exact matching of results between the two simulators. Finally, another expected outcome is the generation of the benchmark solution, which has to be coherent with that from the publication of Lee and Lee Lee and Lee (2022). The last expected outcomes are the writing of the sections on simulator development, verification, and benchmark generation.

B **Advanced simulator** (week 11-12)
Objective(s): Improve the complexity of the simulator to include higher-order effects to reduce the simulation-reality gap.
Required inputs to start: First verified version of the simulator.
Tasks:

1. Implement more complex models (e.g.: include wind, include earth rotation term, include variable gravity, higher initial altitude) (2 days)
2. Unit testing of new models (1 day)
3. Generate optimal solution with more advanced models using ICLOCS (2 days)
4. System verification of simulator using optimal solution with more advanced models (1 day)
5. Generate optimal trajectory plots (1 day)
6. Report on results (2 days)

Expected outcome: The first expected outcome of this work package is a more accurate, verified, simulator. Moreover, it is expected that a new optimal trajectory from extended initial conditions and with more complex models is generated. Finally, it is expected to complete the chapter on simulator development and verification.

C **Guidance** (week 13-16)
Objective(s): In this work package, the objective is to preliminary develop the guidance system based on meta-RL. First, the meta-RL architecture needs to be set up, and an interface with the simulator needs to be implemented, for both LSTM and Transformers architecture. Then, a reward function is developed. Initial training trials of the network should be performed.
Required inputs to start: Fully developed and tested simulator.
Tasks:

1. Set up meta-RL framework (2 days)
2. Implement Transformers-based neural network architecture (2 days)
3. Implement LSTM neural network architecture (1 day)
4. Develop interface RL guidance with simulator (3 days)
5. Develop reward function for RL algorithm (3 days)
6. Implement reward functions (1 day)
7. Train network with different reward functions (4 days)
8. Report implementation meta-RL guidance strategy (2 days)

Expected outcome: The first expected outcome is the creation of the RL framework for the guidance system. Moreover, the chosen neural network architectures are implemented. Furthermore, it is expected to develop different variants of reward function for

RL training and choose the best one. After that, a preliminary version of the RL policy for the guidance system is expected to be generated. Finally, the last outcome is the writing of the RL based guidance system development in the relative chapter.

D **Analysis** (week 17-22)
Objective(s): The objective is to perform a refinement of the guidance system, analyzing different hyperparameters that could improve the performances, such as weights in reward function, and neural network-specific internal parameters. Initially, the reward function is tuned and tested, to quickly iterate it, to find a decent formulation. Then, the hyperparameters are tuned and the performances are compared and evaluated. Then, complete training is performed on the final setup of the reward function and hyperparameters for both LSTM and Transformer architectures. Finally, additional perturbances are added, such as control and navigation errors, to test the robustness of the policy.
Required inputs to start: RL framework and preliminary guidance algorithm.
Tasks:

1. Tune reward function weights (3 days)
2. Report about findings on different versions of the reward function (3 days)
3. Sensitivity neural networks hyperparameters (4 days)
4. Report on tuning results (3 days)
5. Train agent using the final setup in an uncertain environment (4 days)
6. Train agent including navigation and control errors (4 days)
7. Develop Monte Carlo setup (2 days)
8. Evaluate performance in nominal scenario (1 day)
9. Report on performances (2 days)
10. Buffer for delays due to long training (5 days)

Expected outcome: The expected outcome of this work package is the optimal tuned parameters of the RL algorithms. Therefore, it is expected to train the neural networks to find the final policy to be used as guidance system. Finally, it is expected to complete the writing of the guidance system chapter, adding the results and conclusions obtained in this work package.

E **Comparison** (week 23-26)
Objective(s): Monte Carlo campaigns are run for the final policies for both Transformers and LSTM. Their results are compared with Monte Carlo closed-loop simulations where an optimal trajectory is tracked by an LQR controller generated with direct optimization methods.
Required inputs to start: Final guidance policy and benchmark framework.
Tasks:

1. Run Monte Carlo campaign for Transformer-based architecture (3 days)
2. Postprocess and generate plots (2 days)
3. Report results (2 days)
4. Run Monte Carlo campaign for LSTM-based architecture (3 days)
5. Postprocess and generate plots (2 days)
6. Report results (2 days)
7. Design LQR controller (2 days)
8. Tune LQR controller (2 days)
9. Run Monte Carlo of conventional closed-loop Guidance and Control trajectories (2 days)
10. Generate plots (1 day)
11. Report comparison and conclusions (2 days)

Expected outcome: The expected outcome of this work package is to compare the quality of the guidance systems in an uncertain environment, using the optimal policies obtained with the two different RL methods. Moreover, it is expected to reflect on the results between different architectures to understand the strengths, weaknesses, and possible improvements of each method. Finally, it is expected to compare the RL results with the Monte Campaign using conventional Guidance and Control, where a nominal trajectory is tracked by an LQR controller, to understand if a RL-based guidance system can reach the same performances.

F **Finalization** (week 27-32)
Objective(s): The main objective is the finalization of the report takes place, with the generation of the final plots and the writing of the last pieces of the work. A complete review of the document must take place, before the Green Light meeting. Afterward, the thesis will be submitted and the final 4 weeks of downtime between hand-in and defense will be used to iterate on the report, prepare the final presentation, and be ready for the discussion of the project.
Required inputs to start: Completed code and advanced thesis draft.
Tasks:

1. Generate additional plots (2 days)
2. Finalize writing report (3 days)
3. Complete review of the report (3 days)
4. Proofread (1 day)
5. Prepare for Green Light meeting (2 days)
6. Prepare final presentation
7. Prepare defense

Expected outcome: The main expected outcome is the finalization of the thesis report, with a complete review and proofreading. Furthermore, the deliverables for the Green Light meeting are prepared. The last expected outcome of this work package is the preparation of the final presentation and defense.

During the thesis project, programming will be a core part of it, and different programming languages will be used for different purposes. For example, the direct optimization tool (ICLOCS) has a MATLAB implementation. Hence, in the benchmark generation, MATLAB will be used. On the other hand, a lot of standardized RL libraries are present in Python. Therefore, for the RL-based guidance Python will be the designated programming language, where libraries such as RLLIb, from Ray's Project, or Stable Baseline 3, which is a DLR-developed library. Both include a set of reliable implementations of RL algorithms in PyTorch and TensorFlow. Postprocessing of the data and plotting will be mainly done in MATLAB.

A Gantt chart, outlining the tasks coming out of the Work Packages division, is attached to plan the 32 weeks of the thesis.

A.2. Final reflection

After the conclusion of the thesis, it is interesting to reflect on the initial proposal and research plan. As a general comment, the Gantt Chart provided a fairly accurate estimate for most work packages, particularly in the first half of the project. The planning was respected, especially for the first half up to the midterm review. The hand-in of the thesis draft happened about three weeks later than the expected date, mainly due to holidays, not accounted for in the initial planning.

The flight simulator development and verification, and optimal control setup were completed within the planned timeframe, but tuning the optimal solution took longer than expected, with several optimizations running for many hours. However, this delay was mitigated

by running it in parallel with the "Guidance" work package, ensuring no impact on overall planning. While setting up the meta-RL framework and the Transformer and LSTM-based neural network in Python, the nominal trajectory was optimized in MATLAB.

The setup, understanding, and choice of the meta-RL and NNs libraries took more than expected, while the coding part to interface them with the flight simulator was shorter than planned, balancing the timeline.

The reward function tuning took two weeks more than expected, involving many iterations. Many trial-and-error trainings were run, each for a couple of days to infer relevant trends and behaviors. However, the downtime during training was effectively utilized to create a script for running the RL framework on the DelftBlue supercomputer, allowing for parallel training sessions. Including a buffer in the planning to account for potentially long training sessions turned out to be very helpful in avoiding delays.

Running on the DelftBlue supercomputer significantly optimized the time for the "Analysis" and "Comparison" work packages, allowing parallel training sessions and more detailed problem analysis than initially planned. This enabled testing of seven different Transformer-based neural network hyperparameter combinations and multiple LSTM trainings for comparison. The parallelization given by the DelftBlue supercomputer also allowed more training sessions of new neural networks, exploring and assessing different features of the problem, such as the robustness to control and navigation errors, the sensitivity to the removal of network's inputs (mass, actions, and reward), and the introduction of the rate constraint on the thrust magnitude. During the "Comparison" package, while the training sessions were running, some time was allocated to the implementation of a terminal patch to improve the vertical velocity, and to the development of an efficient Monte Carlo setup. Therefore, the time gained thanks to these improvements was spent carefully designing and tuning the LQR controllers. Working in parallel to the NN optimizations, two controller architectures were created for the conventional guidance and control strategy used for comparison. Even if the tuning took longer than expected, the ability to work in parallel on RL and conventional strategies prevented any delay.

Therefore, the bulk of the work was completed with a delay of only one week. However, it prevented the draft completion before to summer holidays, delaying it to the end of August.

In conclusion, dividing the work into distinct work packages and tasks proved essential for continuously reassessing priorities and meeting the thesis objectives. The Gantt chart was invaluable for tracking progress and organizing weekly plans. Although new challenges emerged in the second half of the project, efficient solutions helped optimize the time available. The initial time estimates, while useful for setting expectations, were less accurate as the project progressed and tasks became more complex. Parallelizing tasks and regularly adjusting timelines were the key factors to maintain efficiency and mitigate delays.

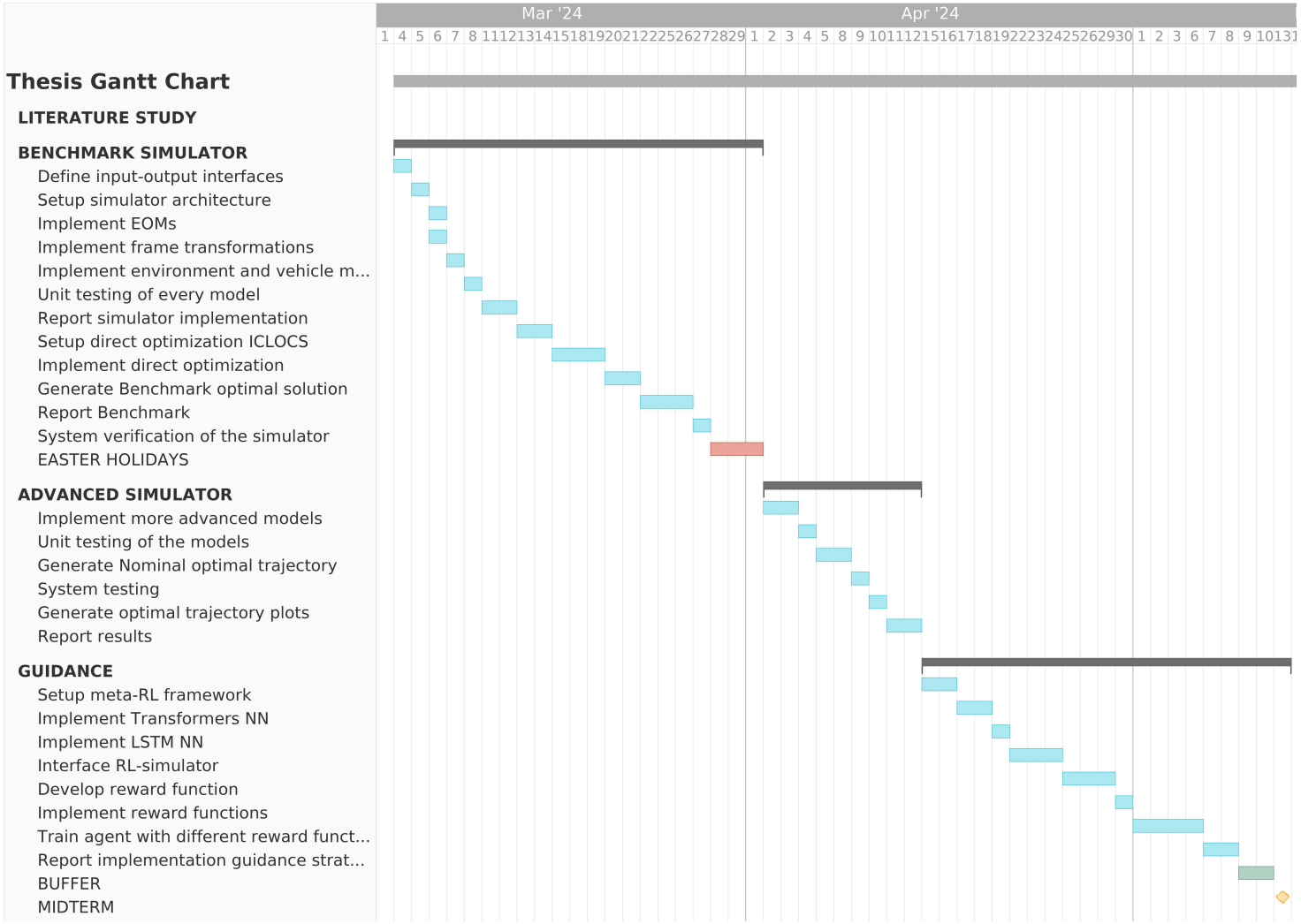


Figure A.1: First part of the Gantt Chart (before Midterm Review).

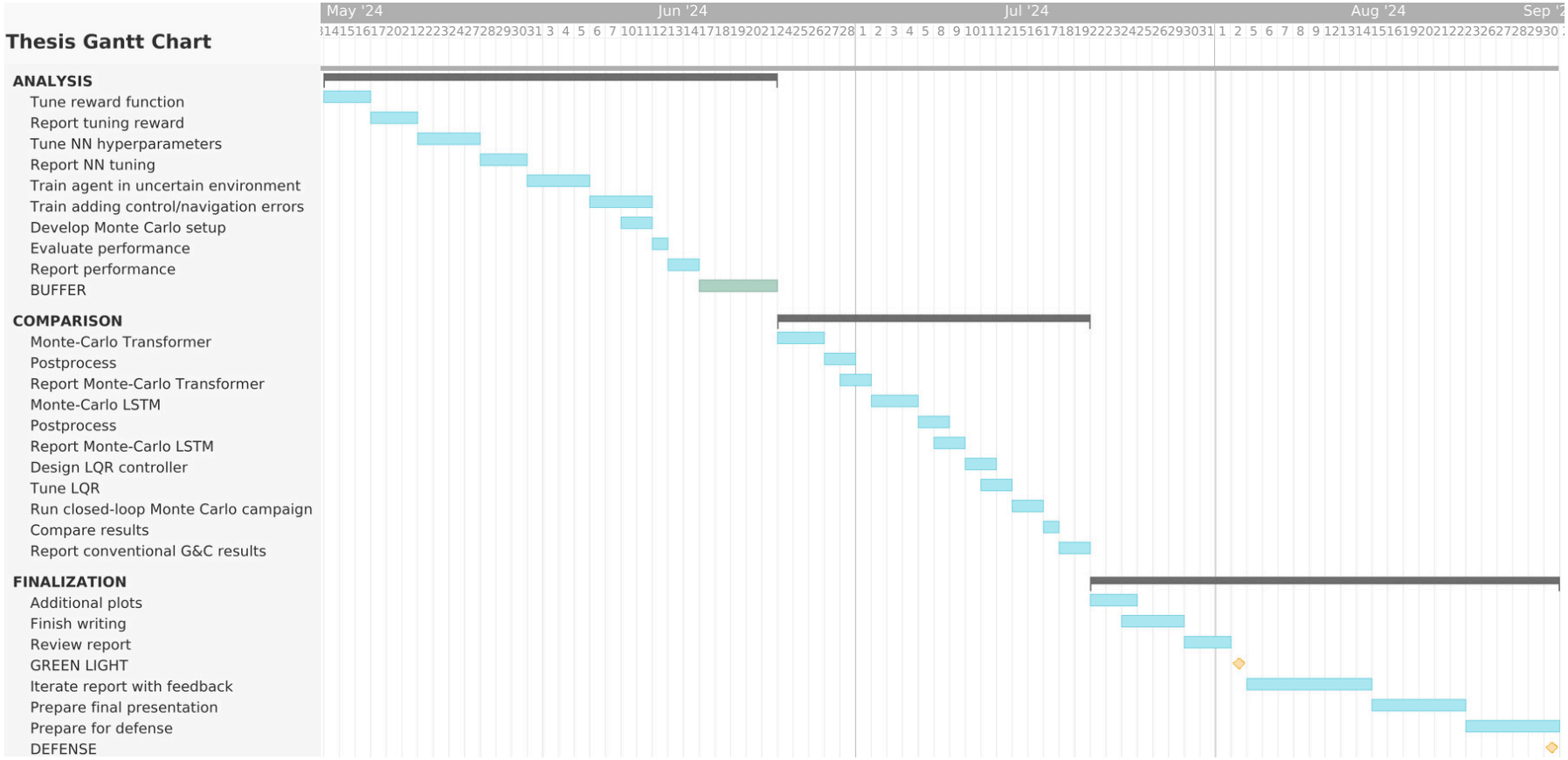


Figure A.2: Second part of the Gantt Chart (after Midterm Review).

B

Reference frames, frame transformations, and state conversions

B.1. Reference frames

Earth centered rotating reference frame $\mathcal{R}$

Another reference frame fixed to the central body, on Earth, is the Earth Centered Earth Fixed (ECEF) reference frame. It has its origin on the center of mass of the Earth, its $Z_{\mathcal{R}}$ axis points towards the north pole and $X_{\mathcal{R}} - Y_{\mathcal{R}}$ plane lies on the terrestrial equatorial plane. $X_{\mathcal{R}}$ points to the Greenwich Meridian, which is the prime meridian, and $X_{\mathcal{R}}$ completes the right-handed system. This reference frame rotates with Earth, around its rotation axis at rotation rate $\omega_E = 7.292115 \times 10^{-5}\text{rad s}^{-1}$ (Petit and Luzum, 2010). Formally, the transformation between ECI and ECEF is not composed only by the rotation about the Earth rotation axis, but it should also account for the relative motion of the $Z_{\mathcal{R}}$ axis with respect to the $Z_{\mathcal{I}}$ axis due to precession and nutation effects. However, given the short simulation time, these two effects are neglected and $Z_{\mathcal{R}}$ coincides with $Z_{\mathcal{I}}$.

Body reference frame $\mathcal{B}$

The body reference frame is fixed to the vehicle, more specifically it has its origin on the CoM of the vehicle, the $X_{\mathcal{B}}$ axis lies along the vehicle's longitudinal axis and it is positive in the forward direction and the $Z_{\mathcal{B}}$ axis is defined to be positive downwards (Mooij, 1994). Therefore, the $Y_{\mathcal{B}}$ axis completes the right-handed set. Following this definition, the roll, pitch, and yaw angles, are defined as the relative orientation of the body frame with respect to the landing site **UEN** frame.

Propulsion reference frame $\mathcal{P}$

Since the engine includes TVC actuators to modify the thrust direction to adjust the motion, an auxiliary reference frame is defined. It is centered on the CoM of the vehicle, its $X_{\mathcal{P}}$ axis is collinear with the resulting thrust force and positive in the direction of this force. The orientation of the propulsion frame with respect to the body frame is given by the deflection of the TVC actuators as ϵ_T and ψ_T, representing their deflection around the $Y_{\mathcal{P}}$ (elevation angle of the thrust force) and $Z_{\mathcal{P}}$ axes (azimuth angle of the thrust force), respectively, as mentioned in Simplício et al. (2020) and Mooij (2019).

Vertical reference frame $\mathcal{V}$

In the vertical reference frame, the origin is in the vehicle's CoM, and its $Z_{\mathcal{V}}$ axis points downwards to the CoM of the central body, along the radial component of the gravitational

acceleration. The $X_{\mathcal{V}}$ axis lies in a meridian plane, it's perpendicular to $Z_{\mathcal{V}}$ and points to the northern hemisphere, while $Y_{\mathcal{V}}$ completes the right-handed set. The $X_{\mathcal{V}} - Y_{\mathcal{V}}$ plane is exactly the local horizontal plane. The local horizontal plane of the landing site is approximately coincident with the instantaneous local horizontal plane of the vehicle, given the tiny effect of the Earth curvature in the very small distances considered in the problem.

Aerodynamic reference frame $\mathcal{A}$

An aerodynamic reference frame is convenient to describe the aerodynamic coefficients and forces. This reference frame is centered in the CoM of the vehicle and has the $X_{\mathcal{A}a}$ axis directed along the airspeed velocity vector $\mathbf{V}_A$ (with respect to the atmosphere considering wind velocity vector), the $Z_{\mathcal{A}a}$ axis directed along the lift force (but in opposite direction) and the $Y_{\mathcal{A}a}$ completes the right-handed set. The relative orientation of this frame with respect to the body frame can be represented through the two aerodynamic angles: the angle of attack (α) and the angle of sideslip (β). If the vehicle is not banking, this reference frame coincides with the trajectory reference frame, otherwise the bank angle is identified as the relative orientation angle between the two.

The same reference frame could be also defined based on the groundspeed, which is the velocity vector relative to the rotating planet reference frame. Since the equations of motion are integrated with respect to a planetocentric reference frame, the velocity given out of them is representative of the groundspeed, because it is indeed the velocity of the vehicle with respect to a fixed point on the surface of the Earth.

The airspeed velocity vector is composed by the groundspeed $\mathbf{v}_G$ and the wind velocity $\mathbf{v}_w$ vectors:

$$\mathbf{v}_A = \mathbf{v}_G - \mathbf{v}_w \tag{B.1}$$

The two aerodynamic angles (angle of attack and the sideslip angle) are computed using the airspeed velocity vector, expressed in body frame.

$$\alpha = \arctan_2\left(\frac{v^{\mathcal{B}}_{A,z}}{v^{\mathcal{B}}_{A,x}}\right) \qquad \beta = \arcsin\left(\frac{v^{\mathcal{B}}_{A,y}}{||\mathbf{v}^{\mathcal{B}}_A||}\right) \tag{B.2}$$

Using $\arctan_2$, α is bounded between $(+\pi, -\pi]$, while β between $(-\pi/2, +\pi/2)$ with 0 included. A positive angle of attack is defined for a "nose-up" attitude, positive sideslip for a "nose-left" attitude.

Pitch fins reference frame $\mathcal{F}_{pitch}$

One of the two sets of aerodynamic surfaces is dedicated to the control of the pitch rotation. In order to compute the fins' aerodynamic forces, a reference frame fixed on this set of fins is defined. The $X_{\mathcal{F},pitch}$ axis is aligned with the mean chord of the fin, with a positive direction toward the tip of the rocket. The $Z_{\mathcal{F},pitch}$ axis is aligned with the pitch axis of the vehicle, but in the opposite direction (i.e. $Y_{\mathcal{B}}$) and $Y_{\mathcal{F},pitch}$ completes the right-handed set. This reference frame, with respect to the body one, is rotated by 90 degrees positively around the vehicle longitudinal axis ($X_{\mathcal{B}}$) and then by the fins deflection angle $\beta_{fin,pitch}$ around $Z_{\mathcal{F},pitch}$. The fins only have a single degree of freedom: rotation around the $Z_{\mathcal{F},pitch}$ axis. Both fins from this pair share this body-fixed reference frame, as shown in Fig. B.1.

Yaw fins reference frame $\mathcal{F}_{yaw}$

The other set of fins is dedicated to the control of the yaw moments and they are aligned with the yaw axis. This reference frame has $X_{\mathcal{F},yaw}$ axis is aligned with the mean chord of the fin, with a positive direction toward the tip of the rocket and the $Z_{\mathcal{F},yaw}$ axis coincident with the $Z_{\mathcal{B}}$ axis. Therefore, this frame is rotated by the fins deflection angle $\beta_{fin,yaw}$ with respect to the $Z_{\mathcal{F},yaw}$ axis in the body frame. The fins only have a single degree of freedom: rotation

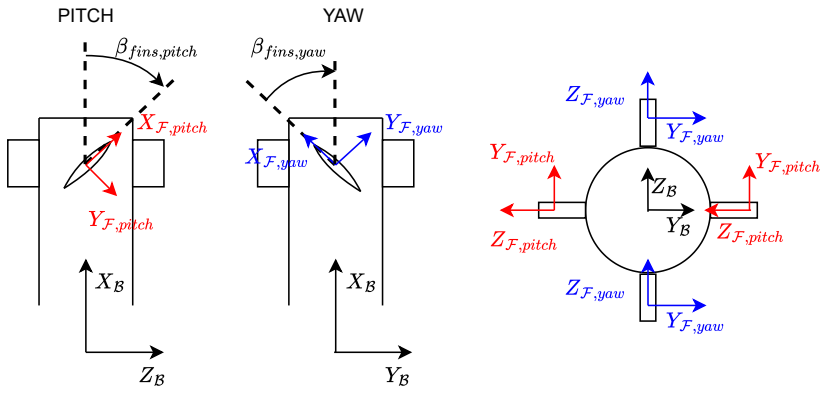


Figure B.1: Fins pitch and yaw reference frame.

around the $Y_{\mathcal{F},yaw}$ axis. Both fins from this pair share this body-fixed reference frame, as shown in Fig. B.1.

It is reminded that, in this document, the notation $\beta_{fin,pitch,i} \quad i = 1, 2$ is used to represent the deflection angles of the fins on the pitch plane and $\beta_{fin,yaw,i} \quad i = 3, 4$ those on the yaw plane.

B.2. Frame Transformations

This section gives an overview of the theoretical ground to operate frame transformations, with a focus on the most common transformation used in this work, concerning the powered descent problem. It is useful to express certain vectors in some specific reference frame. This leads to the use of many vectors in different reference frames, which eventually have to be referred to a single common one. Therefore, translations and rotations are used to transform a vector from a reference frame to another, as shown in Eq. (B.3). The vector is originally in a generic frame A ($\mathbf{v}_A$), and it is rotated and translated to express it in the generic frame B ($\mathbf{v}_B$). The translation is represented by the vector $\mathbf{T}$, which is the difference component by component between the origins of the two reference frames in the 3D space. The term $\mathbf{C}_{B,A}$ is a transformation matrix to rotate from frame A to frame B.

$$\mathbf{v}_B = \mathbf{T} + \mathbf{C}_{B,A}\mathbf{v}_A \tag{B.3}$$

Different methods can be used to transform between right-handed reference frames: Direction Cosine Matrix (DCM), unit-axis rotations, and quaternions.

In three-dimensional space, a DCM is a rotation matrix to transform from one reference frame to another. Indicated as $\mathbf{C}_{B,A}$ it is a 3x3 orthogonal matrix from frame A to B, where the entries are the angles between the unit vectors of the two frames considered:

$$\mathbf{C}_{B,A} = \begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \\ \mathbf{b}_3 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{a}_1 & \mathbf{a}_2 & \mathbf{a}_3 \end{bmatrix} \tag{B.4}$$

However, the parameters needed in the DCM are 9, which means that there are 6 parameters redundant with respect to the minimum parameters (3) needed to represent a three-dimensional attitude, posing a possible computational burden. Therefore, the calculation of the DCMs, using the basis vector of frames A and B is discarded in this thesis.

What is employed in this work are Euler angles, because they have the minimum parameters needed to represent a 3D rotation, they are intuitive to understand and they are usually

available in the dynamics. Euler angles are used to form rotation matrices, employing sequential rotations of unit axis rotation, as explained in Appendix B.2. This approach is used for all frame transformations. The choice of using rotation matrices composed of Euler angles has the double advantage of using the minimum number of parameters and employing the available data. Moreover, having physically concrete numbers gives an easy way to verify the correct implementation, by applying the transformations and checking that the result is the one expected according to physical reasoning.

Unit axis rotation

Unit axis rotations are positive rotations according to the right-hand rule, based on the axis of the reference frame. A rotation by an arbitrary angle α around the x-, y-, or z-axis is represented as:

$$\mathbf{C}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha \\ 0 & -\sin\alpha & \cos\alpha \end{bmatrix} \tag{B.5}$$

$$\mathbf{C}_y(\alpha) = \begin{bmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{bmatrix} \tag{B.6}$$

$$\mathbf{C}_z(\alpha) = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.7}$$

Any rotation between two frames can be expressed as a sequential combination of the three aforementioned rotations. Since they are all orthonormal matrices, also the product of them is an orthonormal matrix. Therefore, the inverse of this matrix is simply the transpose, avoiding any singularity.

$$\mathbf{C}\mathbf{C}^T = \mathbf{C}^T\mathbf{C} = \mathbf{I} \quad \text{or} \quad \mathbf{C}^{-1} = \mathbf{C}^T \tag{B.8}$$

An example is given by taking three successive arbitrary rotations (a positive α_1 around the x-axis, a negative α_2 around the y-axis, and a positive α_3 around the z-axis), using a 3-2-1 formulation, to transform from frame A to frame B:

$$\mathbf{C}_{B,A} = \mathbf{C}_{B,A''}\mathbf{C}_{A'',A'}\mathbf{C}_{A',A} = \mathbf{C}_x(\alpha_1)\mathbf{C}_y(-\alpha_2)\mathbf{C}_z(\alpha_3) \tag{B.9}$$

The first rotation (α_3) is applied to the z-axis of frame B, while the second and third rotations (-α_2 and α_1) are applied to y- and x-axis of the intermediate frames A' and A", respectively. The inverse of such a transformation is obtained by inverting the order of multiplications and the sign of the rotations:

$$\mathbf{C}_{A,B} = \mathbf{C}_z(-\alpha_3)\mathbf{C}_y(\alpha_2)\mathbf{C}_x(-\alpha_1) \tag{B.10}$$

The rotation matrices can be multiplied, to compose multiple rotations between reference frames, for example going from frame A to B, can be decomposed using an intermediary frame I:

$$\mathbf{C}_{B,A} = \mathbf{C}_{B,I}\mathbf{C}_{I,A} \tag{B.11}$$

Euler angles roll (ϕ), pitch (θ), and yaw (ψ) are also defined as a set of 3 sequential rotations about the 3 axis. In addition to that, it has to be remarked that it is important the order of the product between the two matrices because different orders yield different results. A formulation 3-2-1 is used in this work to perform a rotation from reference frame A to B.

To conclude, there is a relationship between the DCM and the Euler angles, so that if one is known, the other could be derived. For example, using the 3-2-1 rotation sequence as in Equation (B.9), the DCM results in:

$$\mathbf{C}_{B,A} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & \cos\theta\sin\phi \\ \sin\phi\sin\psi + \cos\phi\cos\psi\sin\theta & \cos\phi\sin\theta\sin\psi - \cos\psi\sin\phi & \cos\phi\cos\theta \end{bmatrix} \tag{B.12}$$

Conversely, if the DCM is known, the Euler angles can be calculated as:

$$\begin{aligned} \phi &= \arctan 2(C_{23}, C_{33}) \\ \theta &= -\arcsin(C_{13}) \\ \psi &= \arctan 2(C_{12}, C_{11}) \end{aligned} \tag{B.13}$$

Frame transformation for landing applications

In Tab. B.1, transformations between reference frames relevant to the landing problem treated in this work are shown as the product of unit-axis rotations, with an explanation of the corresponding angles involved.

It should be remarked that the inverse transformation (arrow from right frame to left one) is just the transpose of the corresponding matrix, as shown in Eq. (B.10).

Any other transformation between two of the defined reference frames can be done by combining rotations, using intermediate reference frames, as shown in Eq. (B.11).

Table B.1: Reference frame transformations. All adopted from Mooij (1994), except fins transformations.

Frames	**Transformation**	**Variables**		
		Symbol	Unit	Description
$\mathcal{R} \rightarrow \mathcal{I}$	$\mathbf{C}_{\mathcal{I},\mathcal{R}} = \mathbf{C}_3(-\omega_E t)$	ω_E	$\frac{\text{rad}}{\text{s}}$	Earth rotation rate
		t	s	Time from epoch 0 (where $\mathcal{R} = \mathcal{I}$)
$\mathcal{V} \rightarrow \mathcal{R}$	$\mathbf{C}_{\mathcal{R},\mathcal{V}} = \mathbf{C}_3(-\tau)\mathbf{C}_2(\frac{\pi}{2} + \delta)$	τ	rad	Geocentric longitude
		δ	rad	Geocentric latitude
$\mathcal{B} \rightarrow \mathcal{A}$	$\mathbf{C}_{\mathcal{A},\mathcal{B}} = \mathbf{C}_3(\beta)\mathbf{C}_2(-\alpha)$	β_G (or β_A)	rad	Angle of sideslip based on groundspeed (or airspeed)
		α_G (or α_A)	rad	Angle of attack based on groundspeed (or airspeed)
$\mathcal{P} \rightarrow \mathcal{B}$	$\mathbf{C}_{\mathcal{B},\mathcal{P}} = \mathbf{C}_3(-\psi_T)\mathbf{C}_2(-\epsilon_T)$	ϵ_T	rad	Elevation angle of thrust force
		ψ_T	rad	Azimuth angle of thrust force
$UEN \rightarrow \mathcal{R}$	$\mathbf{C}_{\mathcal{R},UEN} = \mathbf{C}_3(-\tau)\mathbf{C}_2(\delta)$	τ	rad	Geocentric longitude
		δ	rad	Geocentric latitude
$\mathcal{B} \rightarrow UEN$	$\mathbf{C}_{UEN,B} = \mathbf{C}_3(-\psi)\mathbf{C}_2(-\theta)\mathbf{C}_1(-\phi)$	ψ	rad	Yaw angle
		θ	rad	Pitch angle
		ϕ	rad	Roll angle
$\mathcal{F}_{pitch} \rightarrow \mathcal{B}$	$\mathbf{C}_{\mathcal{B},\mathcal{F}_{pitch}} = \mathbf{C}_1(-\frac{\pi}{2})\mathbf{C}_3(-\beta_{fins,pitch})$	$\beta_{fins,pitch}$	rad	Fin deflection for pitch control
$\mathcal{F}_{yaw} \rightarrow \mathcal{B}$	$\mathbf{C}_{\mathcal{B},\mathcal{F}_{yaw}} = \mathbf{C}_3(-\beta_{fins,yaw})$	$\beta_{fins,pitch}$	rad	Fin deflection for yaw control

B.3. State conversion

The representations used in EOMs are the Cartesian for translational motion and quaternions for attitude. However, some data may not be already in this preferred form or may be transformed into another representation to enable a better visualization (such as quaternions to Euler angles). Therefore, a brief explanation how to convert from one state representation to the other is shown in the following paragraphs. Only the rotational state conversion is shown, since for the translational motion only a single representation is used. For a complete derivation, the reader may refer to Mooij (1994).

Quaternions-Euler angles conversion

Using the quaternion representation with the 4th element as scalar, and a 3-2-1 sequence rotation, Euler angles are computed from quaternions as:

$$\begin{pmatrix}\phi\\ \theta\\ \psi\end{pmatrix} = \begin{pmatrix}\arctan_2\left[2\left(q_4q_1+q_2q_3\right), 1-2\left(q_1^2+q_2^2\right)\right]\\ -\frac{\pi}{2}+2\arctan_2\left[\sqrt{1+2\left(q_4q_2-q_1q_3\right)}, \sqrt{1-2\left(q_4q_2-q_1q_3\right)}\right]\\ \arctan_2\left[2\left(q_4q_3-q_1q_2\right), 1-2\left(q_2^2+q_3^2\right)\right]\end{pmatrix} \tag{B.14}$$

Viceversa, if the 3 Euler angles are known, the quaternions are computed as:

$$\begin{aligned}\mathbf{q} &= \begin{pmatrix}0\\0\\\sin(\frac{\psi}{2})\\\cos(\frac{\psi}{2})\end{pmatrix}\begin{pmatrix}0\\\sin(\frac{\theta}{2})\\0\\\cos(\frac{\theta}{2})\end{pmatrix}\begin{pmatrix}\sin(\frac{\phi}{2})\\0\\0\\\cos(\frac{\phi}{2})\end{pmatrix} = \\ &= \begin{pmatrix}\sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2})\\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2})\\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2})\\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2})\end{pmatrix}\end{aligned} \tag{B.15}$$

C

Verification tests

C.1. Aerodynamics Model test cases

C.1.1. Body Vehicle

The aerodynamic coefficients are stored in lookup tables, generated with empirical formulations, for the wind axes (C_D, C_L), as well as already for the body axes (C_A, C_N), parameterized only with respect to the angle of attack and the Mach number. Given the axisymmetry of the vehicle, the bi-dimensional dataset can transformed into a three-dimensional representation, because the normal coefficient profile is the same for both the $Y_{\mathcal{B}}$-axis as well as for the $Z_{\mathcal{B}}$-axis, even if with respect to two different angles (β and α respectively).

Similar to what is done by Sagliano et al. (2022), the axial coefficient is evaluated using the total angle of attack (α_T) and the Mach number (M).

$$C_{A,body} = C_{A,body}(\alpha_T, M) \tag{C.1}$$

The total angle of attack is calculated as:

$$\alpha_T = \arccos(\cos(\alpha)\cos(\beta)) \tag{C.2}$$

Moreover, since the aerodynamic coefficients C_A and C_N are provided in a body reference frame with axes with opposite directions (i.e. X-axis pointing toward the engine of the rocket and Z-axis pointing up) to the one defined in this thesis, a minus sign has to be used to compute C_X (and similarly later for C_Y and C_Z).

$$C_{X,body} = -C_{A,body} \tag{C.3}$$

Similarly, coefficients along the body $Y_{\mathcal{B}}$- and $Z_{\mathcal{B}}$-axes are computed, calling twice the same C_N lookup table, once using angle of attack α and once auxiliary sideslip angle β^*, exploiting the rocket's axisymmetry.

$$C_{Y,body} = -C_{N,body}(\beta^*, M) \qquad \text{and} \qquad C_{Z,body} = -C_{N,body}(\alpha, M) \tag{C.4}$$

Consequently, using $C_{X,body}$, $C_{Y,body}$ and $C_{Z,body}$, the aerodynamic forces can be computed directly in body frame, as shown in Equation (3.9).

Similarly to the coefficients on $Y_{\mathcal{B}}$- and $Z_{\mathcal{B}}$-axes, the center of pressure is computed, calling twice its lookup, once using the angle of attack α and once the auxiliary sideslip angle β^*.

$$\mathbf{x}_{CP,\alpha} = [X_{CP}(\alpha, M), 0, 0] \qquad \text{and} \qquad \mathbf{x}_{CP,\beta^*} = [X_{CP}(\beta^*, M), 0, 0] \tag{C.5}$$

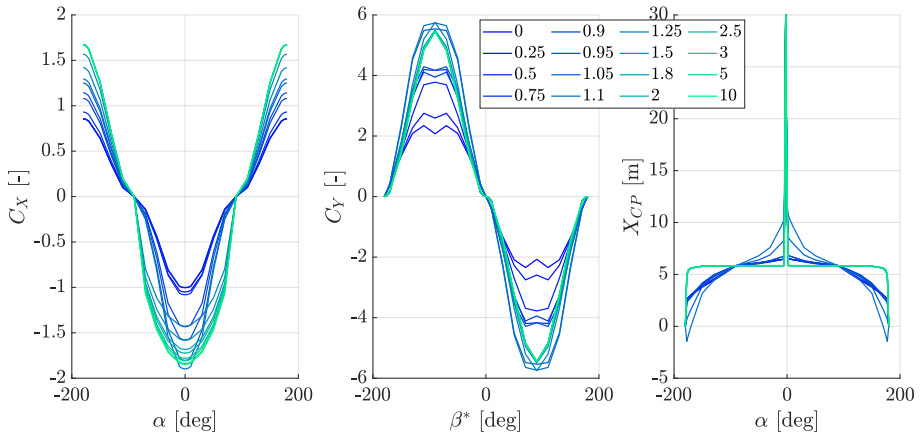


Figure C.1: C_X, C_Y and x_{CP} with respect to the base of the rocket.

In Fig. C.1, the profiles of C_X (with respect to α), C_Y (with respect to β^*), and X_{CP} (with respect to α) are shown, as provided in the lookup tables. C_Z (with respect to α) and X_{CP} (with respect to β^*) are not reported here for the sake of conciseness, since their profile would be exactly identical to those of C_Y and X_{CP}.

To verify the correctness of the implementation of the aerodynamic forces, the outcomes of some test cases are compared to what would be expected with critical reasoning. For example, using α=175 deg, β=10 deg, and a Mach number $M = 0.5$, it would be expected to have a resultant aerodynamic force with a positive component on the body $X_{\mathcal{B}}$-axis (rocket going down bottom first), and negative components on $Y_{\mathcal{B}}$-axis and $Z_{\mathcal{B}}$-axis (due to β and α), with the Y-component larger in magnitude with respect to the Z- one, due to the larger aerodynamic angle. Interpolating the coefficients (with α=175 deg and β^*=170 deg), using a dynamic pressure of $\bar{q}$=5.66 kPa, and a reference area S_{ref}=1.767 m^2, the forces calculated by the simulator, using Eq. (3.9), are equal to:

$$\mathbf{F}^{\mathcal{B}}_{aero,body} = \bar{q} S_{ref} \begin{pmatrix} C_X \\ C_Y \\ C_Z \end{pmatrix} = \begin{pmatrix} 8200.4 \\ -3999.6 \\ -1859.9 \end{pmatrix} \text{N} \tag{C.6}$$

The results are coherent with the already mentioned expected signs and relative magnitudes. Even if only one example is provided here, this critical reasoning process has been applied to many different test cases, verifying the correct implementation of the aerodynamics of the body vehicle.

Similarly, the correct position of the center of pressure is verified. If the β=10 deg was used, and the tables interpolated, the center of pressure would be more than 5 meters from the bottom of the rocket (Fig C.1), being behind the center of mass, with respect to the airflow. Hence, the vehicle would seem stable with respect to β. This is clearly wrong since it is known that the rocket body alone is unstable during the descent phase. Moreover, the rocket would be at the same time unstable with respect to α because close to 180 deg the center of pressure is close to the base of the vehicle, as shown in Fig. C.1. This conflict is also clearly wrong. Conversely, using the auxiliary sideslip angle β^*=170 deg, the vehicle is unstable with respect to both aerodynamic angles, with similar values for the center of pressure, as it should be. Therefore, it is confirmed that the use of the auxiliary angle is needed to correctly interpolate the lookup tables for the rocket descent.

C.1.2. Fins

The fins coefficients follow an analytical formulation, therefore the verification is straightforward. What has to be verified is that the transformation from the fin frame to the vehicle body frame is correct and it is done by testing some cases for which the result can be analyzed also visually.

For example, using a positive deflection of $\beta_{fin,i}$= 5 deg for each of the 4 fins, and aerodynamic angles of α=175 deg and β=10 deg, all four local fins' angles of attack are negative. Therefore, all 4 fins are expected to generate a positive force on the body $X_{\mathcal{B}}$-axis. Moreover, looking at Fig. B.1 and Fig. C.2, the set of fins for yaw control ($i = 3, 4$) is expected to generate a negative force on the body $Y_{\mathcal{B}}$-axis, while the pitch pair ($i = 1, 2$) a negative force on the body $Z_{\mathcal{B}}$-axis. It is also expected that the forces generated by the fin on the yaw axis are larger due to the larger local angle of attack.

The local angles of attack for the four fins respectively:

$$\alpha_i = -170 \text{ deg}, \quad (i = 1, 2) \qquad \text{and} \qquad \alpha_i = -165 \text{ deg}, \quad (i = 3, 4) \tag{C.7}$$

Therefore, using a maximum fins coefficient $\bar{C}_{fin} = 2$, the normal coefficients are:

$$C_{fin,i} = -0.347, \quad (i = 1, 2) \qquad \text{and} \qquad C_{fin,i} = -0.517, \quad (i = 3, 4) \tag{C.8}$$

Using a dynamic pressure $\bar{q}$=5.66 kPa and a fin reference surface $S_{ref,fin}$=0.54 m^2, the fins produce, for each fin reference frame, the normal forces shown in Eq. (C.9):

$$F_{N_{fin,i}} = -1062.1 \text{ N}, \quad (i = 1, 2) \qquad \text{and} \qquad F_{N_{fin,i}} = -1583.0 \text{ N}, \quad (i = 3, 4) \tag{C.9}$$

Therefore, by rotating these forces from the corresponding fin frame to the body frame, it yields:

$$\mathbf{F}^{\mathcal{B}}_{fin,i} = \begin{pmatrix} 92.6 \\ 0 \\ -1058.0 \end{pmatrix} \text{N}, \quad (i = 1, 2) \qquad \text{and} \qquad \mathbf{F}^{\mathcal{B}}_{fin,i} = \begin{pmatrix} 138.0 \\ -1577.0 \\ 0 \end{pmatrix} \text{N}, \quad (i = 3, 4) \tag{C.10}$$

These results match the expectation of the direction and relative magnitude of the forces generated by the fins.

Further verification is done to check whether the moments are calculated correctly. The centers of pressure of the fins are assumed to be fixed and to lie at their attachment point with the body of the vehicle. As shown in Fig. 2.2 and Fig. B.1, the attachment points are symmetrically positioned on the outer radius of the rocket ($r = 0.75$ m), and their x-coordinate ($h = 11.1$ m) is located at the same position on the longitudinal axis of the rocket.

$$\begin{aligned} \mathbf{x}_{CP,fin1} &= (h, r, 0) \qquad \text{and} \qquad \mathbf{x}_{CP,fin2} = (h, -r, 0) \\ \mathbf{x}_{CP,fin3} &= (h, 0, r) \qquad \text{and} \qquad \mathbf{x}_{CP,fin4} = (h, 0, -r) \end{aligned} \tag{C.11}$$

Assuming that they all lie more toward the tip of the rocket with respect to the center of mass (for this example $h - X_{CoM}$=6.5 m) and using the forces previously calculated, the moments around each axis can be easily calculated and compared to those given as output from the simulator.

The output from the simulator yields the results presented in Eq. (C.12), which are identical to the analytical results.

$$\begin{aligned} \mathbf{M}_{aero,fin1} &= (-793.5, 6878.2.5, -69.4) \text{ Nm} \\ \mathbf{M}_{aero,fin2} &= (793.5, 6878.2.5, 69.4) \text{ Nm} \\ \mathbf{M}_{aero,fin3} &= (118.3, 10.3, -1025.9) \text{ Nm} \\ \mathbf{M}_{aero,fin4} &= (-118.3, -10.3, -1025.9) \text{ Nm} \end{aligned} \tag{C.12}$$

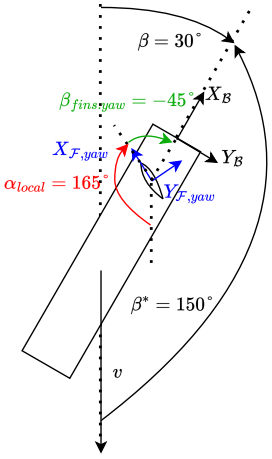


Figure C.2: Auxiliary β^* example.

Finally, summing these components yields:

$$\mathbf{M}_{aero,fins} = (0, 9958.62, -2257.41) \text{ Nm} \tag{C.13}$$

Therefore, compared with the analytical solution, it shows the same results, verifying its implementation. For example, it is correct that the roll moment is zero, since the deflections are the same for each pair, giving symmetrical moments on that axis. Moreover, each fin controlling the pitch axis generates a yaw moment, because the force along the $X_{\mathcal{B}}$-axis is perpendicular to the arm between the fin attachment point and the $Z_{\mathcal{B}}$-axis. However, given the same fin deflection, the moments have opposite signs, correctly resulting in a null moment. Similarly happens for the yaw fins.

It is here reported an additional example confirming the correct use of the auxiliary sideslip angle. It is assumed to be in a scenario where the vehicle is flying with an airspeed velocity in body frame of $\mathbf{v}_A^{\mathcal{B}} = (0, 10, -10\sqrt{3})$. It means that the angle of attack and sideslip angles are $\alpha = 180$ deg and $\beta = 30$ deg, respectively. Furthermore, it is assumed that each fin of the yaw set has a deflection of $\beta_{fin,yaw} = -45$ deg. A schematic of this situation is shown in Fig .C.2. Using the sideslip angle it would yield a local angle of attack $\alpha_{fin} = -75$ deg, which, using Eq. (3.12) and Eq. (3.14) would give a negative coefficient ($\sin -75 < 0$) and a negative force on the fin $Y_{\mathcal{F},yaw}$-axis. Using the auxiliary sideslip angle $\beta^* = 150$ deg, the local angle of attack becomes $\alpha_{fin} = -195$ deg $= 165$ deg, which yields a positive coefficient ($\sin 165 > 0$) and force on the fin $Y_{\mathcal{F},yaw}$-axis. Looking at the schematic in Fig. C.2 and understanding the physics at stake, it is clear that the second solution is the correct one. Moreover, this schematic shows that for positive velocity on $X_{\mathcal{B}}$-axis, during descent, the auxiliary sideslip angle is positive, yielding a negative force, as pointed out in Fig. C.1 and in the example shown in Section C.1.1.

C.2. Propulsion Model test case

The main concern with the propulsion force is assessing that its rotation from the propulsion frame to the body frame is done correctly, according to the TVC angles. Therefore, the verification of this rotation is done for cases where the visualization of the thrust direction is straightforward.

For a positive rotation of the engine around both $Y_{\mathcal{B}}$- and $Z_{\mathcal{B}}$-axes of the body frame, it is expected to generate a positive force on the $X_{\mathcal{B}}$- and $Y_{\mathcal{B}}$-axes and a negative one on the body $Z_{\mathcal{B}}$-axis. For example, by using a thrust magnitude of 50 kN and positive deflection of ϵ_T =5

deg and ψ_T =5 deg, the resulting calculated forces in body frame are:

$$\mathbf{F}_{\mathbf{T}}^{\mathcal{B}} = \begin{pmatrix} 49.62 \\ 4.34 \\ -4.36 \end{pmatrix} \text{ N} \tag{C.14}$$

It is assumed that the point of application of such forces is the gimbal point, which is the point of attachment between the stage and the engine, around which the engine rotates. Assuming that it is located on the longitudinal $X_{\mathcal{B}}$ axis, at a lower position with respect to the center of mass of the vehicle, the resulting moment can be calculated analytically.

$$\mathbf{M}_{\mathbf{T}}^{\mathcal{B}} = \begin{pmatrix} \Delta x_2 F_z - \Delta x_3 F_y \\ \Delta x_3 F_x - \Delta x_1 F_z \\ \Delta x_1 F_y - \Delta x_2 F_x \end{pmatrix} = \begin{pmatrix} 0 \\ -18.74 \\ -18.67 \end{pmatrix} \text{Nm} \tag{C.15}$$

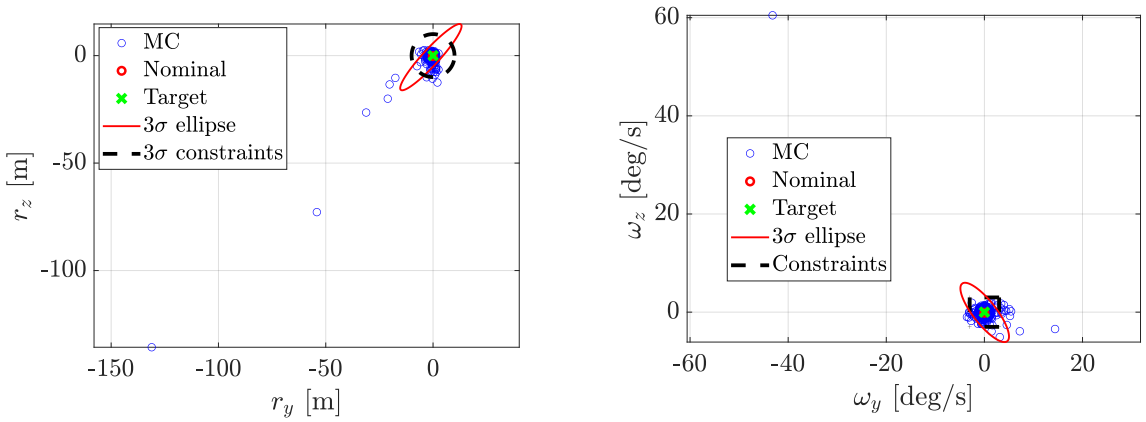
$\mathbf{\Delta x} = \mathbf{x}_{gimbal} - \mathbf{x}_{CG} = (\Delta x_1, \Delta x_2, \Delta x_3)$ is the distance between the gimbal point and the center of mass, expressed in body reference frame.

Using the aforementioned resulting forces and a difference between gimbal point and center of mass position of $\mathbf{\Delta x}$ =(-4.3,0,0) m, a null roll moment around the $X_{\mathcal{B}}$-axis, a negative yaw moment around the $Z_{\mathcal{B}}$-axis and a negative pitch moment around the $Y_{\mathcal{B}}$-axis shall be generated. The output of the simulator (Eq. (C.15)) shows the same results as using the analytical formulation, in accordance with the physical reasoning.

D

Extended initial conditions

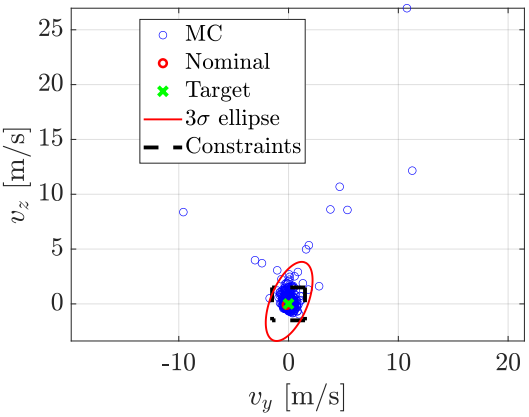
Table D.1 provides the extended initial conditions, where the policy including thrust rate constraint is tested on larger initial conditions compared to the ones where it is trained. On average, the distance from the nominal value is doubled with respect to the training set. Figures D.1 illustrate the terminal results, in terms of final position, velocity, vertical angle, and angular rates. Most of the trajectories (947/1000) meet the terminal constraints, except for the vertical velocity. However, there are some outliers, which are caused by a combination of initial conditions that are too different compared to the training data. In any case, it is remarkable that only 50 simulations completely fail. Since all the initial conditions are sampled from uniform distribution, and given the doubled range, on average 50% of the simulations have at least one initial variable outside of the training range. It means that the agent, even if it does not know how to behave in the few initial steps, taking random actions, effectively recovers once it gets back inside the known state space.



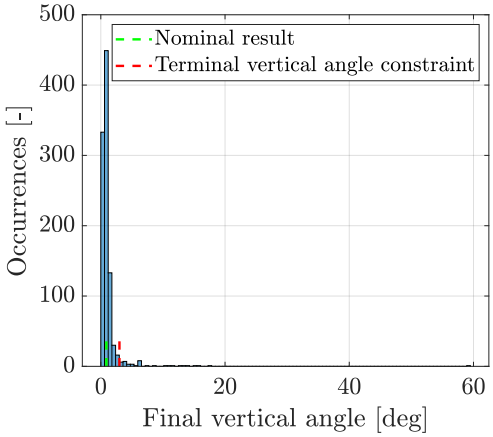
(a) Terminal horizontal position.

(b) Terminal angular rates.

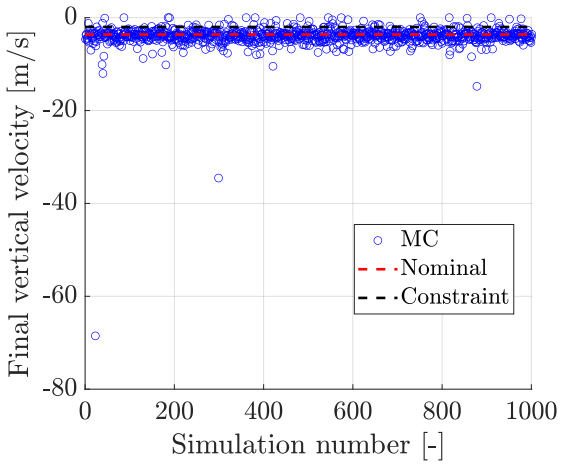
Figure D.1: Plots extended initial conditions using the policy that includes thrust rate.



(c) Terminal horizontal velocity.



(d) Terminal vertical angle.



(e) Terminal vertical velocity.

Figure D.1: Plots extended initial conditions using the policy that includes thrust rate. - Concluded.

Table D.1: Extended initial conditions in UEN landing site reference frame.

Description	Symbol	Unit	Nominal	Original Dispersion	Extended Dispersion
Elevation position	x_0	m	2000	[1900,2100]	[1800, 2200]
East position	y_0	m	50	[180,220]	[150,250]
North position	z_0	m	-50	[-220,-180]	[-250, -150]
Vertical velocity	v_{x_0}	$\frac{\text{m}}{\text{s}}$	-100	[-110,-90]	[-130, -90]
Horizontal velocity (East)	v_{y_0}	$\frac{\text{m}}{\text{s}}$	-25	[-30,-20]	[-40,-10]
Horizontal velocity (North)	v_{z_0}	$\frac{\text{m}}{\text{s}}$	25	[20,30]	[10, 40]
Roll	ϕ_0	deg	0	[-5,5]	[-10,10]
Pitch	θ_0	deg	20	[15,25]	[10,30]
Yaw	ψ_0	deg	20	[15,25]	[10,30]
Roll rate	$\omega_{x,0}$	$\frac{\text{deg}}{\text{s}}$	0	[0,0]	[0,0]
Pitch rate	$\omega_{y,0}$	$\frac{\text{deg}}{\text{s}}$	-5	[-7.5,-2.5]	[-10, 0]
Yaw rate	$\omega_{z,0}$	$\frac{\text{deg}}{\text{s}}$	-5	[-7.5,-2.5]	[-10, 0]
Mass	m_0	kg	4000	[3900, 4100]	[3900, 4100]