POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

DLR

Institute of
Space Systems

Machine Learning-based Reentry Guidance for the ReFEx Mission: Analysis and Comparison of Reinforcement Learning and Genetic Programming

Tesi di Laurea Magistrale in
Space Engineering - Ingegneria Spaziale

Author: **Riccardo Cadamuro**

Student ID: 221739
Advisor: Prof. Francesco Topputo
Co-advisors: Dr. Francesco Marchetti
Academic Year: 2024-2025

Abstract

This research applies two Machine Learning (ML) techniques, Deep Reinforcement Learning (DRL) and Genetic Programming (GP), to the task of generating the online guidance command for the REusability Flight EXperiment (ReFEx) vehicle. The focus is on the atmospheric flight phase of the reentry mission, in which the models trained via DRL and GP are used to produce real-time corrections for the reference guidance command computed beforehand. This further update stage is necessary to account for external disturbances and modeling errors. The ML models are tested and validated in the 6 Degrees Of Freedom (DOF) high fidelity simulator developed for the verification and validation of the Guidance Navigation and Control (GNC) subsystem of the ReFEx mission. Both methods are compared to each other and to the baseline optimization-based guidance algorithm to assess the performances and applicability of ML techniques to a real mission. DRL and GP are chosen for their complementary features: DRL has been used to train a Multilayer Perceptron (MLP) which results in a black-box model, whereas GP can produce a human-readable continuous and differentiable model, which is available as a symbolic expression. Results show that the DRL can deliver the same performances as the baseline guidance algorithm, while the GP achieves slightly worse results still comparable to it. Moreover, both techniques feature online execution times approximately three orders of magnitude faster than the baseline optimization-based strategy.

Keywords: Machine Learning, Deep Reinforcement Learning, Genetic Programming, Atmospheric Reentry, Real Time Guidance, High Fidelity Simulation

Sommario

Questa ricerca applica due tecniche di Machine Learning (ML), Genetic Programming (GP) e Deep Reinforcement Learning (DRL), al compito di generare in tempo reale il comando di guida per il velivolo ReFEx. Lo studio si concentra solamente nella parte di rientro atmosferico durante la quale, i modelli addestrati tramite le due tecniche sopraccitate, vengono usati per calcolare le correzioni da applicare in tempo reale al segnale di guida calcolato prima di iniziare la traiettoria di rientro. Tale fase di correzione è necessaria al fine di fronteggiare disturbi esterni ed errori nella modellizzazione del sistema stesso. Entrambi i modelli di ML sono stati testati e validati nel simulatore ad alta fedeltà sviluppato per la progettazione e validazione del sottosistema di Guida, Navigazione e Controllo (GNC) della missione ReFEx. I due metodi sono stati confrontati tra di loro e in relazione all'algoritmo standard sviluppato per la missione, il quale si basa su un processo iterativo di ottimizzazione, con lo scopo di valutare l'effettiva applicabilità di tali tecniche ad una missione reale, e le corrispondenti prestazioni. DRL e GP sono stati scelti per le loro caratteristiche complementari: il primo è stato usato per addestrare un Multilayer Perceptron (MLP) che di fatto si configura come un modello black-box mentre il secondo è in grado di produrre un modello continuo, derivabile ed interpretabile da un essere umano che si presenta nella forma di un'espressione simbolica. I risultati dimostrano che il DRL è in grado di fornire le stesse prestazioni dell'algoritmo di guida standard, mentre il GP produce risultati con prestazioni leggermente inferiori ma comunque comparabili. Inoltre, entrambe le tecniche sono in grado di generare il comando di aggiornamento in tempo reale impiegando circa un millesimo del tempo richiesto all'algoritmo standard.

Parole chiave: Machine Learning, Deep Reinforcement Learning, Genetic Programming, Rientro Atmosferico, Guida in Tempo Reale, Simulazioni ad Alta Fedeltà

Contents

Abstract **i**

Sommario **iii**

Contents **v**

1 Introduction **1**

1.1 Glided Reentry History 1

1.2 Reentry Guidance state of the art 3

1.3 Research Questions and Thesis Structure 7

2 Reinforcement Learning **11**

2.1 Markov Decision Process 11

2.1.1 Reward 12

2.1.2 Policy and Value Function 13

2.2 Training Algorithms 15

2.2.1 Reinforcement Learning methods classification 15

2.2.2 Deep Q Network (DQN) 17

2.2.3 Policy Gradient Methods 18

2.2.4 Deep Deterministic Policy Gradient (DDPG) 18

2.2.5 Twin Delayed Deep Deterministic Policy Gradient (T3D) 19

2.2.6 Soft Actor Critic (SAC) 19

2.2.7 Trust Region Policy Optimizer (TRPO) 20

2.2.8 Proximal Policy Optimization (PPO) 21

2.3 Artificial Neural Networks 23

2.3.1 Neural Network Model 23

2.4 Summary 25

3 Genetic Programming **27**

3.1 Evolutionary Strategies . . . 27
3.1.1 $(\mu/\rho \overset{+}{,} \lambda)$ - Evolutionary Strategy . . . 28
3.1.2 $(\mu + \lambda)$ - Evolutionary Strategy . . . 28
3.2 Representation of Individuals . . . 29
3.3 Initialization . . . 29
3.4 Evolutionary Operators . . . 30
3.4.1 Crossover . . . 30
3.4.2 Mutation . . . 32
3.4.3 Selection . . . 32
3.5 Open Issues . . . 33
3.5.1 Bloat . . . 34
3.5.2 Convergence . . . 34
3.6 Inclusive Genetic Programming . . . 34
3.7 Summary . . . 36

4 ReFEx: Mission and Modeling 37
4.1 Vehicle . . . 38
4.2 Reference Mission . . . 39
4.3 Training Simulator . . . 41
4.3.1 Vehicle and Environmental model . . . 42
4.3.2 Equations of motion . . . 45
4.3.3 Guidance, Navigation and Control modeling . . . 47
4.3.4 Guidance . . . 48
4.4 Testing Simulator . . . 52
4.4.1 Dynamic and Kinematic Environment . . . 53
4.4.2 Hybrid Navigation System . . . 54
4.4.3 Guidance and Control Computer . . . 54

5 Training Framework Definition 57
5.1 Guidance Law Interface . . . 57
5.2 Training Dataset Selection . . . 59
5.2.1 Dataset Generation . . . 60
5.2.2 Selection Algorithm . . . 61
5.2.3 Genetic Programming uncertainty cases . . . 64
5.2.4 Reinforcement Learning uncertainty cases . . . 65
5.3 Learning Functions Design . . . 66
5.3.1 Genetic Programming Fitness Function . . . 66
5.3.2 Reinforcement Learning Reward Signal . . . 73

5.4 Training Postprocessing 76

6 Results and Comparison 79

6.1 Experiment Description 79
6.2 Baseline Performances 81
6.3 Genetic Programming Experiments 83
6.3.1 Algorithm Settings 83
6.3.2 Training Performances 86
6.3.3 Testing Performances 89
6.3.4 Guidance Law Size 93
6.4 Reinforcement Learning Experiments 94
6.4.1 Algorithm settings 95
6.4.2 Experiments Results 96
6.5 Comparison 104
6.5.1 Guidance Performances 104
6.5.2 Techniques Comparison 107

7 Conclusions and Future developments 115

7.1 Conclusions 115
7.2 Future developments 119

Bibliography 123

A Baseline Guidance Algorithm 133

B Reference Frames 139

C Neural Network and Genetic Programming-law implementation details 143

D Flight Simulations Results 149

D.1 Baseline Results 149
D.2 RL 60UC Smooth Results 150
D.3 GP EVOL 13 Results 151

List of Figures 153

List of Tables 157

List of Symbols **160**

Acronyms **161**

Acknowledgements **163**

1 Introduction

This chapter aims to provide the background information useful to frame this work in a wider picture. It is divided into three main sections. First, section 1.1 contains a historical introduction including the most important non-propulsive reentry missions spanning from the beginning of the Space Age to the current years. Second, section 1.2 contains a reviews of the most relevant online guidance techniques. The review can be considered divided in two macro-sections: the first one contains a quick overview of classical methodologies, while in in the second a more detailed summary of Reinforcement Learning (RL) and GP based solutions is reported. The chapter ends with section 1.3, offering a more detailed overview of the objectives of the study, an introduction of the selected test case and concludes with the presentation of the research questions.

1.1. Glided Reentry History

The need to recover objects from space emerged with the dawn of the Space Age. The first class of vehicle capable of reentering the atmosphere to be developed was the space capsule: a pressurized container designed to transport humans or cargo trough space. They typically feature a blunted conical shape, are optimized for atmospheric re-entry and to guarantee aerodynamic passive stability. A crucial component of these capsules is a heat shield, made from ablative materials, which protects the spacecraft from the extreme temperatures generated during re-entry by slowly burning away. To ensure a safe return to Earth, capsules are equipped with parachute systems which are deployed in the very last phase of the descent flight in order to ensure the final deceleration and a gentle touchdown [53].

However, traditional capsule spacecrafts like Mercury, Gemini, and Apollo lacked of maneuverability: they had very little control and were highly dependent on the reentry initial conditions and on the atmospheric perturbations encountered during the flight. This lack of control promoted the development of lifting bodies and spaceplanes as alternatives.
A pivotal influence came from the X-plane program carried on in the USA. Beginning with the Bell X-1 in 1947, which has been the first vehicle to break the sound barrier, and

culminating in the X-15 hypersonic aircraft, these experimental flights yielded invaluable data on high-speed aerodynamics, materials, and control systems. The X-15 program completed almost 200 flights and allowed for the first studies on boundary-layers, materials and dual-mode control systems: allowing for a smooth transition from the aerodynamic to the reaction control.
In parallel, the U.S. Air Force pursued the Boeing X-20 DynaSoar program: an ambitious effort to create a maneuverable spaceplane. Despite being canceled in 1963, the X-20 seeded important research into vehicle structures, reentry dynamics, and guidance systems. Its influence extended to a new class of vehicles: lifting bodies, which omitted traditional wings in favor of aerodynamic fuselage shapes to generate lift.
Throughout the 1960s and 70s, NASA and the USAF tested numerous lifting-body prototypes such as the M2-F1, HL-10, and X-24 series. These unpowered and rocket-boosted aircrafts demonstrated the feasibility of controlled, unpowered reentry and precision landings—vital for future reusable spacecraft. Some, like the HL-10 and X-24B, directly informed early Space Shuttle designs by proving that wingless or low-wing vehicles could land safely and accurately after atmospheric reentry [53].

During the same period, the USSR followed a similar research program: the Mikoyan-Gurevich MiG-105 "Spiral" and its BOR test series mirrored American research, using suborbital and orbital test flights to study heat shielding, flight control, and automated landing capabilities. These test paved the way that led eventually the Soviet Union to develop and fly the Buran shuttle. The Buran vehicle was very similar to the American Space Shuttle but, differently from this, it was lighter, featured full autonomous capabilities and it used the Energiya stage: a powerful liquid rocket booster with throttling capabilities. The Buran first and only flight occurred on 1988 but the program was terminated shortly after due to the collapse of Soviet Union.
The American Space Shuttle program, initiated in the 1970s, ultimately became the first operational system to combine launch, orbital operations, and runway landing into a single reusable vehicle. Despite high expectations for cost savings, each Shuttle flight cost over 1.5 billion of dollars. However, the Shuttle's ability to perform gliding reentry from orbit and land like a conventional aircraft was an engineering triumph that relied heavily on decades of lifting-body and gliding flight research [53].

International efforts also emerged. Japan's HOPE-X project, started in the 1980s, explored lifting-body technologies through suborbital and drop tests using vehicles like OREX, HYFLEX, and ALFLEX. Although eventually canceled, these tests demonstrated hypersonic flight, heat shielding, and autonomous landing—all essential for future reusable spacecraft.

Noways, several vehicles of this kind are still under development both by government-led agencies and by private companies. The four most important players in this context are Europe, USA, China and Japan.
ReFEx, used as test case in this dissertation, is under development at Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR): it is a winged reusable uncrewed demonstrator for gliding reentry which will be launched by a sounding rocket. At the moment the maiden flight is planned by the late 2026.
Europe and in particular European Space Agency (ESA) is developing another uncrewed orbital Reusable Launch Vehicle (RLV): Space Rider, to be launched on top of a Vega-C rocket, it is a robotic vehicle that can be used to transport cargo or to perform scientific experiments in microgravity and it is expected to fly at the end of the 2025.
In the USA, the well known Boeing X-37B has already performed several orbital missions managing to land successfully on a runway the last of which finished in March 2025. The Dream Chaser vehicle by the Sierra Space Corporation [34] has already performed one atmospheric test even if the first orbital flight has still to be performed. It is designed to be carried on top of a normal rocket and to land horizontally on a runway. At the moment two variants are programmed: a cargo one and another to carry a crew composed by up to seven astronauts.
China is also operating its reentry vehicle: the Shenlong spacecraft: a robotic vehicle similar to the American X-37b [13] .
Finally, the Japanese company Space Walker is also developing a class of space planes and the tests on the WIRES demonstrator have already started [86].

Summarizing, the space industry is moving towards the paradigm of reusability: allowing upper stages to autonomously return to a recover point allows for faster, cheaper and safer missions, therefore it is of primary interest for the next generation of missions.

1.2. Reentry Guidance state of the art

Reentry guidance algorithms can be divided into two main groups [75]:

- Open-loop guidance, also known as implicit guidance, relies on the design of a reference trajectory and control profile which is computed before actually beginning the descent and it is followed during the actual trajectory. For instance, the guidance law of the Apollo missions, due to the relatively simple environment and to the low computational power available onboard, belonged to this category [75].

- Closed-loop guidance, also known as explicit guidance, refers to the class of algorithms able to compute online the reference trajectory. This computation is per-

formed based on the navigation onboard measurements, making it possible to correct iteratively the trajectory itself within a closed-loop feedback logic.

Given the complexity of the scenario under consideration, this study focuses solely on closed-loop guidance techniques. Historically, their implementation has been challenging due to the combination of demanding optimization tasks and the limited computational capabilities of onboard systems. To address this, early solutions involved generating a reference trajectory offline, which was then followed in real time. In this way it was possible to compute online the control signal but the reference trajectory stayed unaltered.

One of the first approaches to online trajectory planning was implemented in the Space Shuttle guidance program. It involved tracking a drag profile defined as a function of the vehicle's energy and it was computed prior the reentry. This profile, approximated by a piecewise quadratic function, allowed for analytical trajectory planning [48].
Another relevant example is the quasi equilibrium guidance law, developed as part of the Advanced Guidance and Control Project launched in 1999 by the NASA's Marshall Space Flight Center, which represents one of the most important examples of closed loop guidance laws [88]. It exploits the Quasi Equilibrium Glide Condition to compute the boundaries in the altitude-velocity corridor by optimizing the bank angle value in time to reach a target while satisfying some path constraints [74]. This method allows to generate on-board a reference trajectory at very low computational cost. However, the reference generated through the original implementation is not corrected during the reentry flight. An evolved version of the algorithm, capable of iteratively correcting the trajectory and to react to unforeseen disturbances is developed in [88].

Recent mathematical developments combined with the advances in computational power have led to the rise of the so called Computational Guidance and Control [38] branch characterized by three properties:

1. Closed loop Guidance and Control (GC) laws and controllers are composed by numerical algorithms;
2. The generation of the GC command requires onboard computation, often involving iterations;
3. The process of determining the GC command is data-based or model-based without requiring further information/conditions.

Trajectories are usually generated after solving an optimal control problem that can be done with methods classifiable in two main categories: direct and indirect [60].

Indirect methods apply the calculus of variations to derive a set of differential equations

that represent the necessary conditions for optimality. This leads to a multi-point boundary value problem, whose solution corresponds to the optimal trajectory sought by the original optimization. As a result, the optimal control problem is transformed into a system of nonlinear equations.

Direct methods on the other hand, approximate the state or control objective of the optimization usually by discretizing its signal. The variable to be optimized is no more a function but it becomes a vector of scalar quantities. In this way the original optimal control problem can be transcribed in the form of a Nonlinear Programming (NLP) problem and can be solved using standard NLP solvers.

Both methods however, have very high computational costs making them mostly suitable for offline trajectory generation, for this reason different solutions have been developed to be used in online applications. One of the most common among them is the so called Sequential Convex Programming (SQP) which consists in successively linearizing nonconvex dynamics and constraints related with the previous iteration's solution. The result is a convex subproblem solvable quickly and reliably with standard convex optimization solvers. This algorithm can converge generally in a small number of iterations and it is well suited for online applications [42].

SQP found several applications both addressing propulsive landing problems and glided reentry ones. It has been applied to a generalized free-final-time 6DOF powered descent guidance proving online trajectory generation capabilities [78]: the algorithm can autonomously detect when to ignite the engine during the trajectory generation process, it features an elliptical aerodynamic model which can be treated in the framework of sequential convexifications and admits the enforcement of constraints. SQP has also been successfully applied in the onboard software of the Xombie vehicle to solve a fuel optimal vertical landing problem [70].

The glided reentry problem, on the other hand, has been addressed in several studies: a SQP algorithm has been used to generate a feasible trajectory in presence of path constraints in the 3DOF environment in [47] while [56] addresses a similar problem in which no-fly zones are included and [40] provides a modified formulation of the SQP to address trajectory generation of an hypersonic reentry vehicle, always confirming its high reliability in terms of convergence and computational performances.

Differently, a lossless convexification method could be applied [1] to solve the fuel optimal Mars landing problem. In this case, the original optimization problem in presence of nonconvex control constraints, could be rewritten as a second-order convex problem dramatically reducing the computational time requested for the solution and guaranteeing

the convergence of it. This formulation, even if powerful can be applied only to some very specific classes of problems.

SQP has been successfully flight tested on the New Shepard mission [6] while, even if the specific implementation is not disclosed, also Spacex uses convex programming-based techniques in the GC algorithm design to allow for the reusability technology [41]. A more inclusive and detailed review about convex optimization applied to the trajectory generation problem can be found in [41], an historical overview is reported in [53].

In this context, ML has gained interest since it allows to train models able to generate real time guidance commands with a very low latency time. Neural Network (NN) in particular have been trained via supervised learning to mimic the behavior of optimal trajectories generated offline. Once the same NN was deployed online on unseen scenarios, it could actually guide the vehicle to the target while satisfying the path constraints [69, 85] . More advanced training techniques include DRL, which differs from standard RL by the fact that the policy or value function is represented by a Deep Neural Network (DNN).

DRL applications to spacecraft guidance and control have been studied in a wide variety of different problems and scenarios [80]. The Deep Deterministic Policy Gradient (DDPG) algorithm was used to generate the online guidance command during reentry of a 2DOF lifting body vehicle in a simplified environment (spherical, non rotating windless Earth) [21]. A more advanced application can be found in [32] where the Twin Delayed Deep Deterministic Policy Gradient (TD3) algorithm is used to derive the lateral guidance command for an Hypersonic Gliding Vehicle (HGV) modeled as a 3DOF body subjected to dynamic no fly zones in a spherical non rotating windless Earth environment. In this case the authors could enforce heat flux, dynamic pressure and load factor constraints other than the no-fly zones. DRL has been also applied to the online guidance of an upper stage launch vehicle leveraging the Trust Region Policy Optimization (TRPO) algorithm [35] and in the framework of the GC loop of an Unmanned Aerial Vehicle (UAV) (using in this case the Soft Actor Critic (SAC) algorithm) [31]. Among the most widely used DRL algorithm for GC applications, it is possible to find the Proximal Policy Optimization (PPO) (subsection 2.2.8). It is designed to handle continuous action spaces and is employed in a wide variety of control problems. For instance, it is used in [89] to train a NN that maps the spacecraft state to the optimal control policy for a robust Earth–Mars transfer. The problem is modeled within the two-body framework, accounting for uncertainties, navigation and control noise, and possible missed thrust events. PPO is also applied in [89] to tackle the Mars vertical landing problem, and in [17] to design closed-loop guidance for a transfer between two libration points in the cislunar environment, modeled using the three-body problem. These applications highlight the algorithm's

effectiveness in a wide range of GC problems involving continuous action spaces and highly nonlinear, uncertain environments. Real-time computational performance tests have shown that generating the guidance command by evaluating a trained NN requires, on average, only about 0.5 ms. When compared to the typical guidance update period of 2 ms in current engineering applications, this suggests that RL-based methods could offer significant improvements in computational efficiency. For further context, traditional guidance based on optimal control typically requires around 20 s on average to compute a solution making NN based guidance solutions interesting from the real time applications point of view [35].

Another technique belonging to the ML branch and that is gaining interest in the engineering community is GP which allows to automatically evolve an interpretable computer program to accomplish an user defined task [82]. As in the RL case, several application of GP to perform GC tasks in the aerospace domain can be found in literature even if its application in this domain are less studied with respect to the RL. GP has been used to synthesize the guidance law for an high performance aircraft in a 3DOF nonlinear environment [49] and to perform a similar task related to the an UAV in the framework of a multi objective optimization [55]. The most important advantage of GP with respect to other ML-based techniques is that the output of the optimization process is a guidance law formulated as a symbolic expression that can be read and evaluated, moreover it is possible to enforce constraints on the function behavior itself directly within the evolutionary framework, as is done in [84]. GP has been used to derive the reentry guidance law for an RLV in a nonlinear 3DOF model in presence of both aerodynamic and atmospheric disturbances, proving the GP capability to deal with uncertain environments [44]. Finally, the GP, and in particular the Inclusive Genetic Programming (IGP) algorithm, has been applied to the reentry of the ReFEx mission focusing in a simplified scenario without initial condition uncertainty. In this framework it has been proven capable of training in a 3DOF environment a guidance function successful in the 6DOF one [46].

It is worth to notice that most of the works available in literature, focuses on developing and testing these algorithms on simulators that were not developed in the context of real missions, thus limiting the validation process of these algorithms only to a specific subset of physical effects and interactions between subsystems.

1.3. Research Questions and Thesis Structure

This dissertation aims to apply DRL and GP to the design of the reentry guidance law of ReFEx, which has been chosen due to the availability of high quality and high fidelity

simulation tools developed by DLR. The work is focused on the glided reentry phase of the mission and it investigates the possibility of training a ML model able to deliver performances comparable or superior to the state-of-the-art guidance algorithm developed specifically for the mission under exam. GP and DRL have been chosen for their complementary features. In particular DRL, and RL more broadly, has demonstrated strong performance in GC applications and a substantial body of research on various RL algorithms has been produced. However, these methods typically result in black-box models, which presents significant challenges for integration into mission critical subsystems such as GNC. By contrast, GP generates human interpretable models and can be configured to produce concise, easily understandable expressions. Nevertheless, it is primarily used for symbolic regression tasks, and its application within the GC domain remains relatively uncommon.

Most importantly, the study focuses in assessing the performances of the two techniques on the same high-fidelity simulator (described in section 4.4) which is the software used to certify the actual state-of-the-art algorithm used in the real mission. The availability of this simulator is arguably the most important feature of the analysis since it allows to gain extremely meaningful insights on the actual behavior of the ML models when applied in real-world scenarios.

In second place, the present dissertation will feature a direct comparison between RL and GP with the objective of highlighting strengths and weaknesses of both techniques. This will be done always keeping as third reference the nominal guidance algorithm with the objective of driving future research and design trade-offs.

Research Questions

The following main research questions have been formulated:

- **[RQ 1]**: How can a RL or GP based model be trained to generate the closed loop online guidance signal for a reentry mission?
- **[RQ 2]**: To what extent can machine learning–based online guidance algorithms (specifically RL and GP) match or exceed the performances of a state-of-the-art guidance law when evaluated in a high-fidelity flight dynamics simulator?
- **[RQ 3]**: What are the comparative strengths and limitations of RL and GP in the context of online guidance for atmospheric reentry missions?

Thesis structure

The dissertation is organized as follows: chapter 2 and chapter 3 contain a theoretical

review respectively of RL and GP to detail their functioning and delineate the concepts that will be necessary to apply these techniques to the actual reentry guidance problem. Chapter 4 delves into the details of the ReFEx case study, putting the accent in first place to the vehicle and to the mission profile (section 4.1 and section 4.2), and in second place to the simulation tools that have been used during the work, distinguishing between a simplified 3DOF environment used for training purposes (section 4.3), and a high-fidelity 6DOF environment used for the final testing (section 4.4). Chapter 5 illustrates how GP and RL training environments have been implemented to address the reentry guidance problem: it details which are the input and outputs of the ML model (section 5.1) and how the training dataset is built (section 5.2), goes into the design details of the GP fitness function and RL reward signal (section 5.3) and concludes with section 5.4 showing how the training postprocessing leads to the selection of the exact model to be tested on the validation simulator is executed. Chapter 6 contains the work results: it analyzes individually both GP and RL based solutions (section 6.3 and section 6.4) and performs the comparison between the two techniques (section 6.5). To conclude, in chapter 7 the research questions are finally addressed and some insights into possible future developments are provided.

2 Reinforcement Learning

RL is a ML technique used to train an agent to generate an optimal sequence of actions that maximize a numerical reward signal. This reward signal reflects the performance of the agent that seeks to improve it by iteratively interacting with the environment while learning the optimal policy function: a function mapping states to feasible actions.
RL differs from supervised learning, where the learning process relies on a labeled dataset containing input-output pairs. In supervised learning, the model learns to reproduce the correct output for each input by minimizing the error with respect to the known labels. In contrast, RL does not receive explicit input-output pairs or correct actions; instead, it must discover the optimal behavior solely through feedback in the form of rewards.
RL also differs from unsupervised learning, which focuses on uncovering hidden patterns or structures in data without any external feedback or labels: consider for example clustering or dimensionality reduction problems [93]. In RL, the objective is not to find structure in the data but to learn behaviors that maximize the reward function which serves as a form of evaluative feedback rather than instructive guidance [77].

2.1. Markov Decision Process

A RL algorithm can be formalized and studied in the framework of the Markov Decision Process (MDP), that represents the problem of learning by interacting with the environment [59, 77].
A system controlled by a decision maker is observed during its evolution in time, each time instant in which the system is observed can be denoted by t. The key element of the MDP is the decision maker that learns how to achieve the prescribed goal by iteratively interacting with the environment. The decision maker is referred to as agent. The agent interacts with the environment by performing at each time step t an action which shall belong to a given feasible set which can be both finite or infinite. At each time step t, the agent obtains from the environment a representation of the system state $s_t \in S$ and based on that it perform an action $a_t \in A(s)$ where $A(s)$ is the set of feasible actions. One time step later, the agent receives a reward: R_{t+1} and finds itself in a new state s_{t+1}.

The agent-environment interaction in the framework of an MDP will therefore lead to a trajectory with the following shape: $s_0, a_0, R_1, s_1, a_1, R_2, s_2, a_2, R_3, ...$ However, in most cases, the agent will not be capable of having ideal observations of the full system state, but it will be provided with partial observations o_t which will depend on the state.
In this framework, the probability of transition to state s' with reward r from state s and action a is computed by definition as follows:

$$p\left(s', r \mid s, a\right) = Pr\left(S_t = s', R_t = r \mid S_{t-1} = s, A_{t-1} = a\right) \tag{2.1}$$

where p is called dynamics function. In a MDP, the probabilities computed trough p fully characterize the environment's dynamics.
It is also possible to compute the expected reward for state-action pairs as:

$$r\left(s, a\right) = \mathbb{E}\left(R_t \mid S_{t-1} = s, A_{t-1} = a\right) = \sum_{r \in \mathbb{R}} r \sum_{s' \in \mathbb{S}} p\left(s', r \mid s, a\right) \tag{2.2}$$

The most important elements and terminology related to the RL problem are described hereafter.

2.1.1. Reward

As above mentioned, the most important feature of the RL with respect to other learning paradigms is its formulation prescribing what performances need to be achieved by the agent but without specifying how to achieve them. This information is encoded in the so-called reward signal that is communicated from the environment to the agent at each time step as consequence to its action.
The purpose of the RL is to maximize the cumulative reward, i.e. the summation of all the single rewards values obtained at each timestep. This task comes particularly natural when the agent-environment interaction can be framed between a specific initial condition and a specific final one; a run from the given initial condition to the terminal one is called episode or trial. The initial conditions of each new episode are independent from the terminal conditions of the previous one. Different episodes can be imagined as different plays of the same game, after one play ends, the initial conditions are reset and the following one starts. For instance, when considering the reentry problem, each episode can be defined by a single element in the uncertainty set of the parameters, therefore the full training will aim to include enough scenarios to obtain a sufficiently realistic representation of the real environment.

In the most simple case, a RL algorithm aims to maximize the expected return after time

step t, that is computed as:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \ ... \ + R_T \tag{2.3}$$

where T refers to the final timestep of the episode [77].
In actual applications, the objective is to maximize the so called discounted reward. Computing the discounted reward instead of the standard cumulative reward in Equation 2.3 is a way to prioritize the weight of future rewards, this can be done by computing the expected discounted reward as:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ ... \ = \sum_{i=0}^{T} \gamma^k R_{i+t+1} \tag{2.4}$$

where the discount rate γ has value between 0 and 1. When $\gamma = 0$ the agent is concerned on maximizing only the immediate reward without giving importance to the future one, on the other hand, if γ approaches 1, the agent gives more importance to future rewards acting in a more farsighted way. In this way it is possible to tune how much we want to prioritize future rewards with respect to present ones. A common rule of thumb is considering $\gamma \geq 0.98$.

2.1.2. Policy and Value Function

The learning of the optimal behavior involves the estimation of a value function that estimates how desirable is for the agent to be in a given state; this concept of desirability is evaluated in terms of expected return. Since the expected return shall be computed for state-action pairs, value functions are defined with respect to specific ways of acting, called policies. A policy is a map from states to the probability of selecting each of the feasible actions as follows:

$$\pi\left(a \mid s\right) = Pr\left(A_t = a \mid S_t = s\right) \tag{2.5}$$

The value function of a state s under a policy π, denoted as $\nu_\pi\left(s\right)$ is the expected return when starting in s and following π thereafter [77]. The objective of the RL is therefore to learn the optimal policy that maps from the system state to the action. RL methods specify how the policy is evolved due to experience.

The state-value function for policy π can be computed as:

$$\nu_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s\right] \tag{2.6}$$

It might be useful to assess how good it is for the agent to perform a given action in a given state, and following the policy π thereafter. The function associated to this concept is called action-value function for the policy π and is defined as:

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s|, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} | S_t = s, A_t = a\right] \tag{2.7}$$

As above mentioned, the ultimate objective is to compute the optimal policy function π^*: a policy π' is better than π if its expected return is higher implying that $\nu_{\pi'}(s) > \nu_\pi(s)$. The optimal state-value function denoted by $\nu_*(x)$ is then defined as in Equation 2.8.

$$\nu_*(s) = \max_\pi \nu_\pi(s) \tag{2.8}$$

In a similar way the optimal action-value function represented by q_* is defined as in Equation 2.9.

$$q_*(s,a) = \max_\pi \; q_\pi(s,a) \tag{2.9}$$

The agent will select the action with the highest q value even if it might be suboptimal since the optimal function q_* is not available until the end of the training. The function $q_*(s,a)$ in Equation 2.9 needs to satisfy the well known Bellman optimality equation which can be formulated in this specific framework as:

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} \; q_*(s_{t+1}, a')\right] \tag{2.10}$$

During the training, at a given timestep, the predicted q value can be computed as

$$q_{predicted}(s,a) = \mathbb{E}\left[\sum_{k=0}^{T} \gamma^k R_{t+k+1}\right] \tag{2.11}$$

which is namely the expected discounted reward starting from the state s, performing the action a and then following the current policy. The predicted q value is then used to compute the loss function relative to the current optimal action value function q_* as follows:

$$L = \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(s_{t+1}, a')\right] - q_{predicted} \tag{2.12}$$

In this way the RL algorithm can iteratively improve the estimation of the function q_* while the training goes on [3, 77]. It is finally noteworthy to report in Equation 2.13 the expression of q_* as function of ν_*.

$$q_*(s,a) = \mathbb{E}\left[R_{t+1} + \gamma \nu_*(s_{t+1}) \mid s_t = s, a_t = a\right] \tag{2.13}$$

2.2. Training Algorithms

The focus is now on the the optimal policy, optimal value function and optimal action value functions with the objective of detailing how to actually represent and compute them. This section will then provide some details about the specific RL algorithm that can be used to train an agent. We talk about DRL when a DNN is used to represent one or more between:

- State;
- Observations;
- State value function $\hat{\nu}(s\ ;\ \boldsymbol{\theta})$;
- Action value function $\hat{q}(s,\ a;\ \boldsymbol{\theta})$;
- Policy function $\pi(s \mid a;\ \boldsymbol{\theta})$;
- Model.

In the previous bulleted list, the symbol $\boldsymbol{\theta}$ represents the weights of the neural network. DRL is generally used when it is necessary to implement a NN to estimate with acceptable accuracy the value function or the policy, this usually happens in problems with an infinite number of possible states and continuous action [37], that is actually the reentry guidance one.

2.2.1. Reinforcement Learning methods classification

Since its initial formulation, a lot of different RL variations have been developed during the years with different characteristics.

Model-Free and Model-Based

The first subdivision can be made based on the agent's knowledge of the environment.

In the Model-Based case, a model of the environment is available to the agent and this

allows it to foresee the effects of its actions: it is therefore possible to compute the results in advance and then planning the strategy. This strategy actually uses the probability distribution and expected reward function related to a MDP defined in Equation 2.1 and Equation 2.2. This approach can learn the value/policy function efficiently, but suffers from identification problems, in particular if the estimated model is not accurate, the actual performances might be much worse than the simulated one [37].
Since the present work deals with highly uncertain dynamics, this approach will not be used.

In the Model-Free case instead, the transition probability distribution and expected reward function related to the MDP are not used during the training since these functions are referred to the model of the environment. When using this approach, the agent learns by trial and error while interacting with the ambient itself, usually requiring a larger number of samples with respect to the Model-Based approach. On the other hand, this approach allows to obtain a trained agent more robust to environment uncertainties [77].

Value-Based and Policy-Based

At each timestep, the agent can choose the action in two ways: by following a policy or by computing the action with the highest long-term reward. Hence, Model-Free RL methods splits in two also from this point of view.
A Value-Based method focuses on finding the optimal action value function, the objective is therefore to maximize the function representing the long-term cumulative reward. The optimal action value function is usually computed recursively using the Bellman equation. The most common method in this category is the Q-Learning with its variants and derivatives. Once the optimal value function is available, the optimal policy might be computed. The most important advantages of these methods is that they are highly effective with small actions spaces but on the other hand, storing all the Q-values could become an issue when the action space increases. Value-Based methods are usually referred as Critic methods [90].
Policy-Based methods instead, aim to learn the optimal policy $\pi(a \mid s)$ mapping from states to actions without computing an estimation of the value function. This class of methods directly updates the parameters describing the policy. In general policy based methods have better convergence properties and are effective also in high dimensional or continuous action spaces with also the possibility to learn stochastic policies. On the other hand, they can lead to high variance and may converge to local minima.
Since the reentry guidance problem addresses a continuous action space, this class will be of interest in the present work. Policy-Based methods are usually referred ad Actor methods [90].

Furthermore, a third option exists which combines the features of the two methods aforementioned. Actor-Critic methods combines the merits of value-based with the merits of policy-based ones. A method of this kind is designed to use a value-based method to learn the q function and to use a policy-based method to learn the π function. They can be regarded as an extension of the value based methods in continuous space or as an improvement of the policy based ones to reduce the sampling variance [90].

On-Policy and Off-Policy

A further classification can be made according to the way the method represent the optimal policy. On-Policy methods evaluate and improve directly the policy, the training will then be used to adjust greedily the parameters of the policy itself. In *on-policy* methods, the agent is only learning from its own trials and errors, in the sense that at each iteration, the current policy function is evaluated and improved only based on the performances of the policy itself.
On the other hand in Off-Policy methods the agent can learn independently from the action performed with the current policy, for example by observing the behavior obtained following the different policies. This kind of methods doesn't require to compute the full episode but for example can reuse also past trajectories [90].

The most important RL algorithms will then be introduced in the next sections.

2.2.2. Deep Q Network (DQN)

The Deep Q Network (DQN) method [52] reenters in the category of the value based methods since its objective is to estimate the action value function in Equation 2.7 using the Bellman equation in Equation 2.10. The value function is approximated through a NN by $q(s, a;\ \theta) \approx q^*(s, a)$ (where θ represents the NN parameters) which is called Q-network.
DQN uses experience replay to improve its training performances. It basically consists in storing in a dataset the experience of the agent at each timestep, represented by the tupla $e_t = (s_t, a_t, r_t, s_{t+1})$, and the learning process is performed by sampling from the above mentioned dataset. Sampling from the replay database allows for a more stable training, better data efficiency and enforces the usage of Independent and Identically Distributed (IID) samples since they are not consequently generated during the same trajectory. An action is then selected based on an ϵ-greedy strategy. The exploitation of the experience replay force the method to be off-policy since samples produced with different policies might be used. Finally, DQN is model-free because the learning process is performed by interacting with the environment without ever creating an actual model

of it.

2.2.3. Policy Gradient Methods

Policy gradient methods are a set of policy-based methods where the agent learns the policy function $\pi(a|s;\ \theta)$ directly. The parameters θ of the policy function are updated via gradient ascent during the training with the objective of maximizing the expected cumulative discounted reward in Equation 2.4. In this context, the objective of the optimization is:

$$J(\theta) = \mathop{\mathbb{E}}_{\pi_\theta}\left(\sum_{i=0}^{T} \gamma^i R_i\right) \tag{2.14}$$

And the parameters of the policy, usually the weights and biases of the NN are updated based on the stochastic gradient ascent applied on the function in Equation 2.14 as shown in Equation 2.15.

$$\theta_{i+1} = \theta_i + \alpha\ \nabla J(\theta_i) \tag{2.15}$$

The main advantage of policy-based methods with respect to value-based ones is that they are well suited to deal with continuous action spaces without having to consult any value function to select the action [77].

2.2.4. Deep Deterministic Policy Gradient (DDPG)

The DDPG is an actor-critic policy gradient method that learns at the same time a Q-function and a policy one. The policy function is computed by the estimated Q-function that is learnt with off-policy data trough the Bellman equation in Equation 2.10. The Q function is used to update the policy function parameters θ^π, which differs from the value function parameters θ^q according to Equation 2.16.

$$\nabla_{\theta^\pi} J \approx \mathbb{E}\left[\nabla_{\theta^\pi}\ Q(s, a|\ \theta^q)\,|_{s=s_t,\ a=\pi(s_t|\theta^\pi)}\right] \tag{2.16}$$

Similarly to the DQN, replay buffers are used to update the critic network (i.e. $Q(s, a, |\theta^q)$ making the DDPG an off-policy algorithm. Since the Q-network being updated is also used to compute the target value for the Bellman equation, this passage is prone to divergence. The problem is solved by using soft target updates consisting in updating the parameters of the target network with a lower frequency. By computing the optimal action via policy function π, the DDPG is well suited to handle continuous action spaces. Unfortunately, while this algorithm can achieve great performances, it is usually extra sensitive to tuning parameters. Another problem of the DDPG is that usually the learned

Q-function quickly raises, resulting in a very relevant overestimation of the actual Q-value. This can lead the actor to fail because it starts exploiting the errors in the Q-function.

2.2.5. Twin Delayed Deep Deterministic Policy Gradient (T3D)

TD3 tackles the issues highlighted at the end of the DDPG section focusing in particular on the one related to the overestimation of the Q-value [20] by using a set of techniques. First, the critic part is substituted by a set of two learners that estimate independently the Q-value: the smaller predicted value is used.
The critic learns two Q-functions: $Q(s, a; \theta_1)$ and $Q(s, a; \theta_2)$, since the actions used to evolve the critic are selected by randomly sampling mini-batches of transitions from the experience replay buffer, it is possible to keep the estimation of the two Q-functions independent from each other. At this point, both Q-functions are evolved towards the same target which is computed according to the Bellman equation Equation 2.10 using the one of the two Q-functions with the lower value.
Secondly, the policy and target networks are updated with a frequency lower than the one used to update the Q-value estimation, similarly to what it was done on the DDPG method. Finally a noise is added to the target action value. This is done in order to avoid (or at least make it less probable) the policy to exploit errors on the Q-function estimation: the noise actually smooths the value function estimation and avoids the policy to overfit errors in it [20]. As in the DDPG case, it can be classified as an actor-critic, mode-free, off-policy algorithm.

2.2.6. Soft Actor Critic (SAC)

SAC is an actor-critic, off-policy algorithm that optimizes a stochastic policy. In this method a central role is played by the entropy regularization: the policy is trained to obtain a tradeoff between the expected return and the the randomness of the policy itself (i.e. its entropy) as it is shown in Equation 2.17 [28].

$$\pi^* = \max_{\pi} \mathop{\mathbb{E}}_{\tau \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t \left(R\left(s_t, a_t, s_{t+1}\right) + \alpha \; H\left(\pi\left(\cdot \mid s_t\right)\right)\right) \right] \tag{2.17}$$

where the entropy H for a Probability Density Function (PDF) P is defined as in Equation 2.18[91].

$$H(P) = \mathop{\mathbb{E}}_{x \sim P} \left[-log \; P(x) \right] \tag{2.18}$$

The coefficient $\alpha > 0$ in Equation 2.17 is a tradeoff parameter that needs to be properly tuned.
In a similar way, it is possible to define slightly different state value and action value functions accounting for the entropy contribution scaled by the coefficient α. The actual SAC algorithm learns the policy π_θ and two different Q-functions: Q_{θ_1} and Q_{θ_2} analogously to the TD3. Similarly to it, both Q-functions are learnt by regression to a single shared target which is computed using the Q-network that predicts the smaller Q value. On the other hand, the SAC algorithm shows some differences with respect to the TD3. The entropy term is always considered in the target network computation and there is no smoothing process in the policy estimation: this can be avoided since the SAC algorithm estimates a stochastic policy, hence the effect that on TD3 was obtained by adding the random noise, it is now a consequence of the uncertainty in the policy output. Learning a stochastic policy with entropy maximization is actually shown to greatly increase the stability of the training. Once the training process is over, the stochastic part of the resulting policy is removed by using the mean action: this in general improves the performances of the original policy [27, 28].

2.2.7. Trust Region Policy Optimizer (TRPO)

TRPO is an actor-critic, on-policy method which aims to provide a guaranteed monotonic improvement on the policy function [71].

Policy gradient methods usually compute the gradient of the policy $\hat{g}$ using the estimator in Equation 2.19 where $\hat{A}_t$ is an estimator of the Advantage function defined as in Equation 2.20 [36] and the expectation operator $\hat{\mathbb{E}}_t$ represents the sample average over a finite batch of samples. The Advantage function is a measurements of how good an action is if compared to other feasible actions at a given state [3].

$$\hat{g} = \hat{\mathbb{E}}_t \left[\nabla_\theta \, log \, \pi_\theta \left(a_t \mid s_t \right) \hat{A}_t \right] \tag{2.19}$$

$$A_{\pi_\theta} = Q_{\pi_\theta} \left(s, a \right) - V_{\pi_\theta} \left(s \right) \tag{2.20}$$

The Policy Gradient in Equation 2.19 is then used inside a Stochastic Gradient Ascent algorithm to maximize the value of the action commanded by the policy. Automatic implementations deals with the cost function $L^{PG}\left(\theta\right) = \hat{\mathbb{E}}_t \left[log \, \pi_\theta \left(a_t \mid s_t \right) \hat{A}_t \right]$ to be maximized, whose gradient is Equation 2.19. The problem with this procedure is that simply following the gradient, often leads to too large policy updates that can lead the algorithm to numerical instability and ultimately to fail to converge [72].

TRPO solves this problem maximizing an objective function that is subjective to a policy update size constraint.

$$\begin{cases} L^{TRPO} = \hat{\mathbb{E}}_t \left[\dfrac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)} \cdot \hat{A}_t \right] \\ \text{s.t.}: \ \hat{\mathbb{E}}_t \left[KL \left[\pi_{\theta_{\text{old}}} \left(\cdot \mid s_t \right), \pi_\theta \left(\cdot \mid s_t \right) \right] \right] \leq \delta \end{cases} \tag{2.21}$$

where $KL[\cdot, \ \cdot]$ represents the *Kullback-Leibler divergence* metric that is an index used to quantify the difference between a given probability distribution and a reference one. The term $\dfrac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)}$ represents instead the ratio between the probability of taking a given action with the updated and previous policy; in the cost function it is multiplied by the Advantage function in order to quantify the beneficial (or not) effect of the variation in the above mentioned probability with respect to the training objective [71, 72].

2.2.8. Proximal Policy Optimization (PPO)

PPO, as TRPO, is an on-policy actor-critic algorithm and it can be considered an improved version of the latter [72]. The PPO algorithm tries to achieve similar results to TRPO but with a simpler implementation. The probability ratio term $\dfrac{\pi_\theta \left(a_t \mid s_t \right)}{\pi_{\theta_{\text{old}}} \left(a_t \mid s_t \right)}$, denoted by the function $r_t(\theta)$ (e.g.: $r_t(\theta_{\text{old}}) = 1$), appears in the TRPO cost function as:

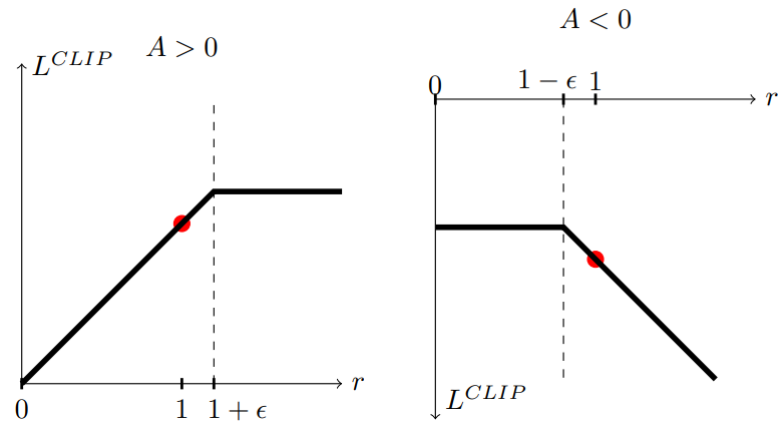
$$L = \hat{\mathbb{E}}_t \left[r_t(\theta) \ \ \hat{A}_t \right] \tag{2.22}$$

Obviously, without any other constraint, this cost function will lead to very large policy update steps that ultimately will make the algorithm numerically unstable as already discussed.
To enforce the constraint on the step size, a different cost function is evaluated:

$$L_t^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t \ , \ \text{clip} \left(r_t(\theta), 1 - \epsilon, 1 + \epsilon \right) \right) \hat{A}_t \right] \tag{2.23}$$

where ϵ is a tuning hyperparameter (a good value has been noted to be equal to 0.2).
The advantage of a cost function computed in this way, is that increments in the policy performance are bounded to be $1 + \epsilon$ times the probability ratio, in this way the effects of policies leading to too high improvements are not considered. This is reasonable since at this stage, each improvement is based on just a few samples therefore it is not statistically meaningful. On the other hand, by taking the minimum in Equation 2.23 there is no inferior limit to the cost value, therefore each update that strongly worsen the policy performances is actually evaluated (Figure 2.1).

The result is producing only small but "safe" improvements in the policy avoiding therefore numerical instabilities but maintaining an easy implementation. This simpler implementation, still manages to keep the KL metric of the update step under control without actually the need to compute it.



(a) Clipped function when $\hat{A} > 0$ (b) Clipped function when $\hat{A} < 0$

Figure 2.1: PPO clipped objective function [72]

Typically in the PPO the policy and value function updates are carried out all at once by gathering in the set of parameters θ the weights and biases of both the actor and critic networks [89] and including in the objective function a terms that accounts for the error in the value function estimation that is computed according to Equation 2.24. The latter objective can further augmented by adding an entropy bonus to ensure sufficient explorations in Equation 2.25 finally leading to the surrogate objective function in Equation 2.26 where c_1 and c_2 are coefficients used to weight the relative importance of the various terms.

$$L_t^{VF}(\theta) = \frac{1}{2} \cdot \mathbb{E}\left[\frac{1}{N}\sum_{k=0}^{N-1}\left(V_\theta(\boldsymbol{s}_k) - \sum_{k'=k}^{N-1}\gamma^{k'-k}R_{k'}\right)^2\right] \tag{2.24}$$

$$S_{\pi_\theta}(\boldsymbol{s}_t) = \mathbb{E}\left[\frac{1}{N}\sum_{k=0}^{N-1}\mathbb{E}_{\boldsymbol{a}\sim\pi_\theta}\left[-log\ \pi_\theta(\boldsymbol{a}|\boldsymbol{s}_k)\right]\right] \tag{2.25}$$

$$L_t^{FULL}(\theta) = \left[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S_{\pi_\theta}(\boldsymbol{s}_t)\right] \tag{2.26}$$

The overall learning process consists in two different phases that are repeated iteratively:

1. Policy Rollout: the current policy π_θ is run in n_{env} independent realization of the environment for n_b training episodes each of them of length N. This process allows to obtain a set of $n_{env} n_b$ trajectories.

2. Policy Update: this is done by running n_{opt} epochs of the Stochastic Gradient Descent algorithm over n_b sampling mini-batches of on-policy data coming from the last rollout.

It is useful to apply the Gradient Descent Algorithm dividing the dataset in mini-batches to obtain a tradeoff between the efficiency of the Stochastic Gradient Descent and the stability of the Batch Gradient Descent. PPO can be considered a *model-free* method, since it works by sampling the environment without identifying it and optimize directly the policy iteratively.
Once the training is completed, during the deployment phase, the stochastic component is removed from the policy function and the action is selected only according to the mean predicted value without considering the standard deviation. For these reasons, PPO is one of the most used RL methods when it comes to solve continuous action control problems.

2.3. Artificial Neural Networks

As previously mentioned, state-of-the-art methods in RL involve training either a policy function π_θ or an action-value function Q_θ to guide the agent in selecting the optimal action at each state. For problems involving infinitely many possible states and continuous actions, exact tabular solutions are impractical due to the so-called curse of dimensionality, making approximation necessary. A particularly effective class of nonlinear function approximators is represented by Neural Networks (NNs), which are employed in all the state-of-the-art RL implementations discussed earlier.
The aim of this section is therefore to provide an overview of the theory and implementation of Artificial Neural Networks [7].

2.3.1. Neural Network Model

A Neural Network is a model structurally and functionally inspired by biological neurons. It consists in a set of nodes called neurons connected by weighted edges organized in layers as shown in Figure 2.2. Each neuron receives a real value signal from all the connected neurons and its output is

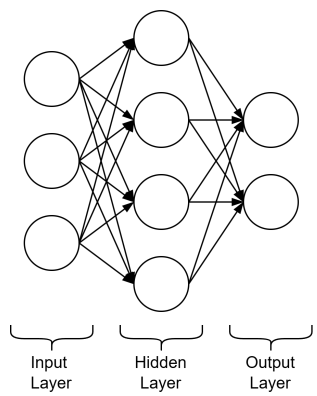


Figure 2.2: Neural Network Model

computed by applying a nonlinear function, called
activation function to the sum of the inputs plus
a bias. When routing a neuron output signal to the following one, the signal magnitude is multiplied by the weight of the edge, which is optimized during the learning process. The basic NN model can be seen a series of functional transformations. Considering a network with D neurons in the input layer and M neurons in the first layer, the first step consists in computing M linear combinations of the input variables $x_1, ..., x_D$ as follows:

$$a_j = \sum_{i=1}^{D} w_{ji}^{(1)} x_i + w_{j0}^{(1)} \quad \text{for } j = 1, ..., M \tag{2.27}$$

where the superscript (1) indicates that the corresponding parameters are in the first layer of the network. The parameters $w_{ji}^{(1)}$ are the weights of the connection between the node i in the input layer and the node j in the output one, while the parameters $w_{j0}^{(1)}$ are the biases. The quantities a_j are known as activations: each of them is then transformed using a differentiable nonlinear activation function $h(\cdot)$ to obtain:

$$z_j = h(a_j) \tag{2.28}$$

The activation functions are generally chosen to be sigmoidal function such as the logistic sigmoid or the *tanh*; in more recent times also the ReLU (Rectified Linear Unit) activation function is frequently employed. The values z_j are then used as input for the following layer as shown in Equation 2.29:

$$a_k = \sum_{j=1}^{M} w_{kj}^{(2)} z_j + w_{k0}^{(2)} \quad \text{for } k = 1, ..., K \tag{2.29}$$

where K is the number of neuron in the second layer. This structure is iterated until the final output layer.
Notice that in a network, not all the possible connections between neurons of next layers must be present.

Following an introduction to the most relevant NN architectures will be provided.

MLP

They can be referred also as Fully Connected Network and are the simplest possible architecture. They have a structure analogous to the one in Figure 2.2, the network graph does not feature any closed directed cycle nor any memory component is included. A standard fully connected network is composed by an input layer with n_I nodes and an

output layer with n_O nodes. If some hidden layers between the input and the output one are present, each with n_i nodes, the network can be defined a deep network.

Recurrent Neural Network (RNN)
Unlike Feed-Forward NN, in recurrent architectures, data are processed across multiple time steps. The fundamental block of RNNs is the recurrent unit that has an hidden state which acts as a form of memory and it is updated ad each time step based on the current input and previous hidden state. In this way, the network can learn from past inputs and account for it into the current processing. The standard network architecture is called Long short-term memory (LSTM). This kind of network is usually employed in speech recognition, language processing and handwriting recognition.

Transformer neural network
Transformer is a modern architecture of NN design to process sequential data using attention layers instead of recurrent ones. This improve the performances of the network in capturing long-distance dependencies between tokens in the input sequence. The core component in the Transformer architecture is the attention head that allows to determine which tokens in the input sequence are most relevant to each other allowing to capture the above mentioned long-distance dependencies. An inclusive description of the Transformer architecture is outside the scope of this thesis but refer to [83] for the original implementation.
So far Tranformers even though they have been initially developed in the context of Natural Language Processing and computer vision, they have applied to a variety of different fields including the guidance problem in aerospace domain [10, 11, 25, 79].

2.4. Summary

In conclusion, the present work will adopt the PPO as training algorithm in theDRL methodology. It has been chosen since it is designed to handle a continuous multidimensional action space and has provided good results in a wide variety of test cases belonging to the aerospace GC domain [10, 17, 23, 89].
The agent is represented by a MLP with three hidden layers. The choice of using a simple fully connected network with respect to more advanced architectures has been done for two reasons:

1. For the sake of a proper comparison with the GP it is necessary to have agents behaving in similar way. As will be seen in chapter 3 and section 6.3, the GP function produces the output by operating only on the input elements, without any

explicit loop or memory component, therefore the MLP has been considered to be the most suitable architecture to have a fair comparison;

2. From the implementation point of view it was simpler to use a MLP in the high-fidelity testing simulator with respect to more advanced architectures.

Finally, it is worth to notice the fundamental importance of writing a reward function that allows the agent to receive a reward even if the simulation cannot reach the terminal conditions or diverges in the very first stages. This is necessary since the initialization of the agent policy will result in semi-random actions by the agent that, without a properly shaped reward function, cannot evolve the weights and biases of the NN in order to solve the problem as expected. The most suitable strategy to design the reward function is to have a piecewise function in which each term accounts for a certain phase or constraint violation/satisfaction.

3 | Genetic Programming

According to Koza [33], GP is *a systematic method for getting computers to automatically solve a problem starting from a high-level statement of what needs to be done*. The high-level statement is formulated in terms of a fitness function which is used to evaluate the performances of a program in solving a given problem.
The nominal GP workflow is summarized in Algorithm 3.1[82].

Algorithm 3.1 GP workflow

```
Initialize population of programs (individuals)
while Termination == false do
    Execute each program in the population and evaluate its fitness value
    Update the population according to the Evolutionary Strategy (ES)
end while
The best program in the last generation is the solution
```

3.1. Evolutionary Strategies

The purpose of this section is to provide an overview of the evolutionary strategies that are commonly used to evolve the population toward the prescribed objective.
The usual goal of an evolution strategy is to optimize some objective function F with respect to a set of decision variables often called object parameters x which in general can be any dataset of finite length. Evolutionary strategies operate on a population $\mathbb{P}$ of individuals $\mathbb{A}$. Each individual $\mathbb{A}_k$ is defined by its set of parameters $\boldsymbol{x}_k$, its fitness value $\boldsymbol{f}_k$ and also by a set of evolvable strategy parameters $\boldsymbol{s}_k$. The latter set of parameters is used to control certain statistical properties of the genetic operators that will be described in section 3.4. In general, during one evolutionary step, the initial population $\mathbb{P}$ which is composed by μ individuals $\mathbb{A}_m$, is used to generate λ offspring individuals $\mathbb{A}_l$. Usually $\lambda > \mu$ therefore the next generation population will be created applying some operation to the $\mathbb{A}_l$ individuals in order to keep μ constant.
Two ES will be analyzed in particular: the standard $(\mu/\rho \overset{+}{,} \lambda) - ES$ and a variant known

as $(\mu + \lambda) - ES$ [5].

3.1.1. $(\mu/\rho \overset{+}{,} \lambda)$ - Evolutionary Strategy

In the $(\mu/\rho \overset{+}{,} \lambda)$ evolutionary strategy, ρ refers to the number of parents involved in the procreation of each children. The "+" symbol means that the parents individuals are involved in the selection process while the "," symbol on the other hand means that they are not, therefore only theλ offspring will be considered. The notation $\overset{+}{,}$ is used to include both cases in a single representation [5].

3.1.2. $(\mu + \lambda)$ - Evolutionary Strategy

The $(\mu + \lambda)$ evolutionary strategy keeps $\rho = 1$ even if in general some kind of recombination is implicitly included (for example trough crossover as described in subsection 3.4.1), and the selection is performed considering both the parents μ and the offspring λ. This is done in order to give the algorithm the possibility to preserve the best genotypes obtained until the current generation [5].
The *standard* $(\mu + \lambda) - ES$ algorithm is described in Algorithm 3.2.

Algorithm 3.2 Standard $(\mu + \lambda)$

```
Initialize ℙ with μ individuals
Initialize the strategy parameters
i = 1
while Termination == false do
    while i ≤ λ do
        Select one or two parents solution randomly from ℙ
        Create a new offspring by applying crossover or mutation to the chosen parents
        i ← i + 1
    end while
    Compute the fitness value of each offspring
    Select μ individuals from the the combined pool of the individual in ℙ and the
    newly generated λ offspring
    Update the population ℙ
    Update the strategy parameters
end while
Return the best solution
```

3.2. Representation of Individuals

Each individual (program) is represented by a tree (Figure 3.1) a tree is built with elements belonging to two different sets [82]:

- The primitive set $\mathbb{F}$ which contains the set of functions that can be used by the program itself. This set of functions may contain the elementary mathematical operators `(+, -, ×, /)` as well as functions `(sin, cos, exp, log, sqrt, tanh, ...)`, logic operators `(AND, OR, NOT, XOR)` and conditional `( If-Then-Else )` or iterative operators `(for, while-do)` but also other set of functions that may be specifically defined by the users. The number of arguments a function takes as input is called *arity*.
- The terminals set $\mathbb{T}$ containing the variables or constants used by the GP algorithm. These represent the leaves of the tree.

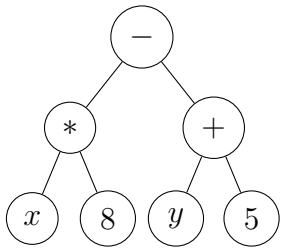


Figure 3.1: Depiction of a GP individual corresponding to the function $8x - (y + 5)$.

Functions and terminal sets shall satisfy the closure and sufficiency requirements. The first property requires that each function in $\mathbb{F}$ accepts as argument any value and data type that may be returned by any function in $\mathbb{F}$ and any value and data type that may be returned by the evaluation of any element in $\mathbb{T}$. This is essential because a failure in one individual within the program population can lead either the whole GP process to failure or can lead it to unpredictable results. This property can be enforced by forcing each function to call only functions whose input argument types match the output ones. In this way also the search space gets reduced and the probability of success in finding the optimal solution increases. The second property requires instead that the set of functions and the set of terminals are actually sufficient to express a solution to the problem [82]. In many cases it is impossible to theoretically prove this property, the choice of the elements in the $\mathbb{F}$ and $\mathbb{T}$ set will therefore depend on the experience and knowledge of the designer.

3.3. Initialization

The GP evolutionary process starts with the population initialization, i.e. the generation of the initial structure of the programs that will lately be evolved. Three standard methods are used to generate GP individuals: the grow, full and ramped half-and-half methods [82].

Grow Method: Select a random symbol with uniform probability from the set $\mathbb{F}$, that symbol will be the root node of the tree. Select then a number of symbols equal to the arity of the root node from the set $\mathbb{F} \cup \mathbb{T}$ sampled with uniform probability to be the children of the root node. Repeat this process recursively for each function node unless the node has depth equal to $d-1$ where d is the maximum tree depth allowed. In case the node has depth equal to $d-1$, the next symbol is taken only from the set of terminal ones. This procedure actually enforces the trees to be fully leafed before reaching the maximum depth. The distribution of trees shape, intended in terms of number of nodes and depth, of the initial population depends in a non trivial way on the ratio between the number of terminal elements and the number of functions of each arity.

Full Method: Works similarly to the *grow method*, but next symbols are sampled only from the $\mathbb{F}$ set until the $d-1$ depth is reached. Once the $d-1$ depth is reached, the final level is completed by sampling only from the $\mathbb{T}$ set. In this way each tree is forced to reach the full depth.

Ramped Half-and-Half Method: To increase the diversity of the initial population, the *ramped Half-and-Half* method divides the initial population in two halves: one half is initialized with the *grow* method and the other half with the *full* method.

3.4. Evolutionary Operators

The evolutionary process is driven by the satisfaction of a user-defined objective. This objective is formulated into a fitness function, used to evaluate the individuals. Each individual will then have an associated fitness value, representing how well an the individual can solve the target problem.
In the vast majority of cases, the fitness function is evaluated over a set of fitness cases that should be chosen in order to properly represent the envelope of situations in which the target program will need to operate. Once the fitness evaluation is completed, the individuals are ranked and a set of evolutionary operations are performed: mutation, crossover and selection [82].

3.4.1. Crossover

The crossover operator (schematized in 3.2a), also known as sexual recombination, creates variations in the population by producing children obtained by recombining the genetic material from two parents T_1 and T_2 chosen by means of selection methods. In the

standard procedure, a node (i.e. the crossover point) is chosen randomly by sampling a probability density function where higher uniform probability is attributed to internal nodes (e.g.: 0.9), and lower probability is given to the root node and to the leaves (e.g.: 0.1). Once the crossover point has been selected, the crossover fragment can be generated by extracting the sub-tree rooted at the crossover point. This process is done for both the parents T_1 and T_2 and for each of them, the selected crossover fragment is removed and replaced with the fragment taken from the other parent. This process can be done to return only one offspring or both of them. Since entire sub-trees are swapped and because of the closure property of the primitive set, this operation also produces syntactically legal programs [82].
Notice that when one crossover point is a leaf and the other one is the root node, the offspring will be much bigger and deeper than the parents: this could be and advisable condition in the very early stages of the evolution, but later could lead to the bloat effect described later. Some limit may therefore be imposed on the offspring size.
Also in this case, many variants of this operator exists, some of them are listed hereafter [58].

One-Point Crossover: The crossover point in both parent is forced to be in the same node of the tree. If the two parents have different shapes, the two trees are analyzed from the root node down to the leaves and only the nodes belonging to the common region (i.e. the region in which the parents have the same shape) are eligible to be crossover point.

Uniform Crossover: The offspring are created by visiting the nodes in the common region and at each locus, assign 50% probability to take that node from the first and 50% probability to take that node from the second parent. If the node to be inherited is at the base of the common region and is not terminal, then the sub-tree rooted there is inherited as well.

Context-Preserving Crossover: The crossover points are forced to have the same coordinates in the tree but points are not forced to belong to the common region.

Size-Fair Crossover: The first crossover point is selected randomly as in the standard crossover but differently from this, the size of the sub-tree to be removed from the first parent is calculated. The knowledge of the size of the removed sub-tree is then used to constrain the choice of the second crossover point to guarantee that the sub-tree removed from the second parent is not too big.

3.4.2. Mutation

The mutation (represented in Figure 3.2b), differently from the crossover, is an asexual operation involving therefore only one parental program. The mutation operation begins by choosing randomly with uniform probability, a mutation point in the selected individual. Once the mutation point has been selected, the sub-tree rooted at it is swapped with a randomly generated one. Analogously to the standard crossover, this operation accounts for a maximum allowed depth thereby limiting the size of the newly created tree [82]. As for the other two cases, different variants of the mutation operator exists and here some of them are reported [58]. It is interesting to note that oftentimes it is beneficial to use multiple types of them simultaneously.

Size-Fair Sub-tree Mutation: Replace a randomly selected sub-tree with another randomly created sub-tree as in the standard mutation operator, but the size of the new sub-tree is constrained to belongs to a certain range.

Point Mutation: A node in the tree is randomly chosen and randomly changed, to ensure the validity of the operation, the arity of the two nodes must be the same.

Swap Mutation: Select a random function node and then randomly permute its arguments (sub-trees).

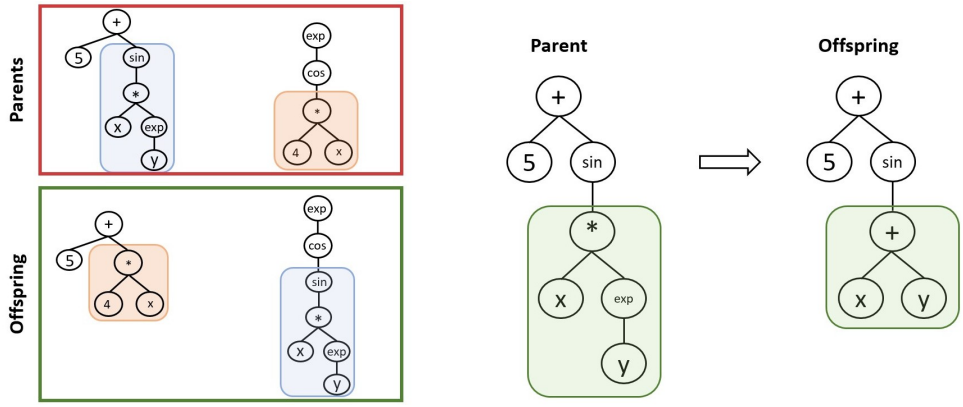
Hoist Mutation: The new offspring individual is a copy of a randoms selected sub-tree of the parent.

Shrink Mutation: Replace a randomly chosen sub-tree with a randomly chosen terminal node.

3.4.3. Selection

Once mutation and crossover have been performed on the current population, the selection operator selects which individuals will be copied into the next generation. Many selection algorithms have been developed during years, some of them are here reported [87].

Tournament Selection: a group of k individuals randomly selected from the population takes part in a tournament. The best individual usually is selected and inserted in the next generation. However some applications, particularly when the population starts



(a) Crossover (b) Mutation

Figure 3.2: Graphical representation of mutation and crossover operations

to converge and several individuals have the same fitness value, may feature a random selection. The process can be repeated n times with the remaining individuals.

Double Tournament: selects an individual using the standard tournament selection but differently from this, individual participating to the latter tournament are not randomly selected from the population but they are each the winner of a previous tournament selection [39].

Fitness Proportionate Selection: select programs to insert in the next generation according to their fitness value. In a population $\mathbb{P}$ with N programs, each program i has probability of being selected equal to $p_i = \frac{f_i}{\sum_{i=1}^{N} f_i}$.

Ranking Selection: sorts programs according to their fitness value, the rank N is assigned to the best program and the rank *1* to the worst one, then a function that can be both linear or not, is used to calculate the selection probability based on their rank.

Fitness Uniform Selection: generates a random number between the lowest and the highest fitness values of the current population, than the individual with the fitness value nearest to the random number is selected. In this way genetic diversity can be maintained also when the initial fitness distribution is not uniform.

3.5. Open Issues

The purpose of this section is to illustrate some open issues of practical relevance about the GP.

3.5.1. Bloat

Often during the GP evolutionary process, the average size of the programs (i.e. the number of nodes) tends to grow as the evolution progresses. However this increase in program size is generally not accompanied by any corresponding increase in performances. The above mentioned phenomenon is known as *bloat* [82]. The nodes in a GP tree can be divided into two classes: inactive code if their code is not executed or it has no effects, and active code it has direct effect on the fitness value. It has been observed that trees with smaller than average sub-trees tends to have less inactive code [76]. There exists multiple techniques to control the bloat effect such as using size and depth limits or the usage of a multi-objective selection technique to optimize contextually the fitness and minimize the program size.

3.5.2. Convergence

In the context of Evolutionary Algorithms (EA) acting on traditional fixed-length strings like the Genetic Algorithm (GA), it is possible to prove that if the algorithm has nonzero mutation rate, it is a global optimizer with guaranteed convergence given enough generations [82]. Even though multiple attempts have been made so far, it is fair to say that at the present time there is no formal mathematical proof that a GP system can always find solutions. In spite of this there exists arguments in support of the conjecture according to the fact that with minor modifications, the GP is guaranteed to visit a solution to the problem given enough generations. The most relevant argument supporting the present conjecture is that the mutation operator is able to generate with nonzero probability each element in the search space independently from the starting one. Given enough generations it is then possible to assess that the solution of the problem will eventually be visited. In general, maintaining a high population diversity has been proven to improve the performances of the GP [45].

3.6. Inclusive Genetic Programming

The IGP [45] is a GP variant developed to promote and maintain the population's diversity and with a set of features particularly useful for GC applications. These features are: 1) the possibility to consider constraints, and 2) the possibility to evolve multiple trees at the same time, making it particularly suited to solve control problems and in particular in designing Multi Input Multi Oputput (MIMO) controllers.

The diversity promotion and maintenance mechanism is based on the subdivision in niches

of the population at each generation. Individuals are associated to a certain niche according to their size (i.e. number of nodes) and the niches boundaries themselves are created dividing linearly the interval between the smallest program and the largest one in the current generation. The number of niches N is kept constant during the evolutionary process but their boundaries changes at every generation. Once the niches are created, both the recombination and selection are performed considering individuals from different niches to maintain the population diversity. To chosen figure of merit to assess the population diversity is the entropy, measured as follows:

$$S = -\sum_{i=1}^{N} (d_i \cdot log\,(d_i)) \tag{3.1}$$

where N is the number of niches and $d_i = \frac{n_i}{n_{tot}}$ being n_i the number of individuals in the *i-th* niche and n_{tot} the total number of individuals in the population. Notice that in Equation 3.1 empty niches are not considered.

The niches creation is embedded into the ES, leading to the Inclusive $(\mu + \lambda)$ strategy, which is a modified version of the standard $(\mu + \lambda)$ [45]. The Inclusive $(\mu + \lambda)$ pseudocode is reported in Algorithm 3.3.

Algorithm 3.3 Inclusive $(\mu + \lambda)$

```
1: Evaluate fitness of the individuals in the population
2: Update Hall of Fame of best individuals
3: while Generation < Maximum Generation Number do
4:     Create n niches according to the maximum and minimum length of the individuals
       in the population and allocate the individuals to their respective niche
5:     Perform Inclusive Reproduction to produce λ offspring
6:     Evaluate fitness of the individuals in the obtained offspring
7:     Create n niches according to the maximum and minimum length of the individuals
       considering both the parents and the offspring, and allocate the individuals to their
       respective niche
8:     Perform Inclusive Tournament Selection to select μ new parents
9: end while
```

The Inclusive Reproduction and Selection are two specific IGP features. The Inclusive Reproduction is composed by the *Inclusive Mutation* and *Inclusive Crossover* [43, 45]. These mechanisms are described hereafter.

Inclusive Mutation: Consist in applying the mutation operation to individuals selected from different niches. The individuals are randomly selected, and all niches are exploited during the offspring generation. The mutation is performed randomly selecting one of the following mechanisms Shrink mutation, Uniform mutation, Insertion mutation or ephemeral mutation, each of them is defined in the DEAP library [18].

Inclusive Tournament: It consists in performing a Double Tournament selection over all the created niches. Moreover, individuals that satisfies the applied constraints are favored [43].

Inclusive Crossover: Performs a one-point crossover between two individuals taken from different niches. The two individuals are chosen as follows: one is the best of the considered niche, this is done in order to transmit his genome to the next generations; the other one is instead selected randomly to maintain a degree of diversity in the population. An algorithm to avoid breeding between the same or very similar individuals is used [43].

The IGP proved to be capable of maintaining higher levels of entropy (i.e. population diversity) during the evolutionary process, with respect to the Standard Genetic Programming (SGP). This was assessed by comparing the IGP and the SGP on several benchmarks [45].

3.7. Summary

GP seems to be a valid option to derive the reentry guidance law proving its robustness when uncertainties are applied to the model or to the environment. Moreover the symbolic formulation of the guidance law is very promising since it results in a white box model on which it is potentially possible to impose constraint at training level, in contrast with DRL which produces a black box model.
Some drawbacks of the present approach include the very high computational effort required to reach the convergence in the evolutionary process and the choice of the functions and terminals sets.
The chosen GP formulation is the IGP [45] which has been specifically developed to address the guidance problem in aerospace domain [44]. Moreover the IGP has already been successfully used to solve a sub-case of the problem analyzed in this dissertation [46] confirming its reliability.

4 | ReFEx: Mission and Modeling

The DLR mission ReFEx has been chosen as reference case for the present study because it is currently in its phase C of development and because a lot of high quality data, including structural information, aerodynamic characterization and the 6 DOF simulator used to verify the performance of the GNC algorithms are available.
This chapter aims to provide the details about the ReFEx mission and about the way it is simulated; it is structured as follows: section 4.1 outlines the architecture and the main features of the ReFEx vehicle, in section 4.2 the reference mission profile is provided with the distinction in its main phases. Finally, section 4.3 and section 4.4 describe how the flight simulations are performed within this study framework. Particular emphasis is posed on the latter two chapters because the simulation aspect is central with respect to the results that will be presented throughout this dissertation. For more general information on ReFEx, see [66]. For a visual simulation of the mission, see [15].

One of the main difficulties connected to this project is the intrinsically high computational cost related to performing the trainings typical of the ML techniques. To address this issue, two different simulators have been used.

The most comprehensive and detailed simulator is the one described in section 4.4: this is a full 6DOF simulator developed in Simulink wherein the whole mission is modeled with a very high degree of detail. Both the Hybrid Navigation System (HNS) and the Guidance and Control Computer (GCC) are here modeled and implemented, coupled with the most detailed environmental modeling available. Unfortunately the computational cost to evaluate the output of this system is substantial. For instance, a single run performed on a Intel(R) Xeon(R) E-2186G CPU @ 3.80GHz takes approximately 20 minutes making its usage unfeasible for training purposes in which it is required to perform hundreds of thousands of trajectory propagations. However, the availability of this tool allows for the most reliable evaluation of the guidance performances on a real system therefore it has been used during the final testing phase in which the guidance policies derived through RL and GP on the 3DOF simulator, have been evaluated through the Monte Carlo approach and compared with the baseline guidance solution currently used and implemented by

DLR.

As aforementioned, in order to make the training feasible, a 3-DOF simulator has been used to design the guidance policies with both techniques and its details will be disclosed in section 4.3. The objective of this software is to have the most similar 3DOF environment to the 6DOF one that will be used as testing setup. This environment is considered reliable since it is the same used to design the real mission guidance strategy even if it has been necessary to perform some adaptations in order to make it suitable for the specific usage requested for the present study.

The full development pipeline, which is valid for both cases (i.e. RL and GP) consists in first performing the training on the 3DOF simplified environment, once the training is complete, the guidance policy is evaluated on a validation dataset again on the 3DOF environment and if the policy can accomplish acceptable performances levels, it is then tested on the 6DOF simulator to compute the real performances.

Note: almost all the simulation software were implemented by DLR and have been used for this study "out of the box". This chapter therefore aims to explain how they work in order to offer a more detailed view of the characteristics of the study itself. Nevertheless, sometimes it has been necessary to implement some additions or modifications to the DLR software in order to develop some features specific for the project in exam. When one of these cases occurs, it will be explicitly specified.

4.1. Vehicle

ReFEx is a winged RLV designed according to the Vertical Take-off Horizontal Landing (VTHL) paradigm. It is a DLR demonstrator mission to develop future winged vehicles with reentry capabilities, in particular, it is designed to follow a realistic trajectory for a general winged RLV going through a wide mach regime, from hypersonic to subsonic, while performing an autonomously guided and controlled flight. The reentry segment, namely ReFEx will be carried to a suborbital trajectory through the *VSB-30* sounding rocket. Since the launcher has no active vector control capabilities, it is passively stabilized. Moreover a spin is induced to reduce its dispersion at the Separation Event. Hence, the payload (i.e. ReFEx) is required to have an almost symmetrical shape to enable a safe launch. On the other hand, the Re-Entry Segment must have an aerodynamic shape to allow for a successful Experimental Phase. In order to meet both the requirements, ReFEx is stored on the top of the launcher with its wings folded within a fairing.
ReFEx has a length of 2.7 m, a wingspan of 1.1 m and weight $\approx 400\ kg$.
The diagonal moment of inertia are $\approx 15\ kgm^2$ for the longitudinal axis and $\approx 240\ kgm^2$

for the other two axis.
The aerodynamic design consists of two fixed foldable wings located at the back of the vehicle, a vertical rudder and two canards close to the nose.
The set of control actuators is completed by eight cold gas Reaction Control System (RCS) thrusters located at the back of the vehicle that are used for attitude control [4].

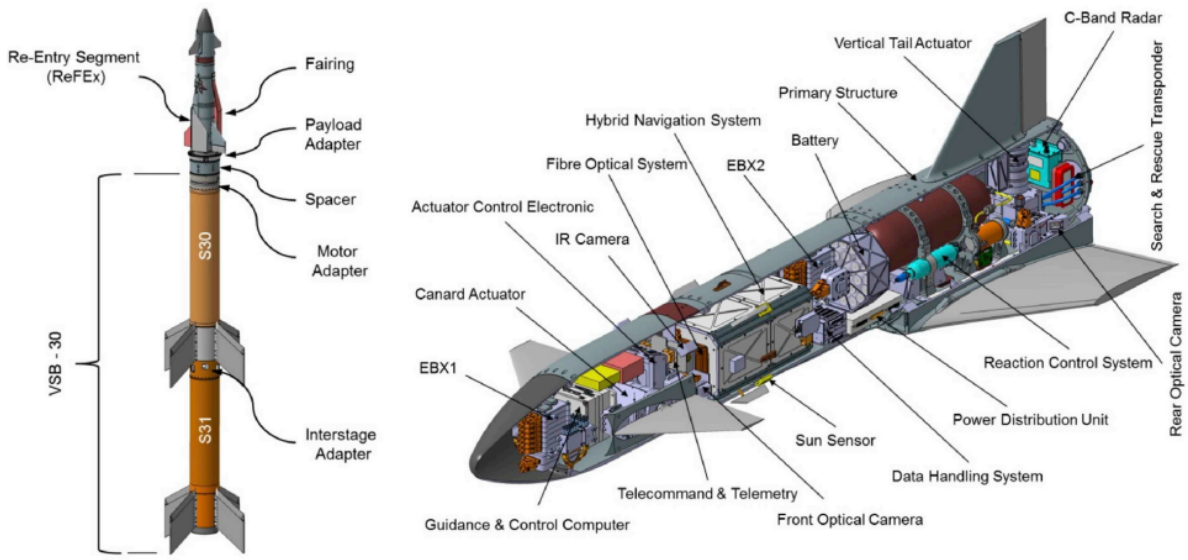


Figure 4.1: ReFEx launch Configuration (left), section view of the Re-Entry segment ReFEx (right) [4]

4.2. Reference Mission

The ReFEx nominal mission profile is divided in four main phases [4]:

1. Launch phase.
2. Experimental phase until Entry Interface (EI).
3. Experimental phase between EI and End of Experiment (EoE).
4. Experimental phase after EoE.

A representation of the mission is shown in Figure 4.2. For a more detailed description of the trajectory, see [9].

Launch Phase ReFEx is launched through the Brazilian bi-stage sounding rocket *VSB-30* [22] from the Woomera test range in Australia. The first stage burns out and separates 12 s after ignition. About 20 s after launch, the second stage of the *VSB-30* is ignited and it burns out ≈ 29 s after. Since the *VSB-30* is just passively stabilized, no ascent control

on the terminal conditions is available therefore, in order to reduce the apogee dispersion, the rocket is also spin stabilized to compensate thrust inaccuracies. The angular velocity is then reduced using a Yo-Yo de-spin system that will be deployed at $t \approx 80\ s$. The fairing separation, and separation from the launcher occurs shortly after, followed by the deployment of the foldable wings. At this point, called *Separation Point*, the *Experimental Phase* and also the controlled part of the flight begins. The maximum achievable Mach number during the Launch Phase is ≈ 5.7.

Experimental Phase until EI In this phase ReFEx is separated from the launcher; the apogee is reached at an altitude of $\approx 130\ km$.During this phase, the vehicle follows a ballistic trajectory since the dynamic pressure is not enough to allow for aerodynamic control of the vehicle. The control during this phase occurs via the RCS thrusters which cannot provide longitudinal control but are used to de-tumble the vehicle, provide a roll maneuver to acquire the sun signal and enable the navigation solution to converge and reach the EI with the correct attitude [62]. Since this part of trajectory is ballistic, the uncertainty in the state of ReFEx at the EI increases considerably. The EI is defined on a dynamic pressure threshold that guarantees that the vehicle attitude is aerodynamically controllable.

Experimental Phase between EI and EoE ReFEx is expected to reach the EI at an altitude of $\approx 60\ km$, $\approx 4\ min$ after the beginning of the mission. The EI is defined to allow the aerodynamic surfaces of ReFEx to have enough control authority. This phase is critical from the control viewpoint because the Re-Entry segment will face very different flight conditions evolving quickly, moreover the aerodynamic static stability properties change dramatically with the mach number. The vehicle does not have sufficient natural longitudinal and lateral static stability in belly-down orientation for Mach number ≥ 2. In order to solve this challenge, ReFEx will fly the initial part of the reentry trajectory in belly-up configuration, once the Mach number is sufficiently low, it will perform a roll maneuver bringing it to the classical belly-down attitude. This maneuver is necessary since otherwise the effectiveness of the canard will decrease, and the spacecraft would become again unstable and not controllable. The trajectory will therefore be computed to keep the Re-Entry Segment within flight corridor defined in terms of angle-of-attack and Mach number.
A further critical maneuver performed during this phase is a 180° turn, bringing the heading angle from $\approx 0°$ to $\approx 180°$ proving the capabilities of the GC subsystem to reorient the vehicle. The purpose of the guidance algorithm in this phase is to bring ReFEx to the target point to be reached at the EoE that is defined in position and

velocity at an altitude of approximately 6 km.

Experimental Phase after EoE Once the EoE condition has been reached, the final part of the mission consists in reducing the kinetic energy of the vehicle while avoiding landing zones that are near to a lake. A lake avoidance maneuver is prescribed in case the current trajectory lands too near to a lake, while the kinetic energy is planned to be dissipated with a final flare maneuver [65].

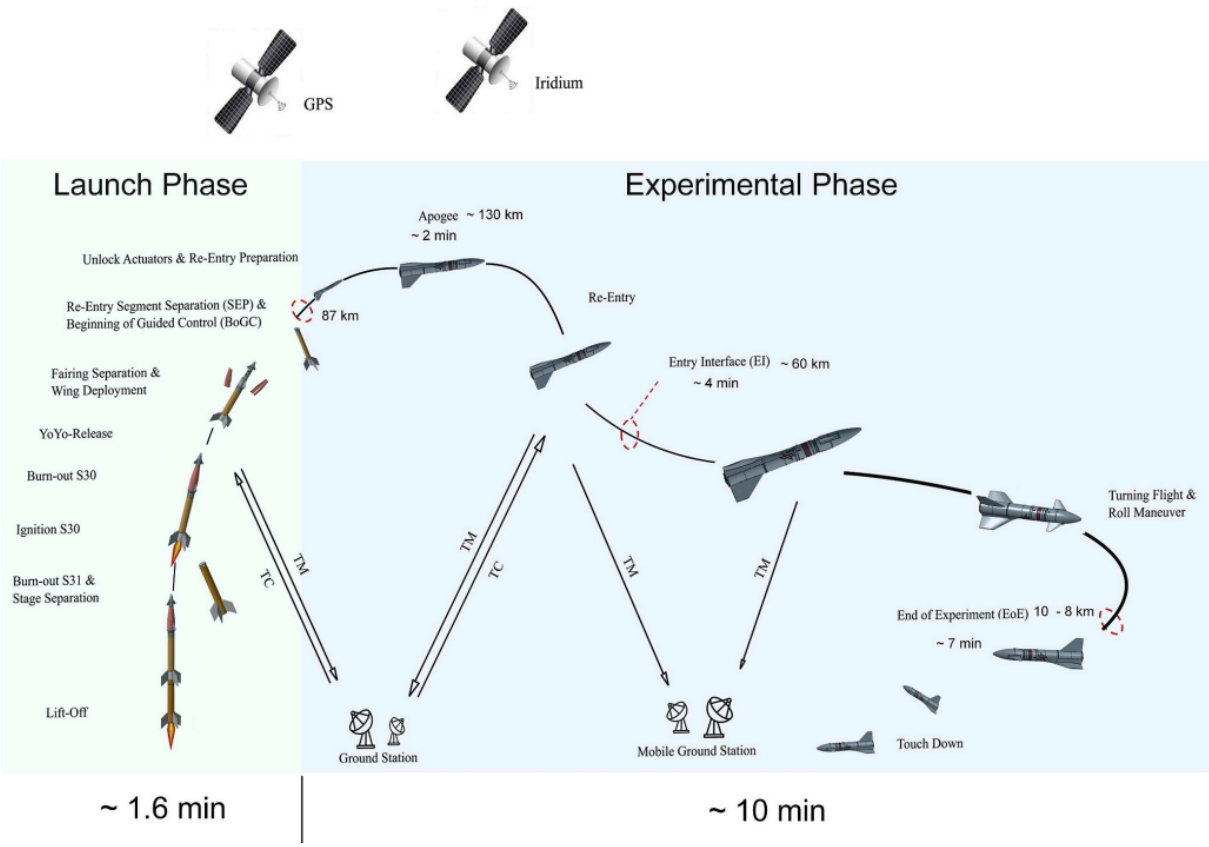


Figure 4.2: Mission architecture and flight events [4]

4.3. Training Simulator

The objective of this section to show a detailed overview of the training simulator in its different components, in Figure 4.3 it is shown the full structure of the software. This model is is based on the one used in [64].

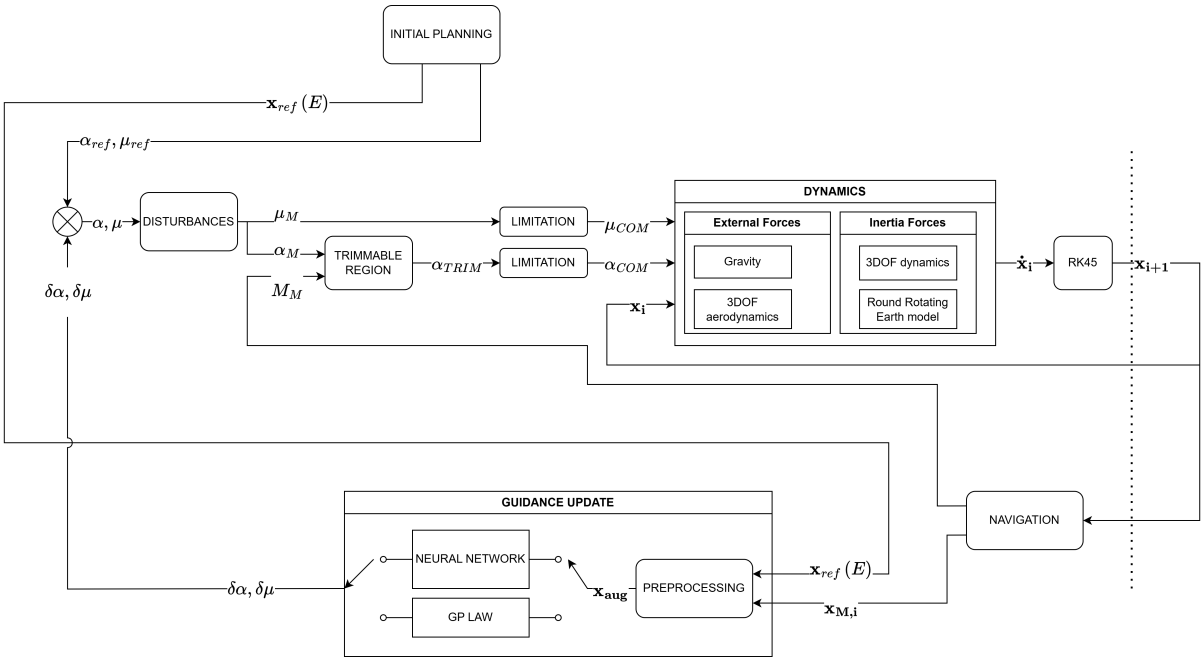


Figure 4.3: Training Simulator Architecture

4.3.1. Vehicle and Environmental model

Both the vehicle and environment models are formulated to incorporate their respective uncertainties during propagation. The uncertainties are sampled from uniform or normal distributions, depending on the nature of the uncertainty itself. Normal distributions can be correlated or uncorrelated, depending in the cross-effect between different parameter uncertainties.

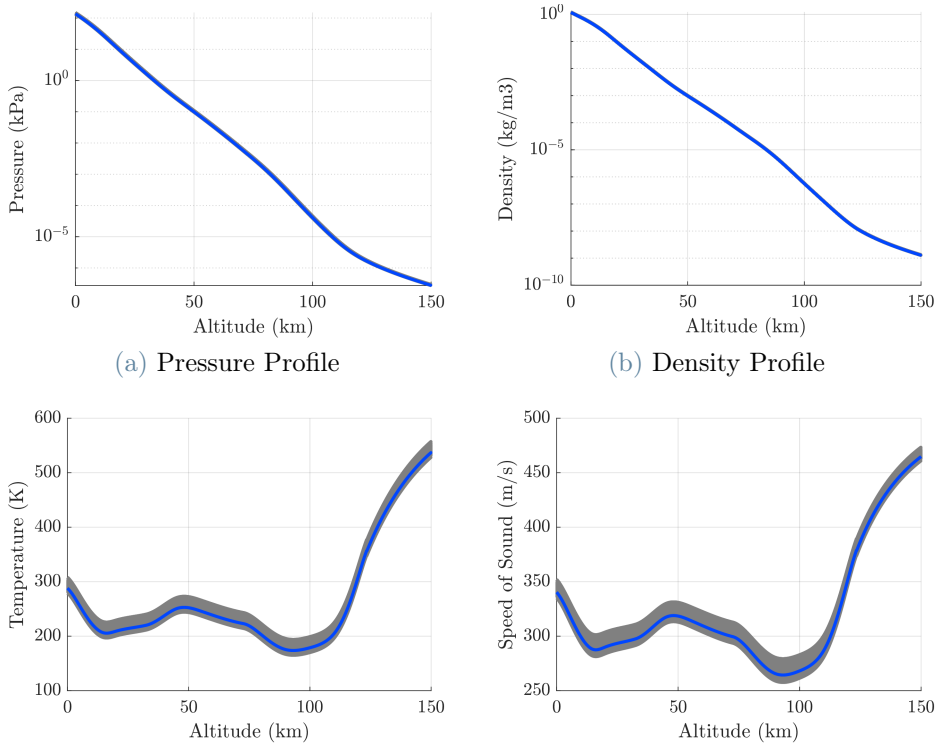
The environmental model is divided in two main sections: atmospheric and gravity model. In the same way, the vehicle model contains the aerodynamics and the mass. Each of them is perturbed as above mentioned except for the gravity one that is considered deterministic.

Gravity model

It accounts for the harmonic components up the the second order, therefore only the J2 perturbation is accounted in the model itself. The gravity acceleration computed in the Earth Centered Earth Fixed (ECEF) reference frame is indicated by $\mathbf{g}_{ECEF}$.

Atmospheric model

Atmosphere is characterized according to the NRLMSISE-00 empirical reference model [57]. It provides information on density, temperature, pressure and speed of sound of the



(a) Pressure Profile (b) Density Profile

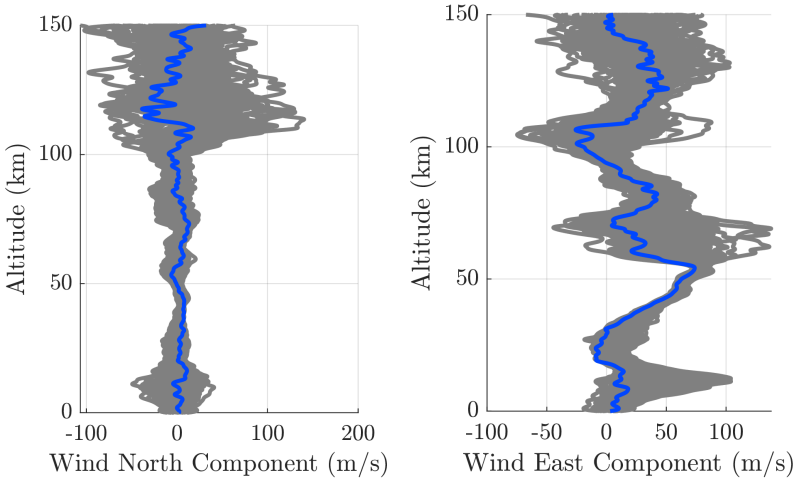
(c) Temperature Profile (d) Speed of Sound Profile

Figure 4.4: Atmosphere characterization - the blue line represents an illustrative instance

atmosphere from the ground level up to the end of the thermosphere (i.e. $\approx 600\ km$). In Figure 4.4 a set of 100 scenarios is reported, one of them is enlightened in blue to provide a more intuitive representation.

The wind on the other hand, is modeled according to the Horizontal Wind Model (HWM)[16] which provides the velocity components on the North-East-Down (NED) reference frame based on a statistical database as function of the location on the Earth and day of the year. In each case, the vertical wind component is set to zero.
Even if at high altitudes the effect of the wind on the trajectory is negligible, under 30 km the impact becomes considerable. In Figure 4.5, 100 samples of possible wind profiles are reported as reference, one of them is highlighted in blue as done before.

In general, the atmospheric models here described might be computationally expensive to be evaluated within a training scenario. In order to solve this issue, the atmospheric data are precomputed to generate a set of realistic scenarios. Before running any simulation, the proper dataset is loaded and the actual atmospheric parameters are computed during the simulation by interpolating from the corresponding lookup table.



(a) Wind North component (b) Wind East component

Figure 4.5: Wind components - in blue an illustrative instance

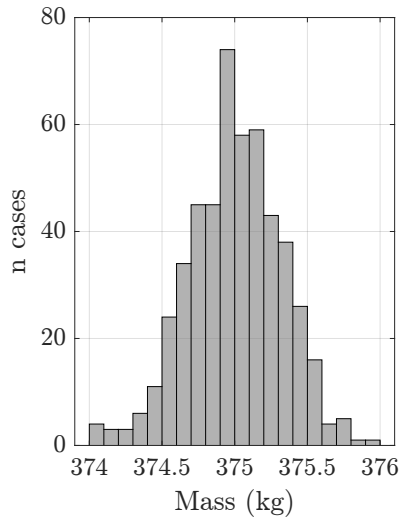


Figure 4.6: ReFEx mass distribution

Mass Uncertainty

The vehicle mass is considered uncertain and its value in each run is sampled by the distribution shown in Figure 4.6. The nominal value is 375 kg.

Aerodynamics model

The aerodynamic coefficients necessary to perform the numerical simulation are available in the form of aerodynamic databases computed via CFD internally in DLR [50, 51]. In particular, in order to perform the 3DOF simulations, only the Cx and Cz are necessary since the flight is considered to be always symmetric (i.e. with null sideslip angle β). The coefficients are tabulated as function of the angle of attack α and of the Mach number M and are computed via linear interpolation from the corresponding lookup table. In a more detailed analysis, the aerodynamic coefficients will be dependent on the deflections of the control surfaces i.e. canard and rudder which, on the other hand, will be function of the control output. This effect is properly modeled in the testing simulator as will be detailed in section 4.4 but it cannot be addressed in the framework of the material-point 3DOF dynamics considered in the present framework. In order to account for this effect, the coefficients that are actually considered during the propagation are the ones corresponding to the trimmed flight condition.

A flight condition is considered trimmed when the control surfaces are deflected such that the total aerodynamic torque acting on the vehicle is null. The aerodynamic forces acting

on the vehicle, computed on the body reference system, are then the following:

$$
\begin{aligned}
F_X &= \frac{1}{2}\rho\left(h\right)\ v^2 S_{ref} C_x\left(\alpha, M\right) \\
F_Y &= 0 \\
F_Z &= \frac{1}{2}\rho\left(h\right)\ v^2 S_{ref} C_z\left(\alpha, M\right)
\end{aligned}
\tag{4.1}
$$

4.3.2. Equations of motion

The vehicle dynamics are those developed in [64], and will be explained below based on the content of that paper. They are computed with respect to the state vector $\mathbf{x} = [r, \lambda, \phi, v, \gamma, \chi]$ where:

- r is the distance from the center of the Earth;
- λ is the longitude;
- ϕ is the latitude;
- v is the norm of the velocity of the center of mass of the vehicle;
- γ is the flight path angle;
- χ is the heading angle.

Notice that each variable is expressed in the Geocentric (GEOC) reference frame. The variables v, γ and χ represent the velocity computed with respect to the ground in the ECEF reference frame; a detailed description of the reference frames used in the study is reported in Appendix B.

$$
\begin{aligned}
\dot{r} &= v\ sin\gamma \\
\dot{\lambda} &= \frac{v\ cos\gamma\ sin\chi}{r\ cos\phi} \\
\dot{\phi} &= \frac{v\ cos\gamma\ cos\chi}{r} \\
\dot{v} &= f_v + \omega^2\ r\ cos\phi\ \left(sin\gamma\ cos\gamma\ - cos\gamma\ sin\phi\ cos\chi\right) \\
\dot{\gamma} &= f_\gamma + \frac{v}{r}\ cos\gamma 2\omega\ cos\phi\ sin\chi \\
&\quad +\ \omega^2\ \frac{r}{v}\ cos\phi\ \left(cos\phi\ cos\gamma - sin\gamma\ sin\phi\ cos\chi\right) \\
\dot{\chi} &= f_\chi + \frac{v}{r}\ cos\gamma\ tan\phi\ sin\chi - 2\omega\ \left(tan\gamma\ cos\phi\ cos\chi - sin\phi\right) \\
&\quad +\ \omega^2\ \frac{r}{v\ cos\gamma}\ cos\phi\ sin\phi\ sin\chi
\end{aligned}
\tag{4.2}
$$

The time derivative of the state vector computation is reported in Equation 4.2. In Equation 4.2, ω is the Earth's angular velocity, f_v, f_γ and f_χ represent the external forces projected on the corresponding components of the state vector divided by the vehicle mass. The two external contributions on the problem are gravity and aerodynamic forces. The gravity acceleration is computed in the ECEF reference frame. In order to include it in Equation 4.2, the acceleration vector is first rotated in the NED reference frame as shown in Equation 4.3 using the rotation matrix in Equation B.3.

$$\mathbf{g}_{NED} = R_{ECEF \rightarrow NED} \cdot \mathbf{g}_{ECEF} \tag{4.3}$$

The gravitational acceleration output of Equation 4.3 is expressed in cartesian coordinates, but to be integrated in Equation 4.2 needs to be transformed in the spherical coordinates system (v, γ, χ). This is done using Equation B.2, in which the vector $\mathbf{x}$ is represented by the vehicle velocity in the NED reference frame, while $\mathbf{dx}$ is instead represented by the gravitational acceleration computed in Equation 4.3. This process leads to the expression of the gravitational acceleration in the GEOC reference frame expressed in Equation 4.4.

$$\mathbf{f}_{g,GEOC} = \begin{bmatrix} f_{g,v} \\ f_{g,\gamma} \\ f_{g,\chi} \end{bmatrix} \tag{4.4}$$

The aerodynamic forces computed in the body axes with Equation 4.1 are first rotated in the framework aligned with the local atmosphere velocity obtaining the Lift (L) and Drag (D) components as:

$$\begin{cases} L = F_x \ sin\alpha + F_z \ cos\alpha \\ D = -F_x \ cos\alpha + F_z \ sin\alpha \end{cases} \tag{4.5}$$

The aerodynamic forces are now transported in the spherical reference but still expressed with respect to the wind frame as follows:

$$\begin{cases} f_{a,v_W} = -\dfrac{D}{m} \\ f_{a,\gamma_W} = \dfrac{1}{m} \cdot \dfrac{L \ cos\mu}{v_{wind}} \\ f_{a,\chi_W} = \dfrac{1}{m} \cdot \dfrac{L \ sin\mu}{v_W \ cos\gamma_W} \end{cases} \tag{4.6}$$

The vector of accelerations due to the aerodynamic forces computed in Equation 4.6 is then transported in the NED reference frame using the inverse rotation of Equation B.3

and then after exploiting the fact that the atmospheric wind is assumed to be slowly varying, the obtained vector can again be transformed in the spherical reference frame using Equation B.2 as already done for the gravity vector. The full sequence of transformations is the following:

$$\begin{bmatrix} f_{a,v_W} \\ f_{a,\gamma_W} \\ f_{a,\chi_W} \end{bmatrix}_W \rightarrow \begin{bmatrix} f_{a,N} \\ f_{a,E} \\ f_{a,D} \end{bmatrix}_W \approx \begin{bmatrix} f_{a,N} \\ f_{a,E} \\ f_{a,D} \end{bmatrix}_G \rightarrow \begin{bmatrix} f_{a,v} \\ f_{a,\gamma} \\ f_{a,\chi} \end{bmatrix}_G \tag{4.7}$$

where the suffix W refers to the fact that the components are expressed in relation to the wind, while G is referred to the ground.

To conclude, the terms f_v, f_γ and f_χ in Equation 4.2 are computed as:

$$\begin{bmatrix} f_v \\ f_\gamma \\ f_\chi \end{bmatrix} = \begin{bmatrix} f_{a,v} \\ f_{a,\gamma} \\ f_{a,\chi} \end{bmatrix} + \begin{bmatrix} f_{g,v} \\ f_{g,\gamma} \\ f_{g,\chi} \end{bmatrix} \tag{4.8}$$

Finally, the equations of motion in Equation 4.2 are integrated using an adaptive Runge-Kutta method, with methods of order 4 and 5 [64].

4.3.3. Guidance, Navigation and Control modeling

In order to increase the degree of accuracy of the training simulator, three modules aiming to represent the effects due to the Navigation, Guidance and Control subsystems presence, which are on the other hand fully modeled in the Testing Simulator, are included and here described.

Navigation

The implementation of a way of simulating the navigation errors in the 3DOF simulator is one of the original modification implemented during the development of the project with respect to the original DLR implementation. A performance navigation model (i.e. aiming to simulate the effect of the sensors without actually simulating their internal dynamics) was only present in the 6DOF testing simulator and it has been adapted to the training one.

The objective is to model the effect of the navigation errors in providing a state vector to the GCC which differs from the real one. Since in this context modeling all the different sensors would not be possible and in general not even useful, the performance model developed by DLR and used during the development of ReFEx has been adapted and

implemented on the training simulator.

For this purpose, multiple fictitious sensors, each of them modeling the noise as an autoregressive random process, have been defined: position in the ECEF frame, velocity in the ECEF frame, Mach number, Geodetic Altitude, Angle of Attack and Angle of Bank. Each of them is assigned to generate a perturbation in the real value of the proper parameter (in this context *real* refers to the output of the propagation) to be submitted to the guidance and control logic in order to account for the navigation errors that will also be encountered on the actual system.

Validation

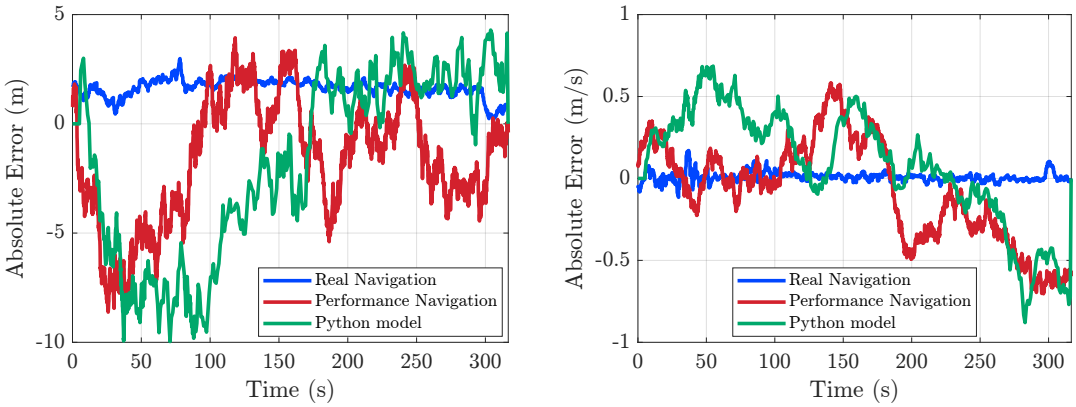
The performance model to be used during the training has been validated with respect to the Simulink implementation used in the Testing Simulator. In particular, the Testing Simulator, allows to use two different models to simulate the HNS: the performance model to which the current implementation refers to, and the real model which is used during the final testing simulations as will be detailed later. Both the implementations have been considered to create the comparison used to perform the validation.
The comparative analysis has been performed only considering the output of the position and velocity sensors. This choice has been made because other quantities (i.e. mach, angle of attack, bank angle) go through different acquisition processes when comparing 3DOF and 6DOF simulations therefore it was impossible to have a completely fair comparison.

Due to the stochastic nature of the disturbances, having a 1:1 comparison was also difficult due to the different Random Number Generator (RNG) algorithms used by Python and Simulink (and the issues in modifying the Simulink implementation for this test), therefore a qualitative match in terms of trend and order of magnitude was considered sufficient to validate the noise modeling.
Figure 4.7 report a test run to show the performances of the three aforementioned methods. As can be seen, the python model is consistent with respect to the performance navigation model while they both provide higher noise values with respect to the real navigation model. Higher noises have been accepted in the 3DOF environment in order to account of the modeling differences with respect to the 6DOF.

4.3.4. Guidance

The ReFEx baseline guidance strategy is divided in two main phases:

1. Initial planning: performed during the ballistic flight phase, in which the reference guidance signal is computed by solving a reduced optimal control problem [64];



(a) Position error in ECEF reference frame (b) Velocity error in ECEF reference frame

Figure 4.7: Navigation model validation

2. Periodic update: performed during the actual reentry phase, in which the reference signal is updated every $2s$ by solving a single optimization step of a problem similar to the one solved during the initial planning [65].

A more detailed description of the actual guidance algorithm is reported in Appendix A. Within this study, the output of the initial planning is computed before the beginning of the simulation and it is considered as reference, while the RL and GP based guidance laws play a role in performing the periodic update instead of the optimization-based algorithm originally implemented.
Since the training simulator is never used to perform a full flight simulation with the baseline guidance strategy, the real algorithm is not implemented in it but an equivalent version suitable for the simulation with GP and RL is used as described in Algorithm 4.1, showing the conceptual functioning of the guidance subsystem only; the integration of the NN and GP-law in the 3DOF simulator is also a modification with respect to the original DLR implementation of the software. The control limitation logic will be explained in the next paragraph while for the initial planning algorithm details, refer to Appendix A.

Control

The guidance algorithm (i.e. the Neural Network or the Genetic Programming Law) output is the setpoint expressed in terms of angle of attack and bank angle. This is not enforced instantly during the propagation, but there exists a model of the control logic with the objective of applying to the propagator a feasible set of angle of attack and bank angle in such way that once passing to the testing environment, the propagation evaluated on the 3DOF is as much realistic as possible. In general, the propagation

Algorithm 4.1 Guidance strategy on training simulator

```
Select the uncertainty case (see section 5.2)
Load the precomputed corresponding reference trajectory
Start the flight simulation
while E > E_target do
    Compute the state of the vehicle in the GEOC reference frame
    Compute the reference guidance command α_R and μ_R
    if GP guidance selected then
        u_α, u_μ ← Evaluate GP guidance law
    else if RL guidance selected then
        u_α, u_μ ← Evaluate NN output
    end if
    α_guid ← α_R + u_α {Compute the Angle of Attack command by correcting the initial reference}
    μ_guid ← μ_R + u_μ {Compute the Bank Angle command by correcting the initial reference}
    Add disturbances to α_guid and μ_guid
    α_C, μ_C ← Apply control limitations to α_guid and μ_guid
    Perform integration step
end while
End Flight Simulation
```

function implements a full logic aiming to filter the *raw* guidance command in order to generate an actual command that is feasible for the real system [64].

The first constraint imposed consists in maintaining the angle of attack within a predefined *flyable* region. This region is defined in the $\alpha - Mach$ space and it is computed according to the *CFD* databases to avoid regions in which the vehicle is aerodynamically underactuated. The commanded angle of attack is therefore saturated according to this corridor between values computed as function of the Mach number. Further details about the flyable region will be provided in the next paragraph.

After this, the constraints on angular rates and angular accelerations are enforced directly within the propagation routine and therefore they do not need to be applied *externally*. Limits on angular rates and accelerations are imposed individually for angle of attack and bank angle at each propagation step by computing the limiting values in the accelerations as follows:

$$\begin{cases} \ddot{\theta}_{upper} = \min\left(\ddot{\theta}_{max}\ , \dfrac{\dot{\theta}_{max} - \dot{\theta}_{prev}}{dt}\right) \\ \ddot{\theta}_{lower} = \max\left(-\ddot{\theta}_{max}\ , -\dfrac{\dot{\theta}_{max} + \dot{\theta}_{prev}}{dt}\right) \end{cases} \tag{4.9}$$

notice that the variable θ in Equation 4.9 refers either to α or μ since the same approach

is applied to both. The commanded angular acceleration is then computed, if needed it is saturated according to Equation 4.9 and the actual guidance command is computed backward from the accepted value of acceleration.

The set of constraints enforced (in order) is reported in Equation 4.10

$$\begin{aligned} \alpha\left(M\left(t\right)\right)_{min} \leq \alpha\left(t\right) \leq \alpha\left(M\left(t\right)\right)_{max} \\ \ddot{\alpha}_{min} \leq \ddot{\alpha}\left(t\right) \leq \ddot{\alpha}_{max} \\ \ddot{\mu}_{min} \leq \ddot{\mu}\left(t\right) \leq \ddot{\mu}_{max} \end{aligned} \tag{4.10}$$

Flight Envelope

To account within the 3DOF environment for the 6DOF aerodynamics effects, in particular concerning the stability of the vehicle, the trimmability analysis reported in Figure 4.8 has been performed, repeating the one shown in [61]. The parameter considered in the colormap is the so called *trimmability value* which is an index of the range in the sideslip angle β for which the vehicle can be trimmed. A mesh of flight conditions defined with respect to mach number M, angle of attack α and sideslip angle β has been considered and, for each couple (α, M), the aerodynamic coefficients at different values of β have been analyzed. If the flight condition results to be trimmable it is assigned a $+1$ value, 0 otherwise. By summing up the results associated to each different β value, it is possible to obtain the map in Figure 4.8, in which warm colors refer to flight conditions easy to be trimmed (i.e. with several sideslip angles β in which a trimmed configuration of the control surfaces exists) while cold colors refer to flight conditions in which the vehicle is underactuated and therefore likely to be unstable.

It is important to notice that even if the map appears to be full, not all the coefficients are actually completely reliable from a simulation point of view. In particular, the original database computed via Computational Fluid Dynamics (CFD) simulation contains only the flight conditions within the black envelope, data outside it are based on coefficients computed via extrapolation with respect to the aforementioned data therefore results slightly outside this region are still considered acceptable while results far from it should be considered unreliable. Based on this, the flight corridor described by the magenta lines has been designed. This represents the stability corridor already cited in the control limitation section ad it aims to avoid the regions of high instability risk. As can be noticed, there are two regions in which the flight corridor goes out of the one related with the availability of aerodynamic data. This is a conscious choice by the designers in DLR who consider data in those regions still reliable. Moreover even if in the $\alpha - Mach$ space they seem to be very wide, when it comes to the time domain, the ReFEx vehicle spends a

very short time within them.

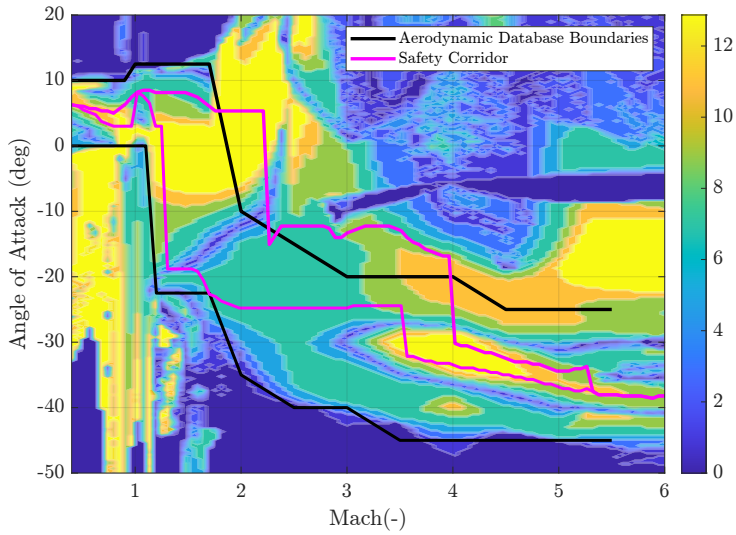


Figure 4.8: Trimmability of ReFEx with respect to α and M

4.4. Testing Simulator

The objective of this section is to describe the *Testing Simulator* (see [26]). This is a Simulink implementation of the full 6DOF dynamics of ReFEx including a detailed simulation of the vehicle itself, of the environment and of the GNC software. The simulator is divided in three main blocks: HNS GCC and Dynamics and Kinematics Environment (DKE). Figure 4.9 shows a simplified representation of the architecture of this simulator.

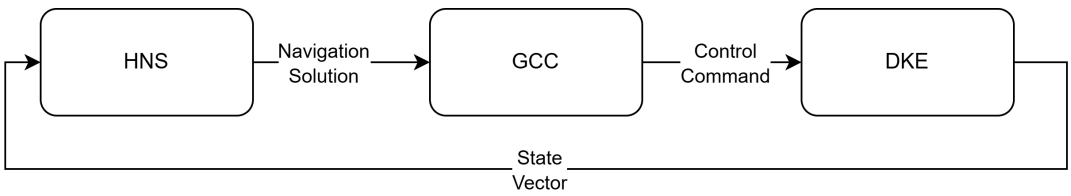


Figure 4.9: Simplified functional architecture

In the following sections, each of the three most important blocks will be detailed in order to clarify the biggest differences with respect to the simulator used during the training.

It is fundamental to notice that this simulator is internally used by DLR to certify and qualify the GNC algorithms applied to the ReFEx mission.
The outcome of this study therefore, has been tested on the most realistic possible environment and in general, the guidance laws in terms of performances, are tested in the same conditions used to verify the performance of the one which are actually implemented.

4.4.1. Dynamic and Kinematic Environment

The Dynamic and Kinematic Environment is the name that refers to the model plant. It contains a full 6DOF simulation of the vehicle dynamics which is divided in three main parts: the environment, the vehicle and the actuators model.

Environment: it is modeled accounting for atmospheric, magnetic and gravitational field contributions. In general the atmospheric model is the same used in the training simulator.
Also the gravity is modeled in the same way with respect to the 3DOF but since in this case we are considering an extended body with inertia properties, the torque due the gravitational force is also included in the simulation.

Vehicle:
The simulator features a very detailed physical model of the vehicle including:

- Complete 6DOF dynamics as already mentioned;
- Time-varying mass properties: this is due to the fact that during the initial stages of the reentry, the vehicle attitude can only be controlled via RCS therefore the effect on the mass and mass distribution (center of mass and inertia matrix) of the fuel depletion are accounted for. Moreover, also the variation in the position of the center of mass due to the thermal bending that occurs as a consequence of the aerodynamic heating developed during the reentry is considered.
- The aerodynamic model is an extension of the one used in the 3DOF simulation since now the full set of coefficients to propagate the 6DOF dynamics is necessary. In particular there are 6 coefficients (Cx, Cy, Cz, Cl, Cm and Cn) for each of: main body, wings, canards and rudder in order to account for the different effects and the deflections of each of them. As aforementioned, the aerodynamic data are produced through CFD simulations.

All the forces and torques deriving from different contributions are summed up to compute total force and torque acting on the vehicle and finally are passed to the routine assigned to compute the state derivative to be propagated.

Actuators:
The actuator dynamics is also considered during the simulation. In particular, two different types of actuators are considered: the aerodynamics surfaces and the RCS thrusters

each of them modeled as an independent body with its own internal dynamics and uncertainties. In Figure 4.10 it is reported a conceptual flowchart of the components within the DKE model.

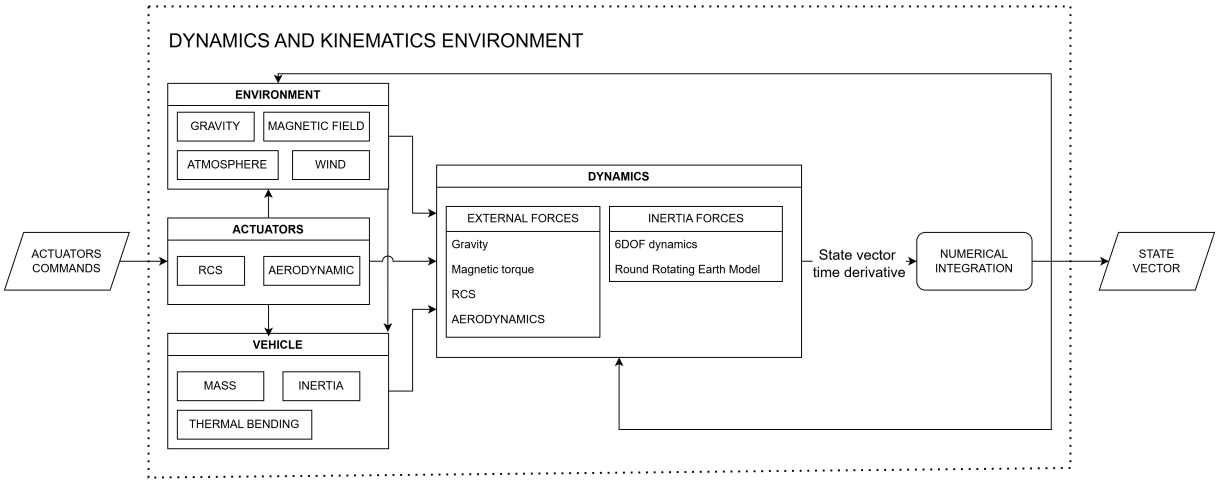


Figure 4.10: DKE conceptual architecture

4.4.2. Hybrid Navigation System

The objective of the HNS is to implement the navigation functions for the ReFEx mission by fusing data from inertial and non inertial sensors. The two most important functions of the HNS are:

1. Providing the Navigation Solution to the GCC.
2. Mission Vehicle Management (MVM): switches between different operative modes.

A comprehensive description of the system is out of the scope of this thesis, but more detailed information about the navigation software can be found in [62] and [14], while the sensor suite is described in [73].

4.4.3. Guidance and Control Computer

The last section of this chapter focuses on the GCC simulation. As can be deduced from the title, the software itself is divided into two main parts: the *Guidance* block and the *Control* block [26]. The Guidance section is the only section of the 6DOF simulator that has been modified with respect to the original in order to allow the generation of the guidance signal both with the NN and with the GP derived law. The specific implementation for the RL an GP guidance will be discussed lately in the dedicated sections. As previously done, Figure 4.11 shows a conceptual representation of the GCC

architecture including also the GP and NN laws.

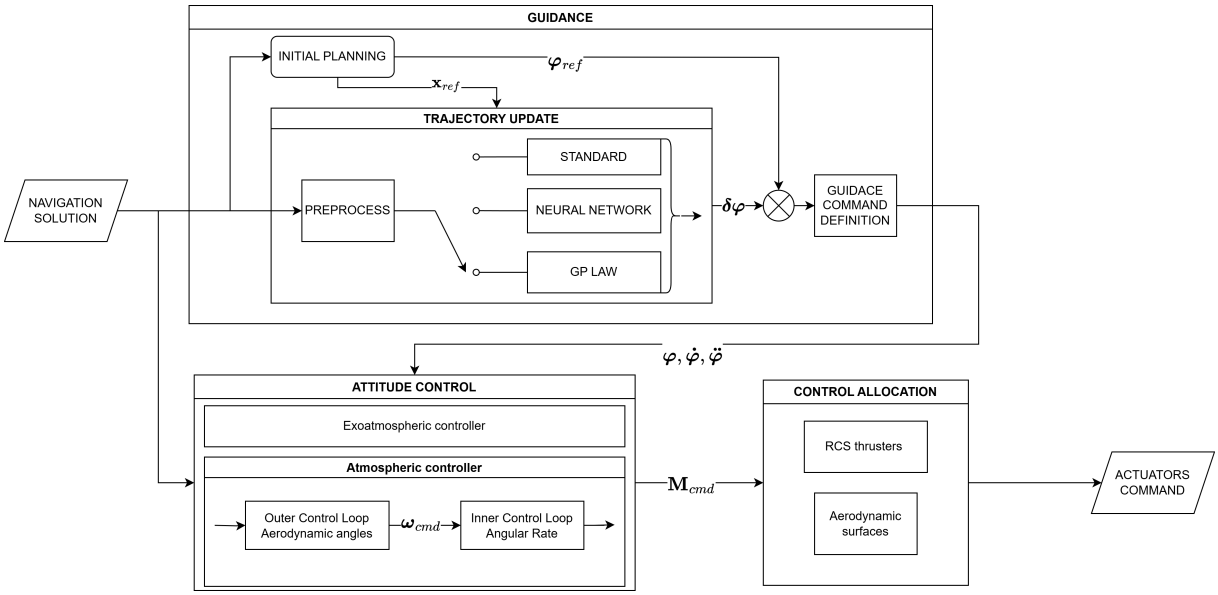


Figure 4.11: GCC conceptual architecture: $\boldsymbol{\varphi}$ is the set of guidance commands α and μ, $\mathbf{x}$ is the GEOC state vector, $\boldsymbol{\omega}_{cmd}$ is the commanded angular rate vector and $\mathbf{M}_{cmd}$ is the commanded torque

Control

Since after the separation from the launcher, the vehicle is injected into a suborbital trajectory, the controller is divided into two subsystems: an exoatmospheric and an atmospheric controller. The exoatmospheric controller is used in the corresponding flight phase, in which the dynamic pressure is not high enough to produce relevant aerodynamic forces, therefore the attitude is only controlled via cold gas thrusters based RCS. Contrarily, when the dynamic pressure gets high enough to allow for aerodynamic attitude control via aerodynamic surfaces, the atmospheric controller is triggered. A detailed description of the control system is outside the scope of this thesis but can be found in [62, 67].

Guidance

One of the the objectives of the ReFEx mission is to reach a prescribed final condition that is defined in terms of position and velocity with a given tolerance. In order to meet these requirements, the guidance algorithm is designed to perform two main tasks:

1. *Perform a major trajectory update*: since the launch phase is unguided, there is a huge uncertainty on the separation conditions, and due to the ballistic flight until the EI, the uncertainty will increase further.

During the exoatmospheric ballistic flight, the purpose of the the guidance algorithm is to solve a simplified optimal control problem and generate the main reference trajectory accounting for the navigation solution [62].

2. *Perform regular minor trajectory updates*: corrects the accumulating effect of modeling uncertainties and control errors solving again a simplification of the optimal control problem [65]. Due to these limitations, approximately the 26% of flight simulations aiming to just follow the initial trajectory update, fail to meet the requirements. Hence, a periodic trajectory update is necessary in order to ensure the success of the mission.

The output of the Guidance software is the set-point of both the attack angle α and the bank angle μ while the sideslip angle β is always commanded to *0*. The control signal is then generated and the aerodynamic surfaces actuator command is produced. An high level scheme of the full GNC architecture is reported in Figure 4.12 while a more inclusive overview of the algorithm is offered in Appendix A.

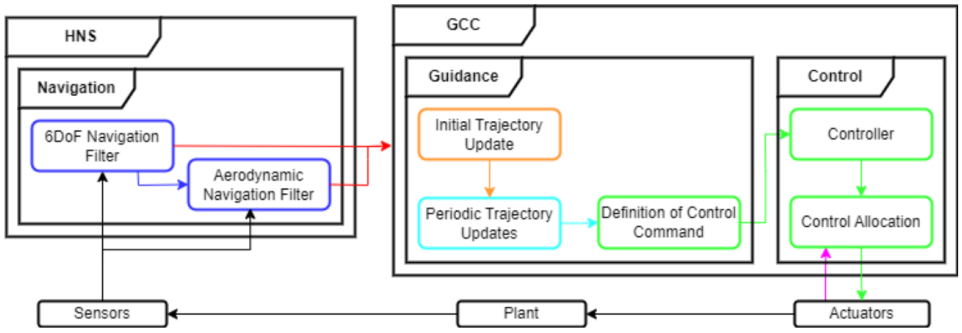


Figure 4.12: High level GNC system structure [62]

5 | Training Framework Definition

The objective of this thesis is to compare RL and GP on the same guidance problem. To ensure a fair comparison, both algorithms have been configured to operate under the most similar possible conditions. This chapter outlines the general training framework, with particular emphasis on which parameters are kept constant between the two methods and which, by necessity, differ due to the intrinsic characteristics of each algorithm.

One of the main challenges of this project is developing a guidance system capable of coping with environmental uncertainties, the most critical of which is the significant dispersion in initial conditions. Although GP has already demonstrated the ability to generate a guidance law that effectively manages environmental uncertainties under fixed initial conditions [46], its performance across the entire scenario remains unproven. Furthermore, no prior tests involving RL have been conducted in this specific context.

The chapter is organized as follows: in section 5.1 it is detailed the input/output interface of the guidance program which is common between the RL and GP explaining in particular why some specific parameters have been chosen. The way the training dataset is generated is described in section 5.2 while section 5.3 explains in detail how the GP fitness function and the RL reward signal are designed.

5.1. Guidance Law Interface

The first iteration of guidance laws design was conducted using the same setup as in [46], that consisted in using as input variables the absolute GEOC state adimensionalized. However the analysis reported in [46] aimed to solve a simplified version of the problem here analyzed: that case considered a fixed initial condition while the objective now is to account for the full dispersion at the separation from the launcher. This solution even though it could achieve good results in the case described in [46], it yielded poor results in the present scenario, probably due to the difficulty for a guidance law of that kind, to generalize a so wide variability in the initial conditions.

This outcome highlighted the need to modify both the input variables and the method

for selecting training dataset, which originally consisted in choosing the training scenarios evenly spaced according to the severity of the corresponding perturbation but without accounting for the initial condition since it was fixed. The modifications in the input variables are discussed here, while the method for generating the training dataset is detailed in the next section.

Regarding the input/output interface, the primary requirement for the guidance law is to rely on quantities provided by the HNS and available to the GCC. For this reason, the GEOC absolute state vector $\mathbf{x} = [h, \lambda, \phi,\ v, \gamma,\ \chi]$ has been retained.
The state vector has been augmented with the tracking errors associated with the GEOC states, which are hereafter referred to as relative components, computed as the difference between the current state and the corresponding component in the reference trajectory. The reference trajectory is generated during the initial planning phase of the guidance algorithm so it is available to the GCC during the guided reentry phase. During flight, each relative state is obtained by comparing the absolute coordinates to the reference state at the same energy level as shown in Equation 5.1.

$$x_i^{rel}(t) = x_i(t) - x_i^{ref}(E(t)) \tag{5.1}$$

As mentioned at the beginning of the section, the wide dispersion in the initial conditions has been identified as problematic for the guidance law formulation which shall be able to produce a general solution valid for each of the initial states. The relative component added to the input variables of the guidance law seen in Equation 5.1 has the effect of mitigating this problem from a numerical point of view.
At the beginning of the reentry phase, the trajectory is very close to the reference one so the x_i^{rel} components will assume values in a much smaller range if compared to their absolute counterparts x_i. In general this effect (having input variables in a smaller range) is valid during the whole trajectory and it helps the numerical stability of the algorithms. However, considering as input variables for the guidance law the tracking errors only, would prevent the vehicle from knowing its position with respect to the target and the final accuracy would have been intrinsically bounded to the precision of the reference itself which might not be exact. For this reason, also the GEOC absolute state vector is considered in the set of input variables for the guidance law. Finally, the state vector is further extended to include the energy term, which encodes information about the current flight phase relative to the overall mission.

To improve training performances, all input states have been normalized: absolute states and the energy term are scaled to fall within the range [0, 1], while relative states are

scaled to lie within $[-1,\ 1]$. The normalization is performed in Equation 5.2 by defining fixed minimum and maximum values, which are used as reference bounds for scaling each input component. Note that symbols marked with a hat $(\hat{\cdot})$ refer to scaled quantities.

$$\begin{aligned} \hat{x}_i &= \frac{x_i - x^i_{min}}{x^i_{max} - x^i_{min}} \qquad \hat{x}_i \in [0, +1] \\ \hat{x}^{rel}_i &= \frac{x^{rel}_i}{x^{rel,\ max}_i} \qquad \hat{x}^{rel}_i \in [-1, +1] \end{aligned} \tag{5.2}$$

This leads to the final expression of the input state vector in Equation 5.3 which is used both for the GP and for the RL. In the latter case, it takes the name of **observation vector**.

$$\mathbf{x}_{in} = \left[\hat{h}_{rel},\ \hat{\lambda}_{rel},\ \hat{\phi}_{rel},\ \hat{v}_{rel},\ \hat{\gamma}_{rel},\ \hat{\chi}_{rel},\ \hat{h},\ \hat{\lambda},\ \hat{\phi},\ \hat{v},\ \hat{\gamma},\ \hat{\chi},\ \hat{E}\right] \tag{5.3}$$

5.2. Training Dataset Selection

When dealing with ML techniques, the selection of the training dataset is of primary importance in order to ensure good performances. As already mentioned, the present study involves a training environment different from the testing one therefore during training the objective is to have an environment as representative as possible of the testing one. To achieve this goal, two kind of uncertainties have been defined:

- Model/Environmental uncertainties: vehicle mass and aerodynamic coefficients, atmospheric properties and initial conditions;
- Noises: these are noises applied to the input and output of the Guidance and Navigation algorithms during the training phase to account for the errors that will occur in the real application.

This section will focus on the first type of uncertainties since the second one are directly embedded in the training simulator logic as detailed in section 4.3.

Model and environmental uncertainties are generated using specific routines designated to attribute to each parameter a realistic value by sampling from predefined probability density functions. The uncertainty generation process has been constructed to sample the uncertainties always with the same seed for the selected run to ensure the reproducibility of results and it is integrated in the initialization phase of the testing simulator. Both the training and validation datasets, each consisting of 500 elements, have been generated in this way.

In this dissertation, the term **Uncertainty Case (UC)** refers to an instance encom-

passing all model and environmental uncertainties, such as vehicle mass, aerodynamic coefficients values, atmospheric properties and initial conditions, combined.

5.2.1. Dataset Generation

The dataset is generated by creating 1000 UCs and for each of them it is performed a flight simulation using the original DLR implementation of the 3DOF simulator. During these simulations, the vehicle only tracks the reference guidance signal produced during the initial planning phase (computed during the exoatmospheric phase before re-entering the atmosphere) without actually correcting it as the complete baseline guidance algorithm would have done. By logging the error value with respect to the target it is possible to gain an information about the severity of the uncertainty case itself. Knowing the severity of the perturbations present in each of them is a key information for the selection of the uncertainty cases to be used during the training.
The error with respect to the target is computed by taking the difference between the state vector of ReFEx at the EoE and the target state. Equation 5.4 define the relative error component-wise where ϵ_i is the absolute tolerance on the i-th state component. The total error, defined as in Equation 5.5 is instead the norm of the vector defined in Equation 5.4.

$$e_{EoE,i} = \frac{|x_{EoE,i} - x_{T,i}|}{\epsilon_i} \tag{5.4}$$

$$e_{EoE} = \sqrt{\sum_i e_{EoE,i}^2} \tag{5.5}$$

The output of this process consists in the state vector at the EI for each uncertainty case considered, coupled with the corresponding end-of-trajectory error, computed as above mentioned, and the output of the guidance planning algorithm. The latter includes the command sequence computed during the planning phase, tabulated as a function of the specific energy and the corresponding trajectory. This trajectory is intended as the trajectory that is computed during the optimization occurring as part of the initial planning routine which consists in a 3DOF propagation accounting for the specific initial condition but with an unperturbed environmental model. The reference trajectory is then exported as the sequence of GEOC coordinates in the ECEF reference frame tabulated with respect to the specific energy.

This process is repeated for the 1000 uncertainty cases that are considered within this study. The full dataset is finally partitioned in two: uncertainty cases from 1 to 500 will be considered as training dataset while uncertainty cases from 501 to 1000 will be

considered as testing dataset.

5.2.2. Selection Algorithm

While the testing dataset could be easily used in its entirety, the same cannot be said for the training one: this is due to the specific features of RL and GP for which the training dataset size affects the total training computational time. However, these issues will be discussed in paragraph 5.2.4 for the RL case, and in paragraph 5.2.3 for the GP one. The objective of this section is therefore to detail how to select a given number of uncertainty cases from the full training dataset. The algorithm designed to perform this task is reported in Algorithm 5.1 and is described hereafter.

Algorithm 5.1 Uncertainty cases selection algorithm

Require: $\mathbb{D}$ is the full initial dataset with size $[n_D \times 7]$
Require: Each row of $\mathbb{D}$ contains the parameters in $\mathbb{P}$ (Equation 5.6)
1: **for** $s \in [h,\ \lambda,\ \phi,\ v,\ \gamma,\ \chi]$ **do**
2: $data \leftarrow \mathbb{D}_{1:n_D,\ s}$ array {Extract initial conditions array corresponding to the selected component}
3: $l_{val,s},\ l_{idx,s} \leftarrow min\,(data)$ {Log both the value and the corresponding index}
4: $u_{val,s},\ u_{idx,s} \leftarrow max\,(data)$ {Log both the value and the corresponding index}
5: $m_{val,s},\ m_{idx,s} \leftarrow mean\,(data)$ {Log both the value and the corresponding index}
6: **end for**
7: $data_{err} \leftarrow \mathbb{D}_{1:n_D,\ e_{EoE}}$ array
8: $e_{val},\ e_{idx} \leftarrow$ select n_e equally spaced elements from e_{min} to $\max\,(data_{err})$ from $data_{err}$ {Log both the values and the corresponding indexes}
9: $c_{idx} \leftarrow$ concatenate $l_{idx},\ u_{idx},\ m_{idx},\ e_{idx}$ {Indexes of the selected points within $\mathbb{D}$}
10: $N_p \leftarrow 18 + n_e$
11: **for** $k \leftarrow 1$ **to** N_p **do**
12: $j \leftarrow c_{idx,k}$ {Extract the k-th index from the array with the concatenated indexes}
13: **for** $s \leftarrow 1$ **to** 7 **do**
14: $\hat{p}_s \leftarrow \dfrac{\mathbb{D}_{j,s} - l_{val,s}}{u_{val,s} - l_{val,s}}$
15: **end for**
16: $f_k \leftarrow \sum_{i=1}^{7} w_i \cdot \hat{p}_i$
17: **end for**
18: sort(f)
19: $f_{sel} \leftarrow$ sample n_{uc} points equally spaced from f {select the fitness of the points to be used in the reduced database}
20: $idx_{sel} \leftarrow$ indexes in $\mathbb{D}$ of the points corresponding to the f_{sel} values
21: $pt_{sel} \leftarrow \mathbb{D}_{idx_{sel},1:6}$ {Coordinates of the selected points}
22: **return** pt_{sel}

Due to the constraints in the number of UCs that can be used during the training, it

is crucial to select a set as informative as possible with respect to the full distribution of uncertainties. Since one of the biggest challenges associated to the GNC subsystem for the ReFEx mission is dealing with the huge dispersion in the initial conditions, the parameters considered to determine the level of informativeness related to an uncertainty cases are: its state at initial condition (i.e. EI), and the error at the end-of-trajectory obtained following the initial planning reference commands (paragraph 5.2.1). Such error aims to represent the severity of the perturbations considered in the UC under evaluation. To conclude, 7 scalar parameters are considered when selecting the uncertainty cases to be used during the training, namely:

$$\mathbb{P} = [h_{EI}, \lambda_{EI}, \phi_{EI}, v_{EI}, \gamma_{EI}, \chi_{EI}, e_{EoE}] \tag{5.6}$$

In order to have a representative subset, both the nominal conditions and the worst case one should be included in the selection and in general, when dealing with a multidimensional distribution, a given uncertainty case might be a worst case for some parameters while being close to the nominal for others. Hence, the following uncertainty case selection algorithm (reported also in Algorithm 5.1) has been designed. The algorithm takes as input the following parameters:

- n_{uc}: number of uncertainty cases to be selected;
- n_e, e_{min}: number of points and minimum value of error to be considered in the error space when creating the subdivision, their functionality will be clarified later;
- $\mathbb{D}$: training dataset of size $n_D \times 7$, contains the 7 parameters listed in Equation 5.6 for each of the n_D elements in the dataset.
- $\mathbf{w}$: relative weights associated to each parameter, the algorithm is designed to allow attributing different priority levels to the spacing in each parameter space. This means that the uncertainty cases selected will have a much comprehensive spacing with respect to the parameters with higher priority.

For each of the six coordinates parameters, the minimum, maximum and mean values within the dataset have been computed and the corresponding index is stored. A slightly different procedure is applied to the e_{EoE} parameter: in this case it is created an array of n_e equally spaced elements ranging from the minimum error value prescribed e_{min} to the maximum value present on the dataset. This passage is included in order to prevent the algorithm to only select points with very low error value due to the skewness of the distribution towards low values.
In this way it is created a subset $\mathbb{S}$ of $18 + n_e$ elements selected from $\mathbb{D}$ (the value 18

comes from the fact that we are selecting 3 points for each of the 6 states).
Once $\mathbb{S}$ has been computed, to each point it is assigned a fitness value computed as follows:

$$f = \sum_{i=1}^{7} w_i \cdot \hat{p}_i \tag{5.7}$$

where $\hat{p}_i$ represents the value of the i-th parameter (referring to the set in Equation 5.6) scaled to have a numerical value between 0 and 1.
The last stage of the algorithm orders the elements in $\mathbb{S}$ according to their fitness value f and samples n_{uc} points equally spaced with respect to the f value.

In Figure 5.1 the full dataset is represented highlighting the single components of the state vector at the beginning of the simulation. Each dot represents a single uncertainty case and in each figure it is plotted the total error computed at the EoE as function of the initial value of each state vector component. Since within the simulation the aerodynamic forces acting on the vehicle are actually computed only below the 60 km altitude, the initial conditions considered for the training purposes are retrieved by taking the first sample of each reference trajectory below the 60 km altitude threshold. This explain the different distribution of the altitude samples with respect to all the others.

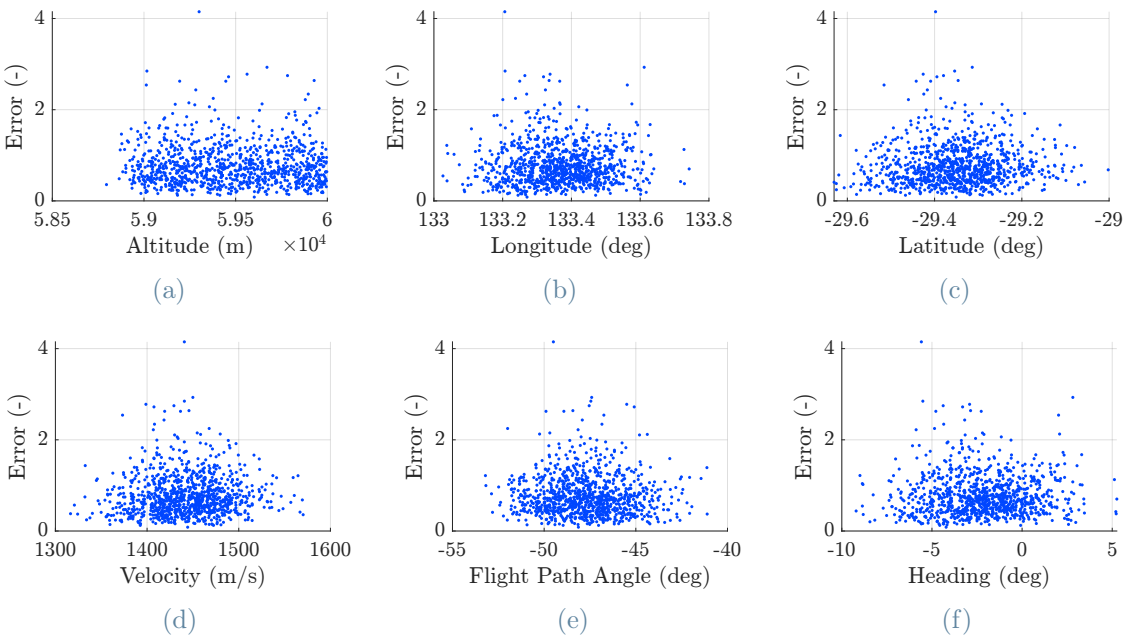


Figure 5.1: ReFEx state at the beginning of the simulation

In subsection 5.2.3 and subsection 5.2.4 the specific dataset used for both techniques will be detailed.

5.2.3. Genetic Programming uncertainty cases

In the GP framework the selection of the training dataset is particularly important due to the fact that the evolution algorithm requires at each iteration to fully simulate from the EI to the EoE each UC that is included in the training dataset. This feature imposes a very strong constraint in the number of uncertainty cases that can be included in the dataset itself: a too large training dataset will lead to unacceptably high computational times. On the other hand, a too small dataset would not be informative enough to obtain acceptable results. The tradeoff between these two requirements has been found in selecting a dataset containing 25 elements selected with the algorithm detailed in subsection 5.2.2. The chosen uncertainty cases are shown in Figure 5.2. Notice that the same 25 selected uncertainty cases are highlighted in each figure, each of them showing a different component in the state vector at the initial conditions.

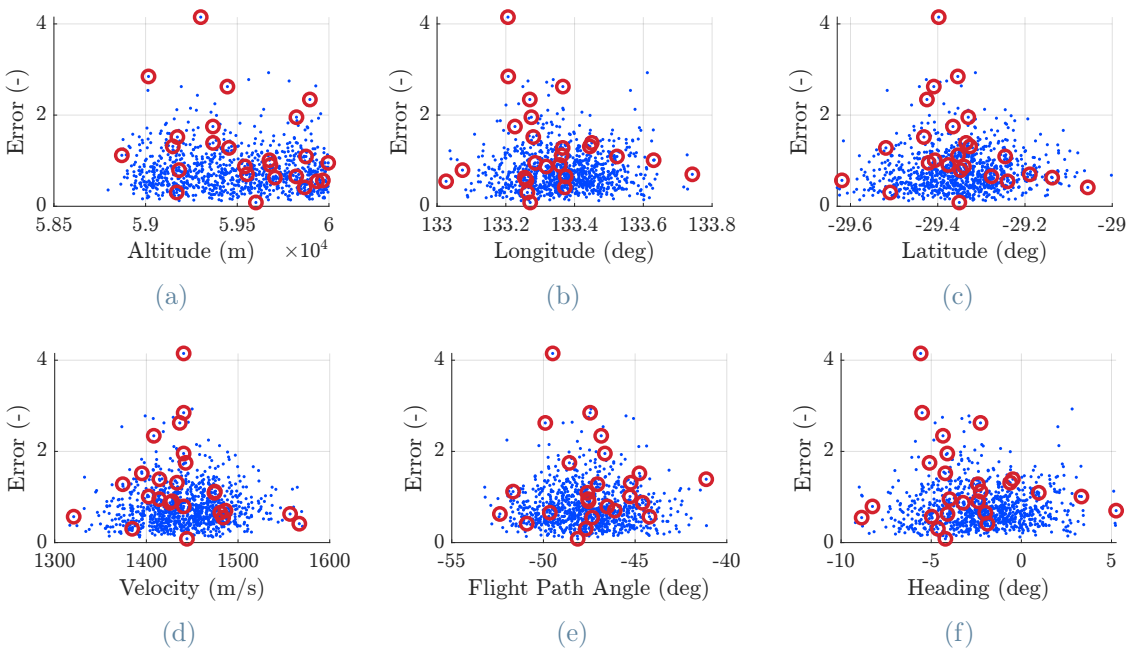


Figure 5.2: GP training dataset

Due to the small number of uncertainty cases that can be used, the selection algorithm parameters have been tailored aiming to include in the dataset both the extreme cases and the nominal ones.

5.2.4. Reinforcement Learning uncertainty cases

Differently from the GP, the RL can sample from the training dataset a subset of uncertainty cases and propagate only them at each training iteration. Between different iterations, the sampled subset will be different and in this way within the full training it is possible to include a higher number of uncertainty cases. The training dataset should then be large enough to contain enough information to allow the agent to properly learn the task but on the other hand, a larger dataset will require a larger computational time to be fully exploited during the training itself (i.e. each UC should be considered several times during the training).
In the present study, three different datasets have been employed and their results compared:

1. 25 elements dataset, the same used during the GP training (see subsection 5.2.3)
2. 60 elements dataset, as shown in Figure 5.3: in this case the dataset is generated giving the same relative weight to each state component since the relatively high cardinality of the dataset itself allows for a proper representation of the full uncertainty distribution. As in the GP case, the same selected uncertainty cases are highlighted in each figure.
3. 500 elements dataset, in this case the full training dataset generated as described in subsection 5.2.1 is imported and used to train the agent.

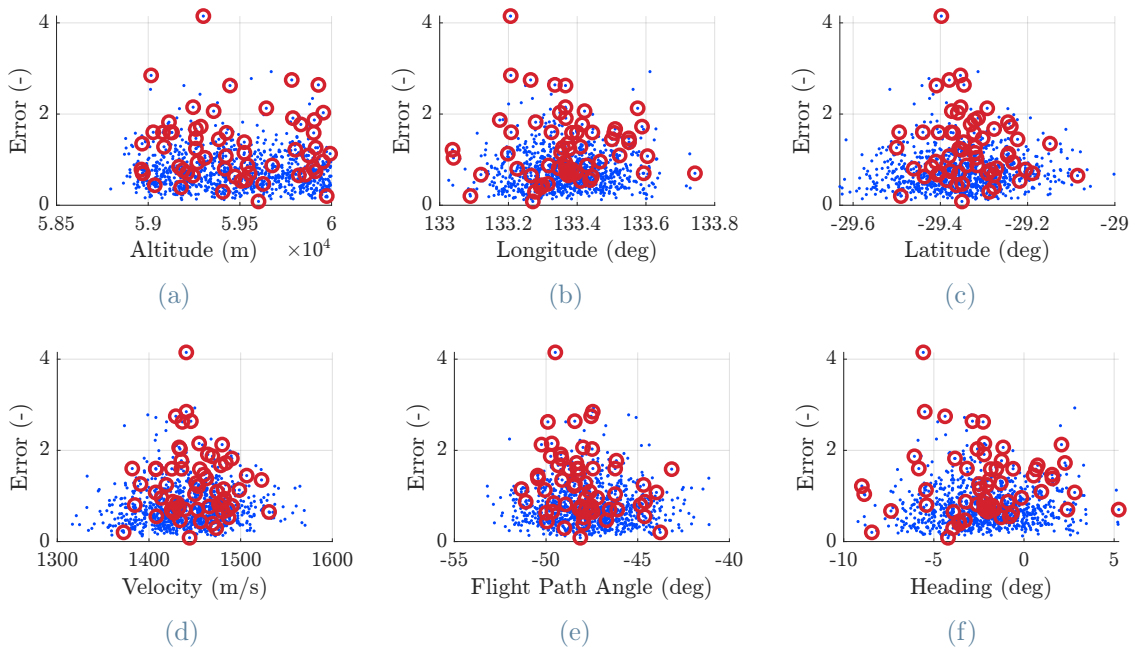


Figure 5.3: RL training dataset

5.3. Learning Functions Design

Both RL and GP require the definition of a function that evaluates the performance of the current guidance law by assigning it a reward and a fitness value respectively. The maximization/minimization of this value directly leads to the optimal guidance law itself. Since the learning mechanisms of GP and RL differ significantly, particularly in how the fitness function and the reward signal are formulated, these aspects are discussed in separate sections: paragraph 5.3.1 covers the GP fitness function, while paragraph 5.3.2 addresses the RL reward signal. Although the two functions differ in several non-negligible ways, they ultimately serve the same purpose. For this reason, the term **learning function** is used throughout the text to refer to both when appropriate. Defining the learning function properly is essential for successful training: it must clearly express the training objectives, rewarding desirable behaviors and penalizing undesirable ones. A poorly defined learning function can lead to suboptimal agent performance or even promote unexpected behaviors that may become dangerous when deployed in a real-world environment.

Before beginning the discussion about the implementation of the specific learning functions, it is noteworthy that the vehicle is designed to satisfy the set of constraints to ensure its physical integrity (i.e. limits on dynamic pressure, heat flux and load factor) through the entire trajectory envelope, therefore it is not needed to enforce them during the trajectory updates. On the other hand, constraints to ensure the vehicle controllability, namely constraining the trajectory in a defined Mach-α corridor to avoid regions in which it is underactuated, and limiting the rate and acceleration of the guidance commands ($\dot{\alpha}$, $\ddot{\alpha}$, $\dot{\mu}$, $\ddot{\mu}$) are already enforced within the simulator itself (see section 4.3).

5.3.1. Genetic Programming Fitness Function

The GP training proceeds by dividing the evolution into generations. In each generation, a set of individuals is evaluated, and a fitness value is assigned to each of them based on its performance. Every individual is evaluated across the entire training dataset, ensuring that the fitness value reflects the behavior over all the propagated trajectories under each uncertainty case.

The IGP algorithm considers a fitness value and a penalty value: it first drives to zero the penalty which acts as a constraint on the individual; once all the constraints are satisfied (i.e. the penalty is identically null), the IGP minimizes the fitness function which has the objective of encoding the actual desired flight performances.

Penalty Function

As aforementioned, the evolutionary penalty is used to apply some constraints to the guidance law that must be satisfied during the evolution. In this case, two different constraints are applied. The first requires the EoE point (i.e. the propagation exit condition) to happen at a positive altitude with respect to the ground level. This is enforced in order to strongly penalize those guidance laws that will result in crashing the vehicle on the ground. The second constraint imposes a limitation in the guidance command range. In particular, the bank angle correction is limited to the $\pm$ 90° range, while the angle of attack correction is limited such that the commanded angle of attack is within the range in which the aerodynamic data are defined (i.e. in between $-50°$ and $+20°$). This is done in order to prevent the guidance law to generate asymptotes and more generally commands that are a-priori unfeasible.
The above mentioned penalties are implemented as in Equation 5.8 where the terms h_{ref}, α_{ref}, μ_{ref} and t_{ref} are user selected parameters to adimensionalize the penalty terms.

$$\begin{aligned} p_{alt} &= \begin{cases} \dfrac{|h_{EoE}|}{h_{ref}} & \text{if} \quad h_{EoE} \leq 0 \\ 0 \quad \text{if } h_{EoE} > 0 \end{cases} \\ p_{\alpha} &= \frac{1}{t_{ref} \cdot \alpha_{ref}} \cdot \int_{t_0}^{t_{EoE}} e_{\alpha}(t) \ \ dt \\ p_{\mu} &= \frac{1}{t_{ref} \cdot \mu_{ref}} \cdot \int_{t_0}^{t_{EoE}} e_{\mu}(t) \ \ dt \end{aligned} \tag{5.8}$$

where, using θ as general parameter to refer both to α and to μ:

$$e_{\theta}(t) = \begin{cases} 0 & \text{if } \theta_{com}(t) \in [\theta_{min}, \theta_{max}] \\ |\theta_{com}(t) - \theta_{max}| & \text{if} \quad \theta_{com}(t) > \theta_{max} \\ |\theta_{com}(t) - \theta_{min}| & \text{if} \quad \theta_{com}(t) < \theta_{min} \end{cases} \tag{5.9}$$

The final penalty value associated to a given uncertainty case is then computed by summing the three contributions that appear in Equation 5.8 as follows:

$$p_{uc} = p_{alt} + p_{\alpha} + p_{\mu} \tag{5.10}$$

Once all the penalty values associated to each uncertainty case evaluated by a given

individual $p_{uc,i}$ are known, the penalty value associated to the individual is obtained as:

$$p_{ind} = \frac{\sum_{i=1}^{n_{uc}} p_{uc,i}}{n_{uc}} \tag{5.11}$$

Fitness Function

Defining the fitness function is more complex than specifying the penalties, as it is responsible for guiding the evolution. This function provides the only information available to the GP for evaluating the performance of individuals, which must be expressed through a single scalar value.

Since the objective of the evolution is to maximize the number of uncertainty cases in which the guidance law succeeds in driving ReFEx to the target state, the fitness function must reflect this driver. Within this study, a run is considered successful if each of the relative error components defined in Equation 5.4 is lower than 1, meaning that the vehicle reaches the EoE within an hypercube centered on the target state $\mathbf{x}_T$ and with tolerance ϵ_i on each dimension.
Since each individual during the training evaluates n_{uc} uncertainty cases, the fitness function is designed to have a step-wise behavior determined by the number of successes. Hence, independently from the specific performances, if a given individual can achieve a greater number of successes than another, the former individual will have a lower (better) fitness value than the latter. In this way, the success rate associated to each individual determines the range in which its fitness value will belong to. The specific value within the range is then computed according to the exact performances of the considered guidance law across the considered uncertainty cases.
This approach leads to the following fitness function structure:

$$F_{ind} = F_0 + g \tag{5.12}$$

where F_0 represents the first order contribution connected to the success ratio of the given individual, while g represents the second order contribution connected to the performances of the individual on each single uncertainty cases.

Computation of F_0

The success rate is computed as $SR = \frac{n_{success}}{n_{uc}}$. Conversely, the failure rate is given by $FR = \frac{n_{fail}}{n_{uc}}$. Alongside it, $\epsilon_{RF} = \frac{1}{n_{uc}}$ represents the minimum possible increment in the

FR parameter. Based on these definitions, the F_0 value is then computed as follows:

$$F_0\left(FR\right) = 10^{K_{F_0} \cdot FR} - 1 \tag{5.13}$$

the function has this form in order to have $F_0(FR) = 0$ when the failure rate is 0 (i.e. all the uncertainty cases are solved successfully) and to guarantee a larger separation between consequent layers while the failure rate increases. The parameter K_{F_0} has the objective of controlling the layer size with respect to the failure rate and is a user defined constant.

Computation of g

The computation of g goes through two sequential phases:

1. Calculate the fitness values $f_{uc,i}$ associated to the performances of the guidance law under evaluation on each uncertainty case in the training dataset;
2. Aggregate all the $f_{uc,\ i}$ value to determine the overall performances of the guidance law on the training dataset.

The fitness value associated to each uncertainty case, $f_{uc,\ i}$ accounts for three contributions:

1. $f_{EoE,\ i}$: error with respect to the target at the EoE;
2. $f_{trj,\ i}$: integral of the error with respect to the reference trajectory in the last t_{GO} seconds of simulation;
3. $f_{corr,\ i}$: penalization in case the flight trajectory leaves the flight corridor envelope in Figure 4.8.

The term $f_{EoE,\ i}$ is designed to be the most significant component in the $f_{uc,\ i}$ value: it is the guidance law performance main driver and it measures the error with respect to the target at the EoE. Minimizing this error is the objective of the guidance law. $f_{EoE,\ i}$ is computed according to Equation 5.14.

$$f_{EoE,\ i} = \begin{cases} \ln\left(\hat{e}^2_{EoE}\right) & \text{if } FR = 0 \\ \max\limits_i \left(e^2_{EoE,i}\right) & \text{if } 0 < FR < 1 \\ \sum\limits_i \left(e_{EoE,\ i} - 1\right)^2 & \text{if } FR = 1 \end{cases} \tag{5.14}$$

$f_{EoE,\ i}$ is computed differently according to the failure rate FR value. When the guidance law can solve all uncertainty cases, the total error will belong to the interval $e_{EoE} \in \left[0,\ \sqrt{5}\right]$ since in the worst case each of the relative error components $e_{EoE,i}$ will be 1.

Notice that as will be shown also in section 6.1 the flight path angle at the EoE is not constrained, therefore the target state is defined only by 5 scalar values. In this way it is possible to use a scaled version of the total error $\hat{e}^2_{EoE}$ to assume values between 0 and 1. The cost is then defined logarithmically to be inferiorly unbounded.
When the failure rate is between 0 and 1, the fitness function is modified to prevent the GP from having the possibility to "hack" it by achieving a very high precision on some components while leaving others outside the requirements, for this reason the cost associated to each uncertainty case is based on the performances related to the worst case component.
Finally when the guidance law fails to drive the vehicle to the target in all of the training cases, the cost is computed quadratically to be superiorly unbounded.

The objective of the term $f_{trj,\ i}$ is to help the GP convergence by exploiting the reference trajectory computed during the planning phase which achieves a very high accuracy even if it might be unfeasible due to the environmental perturbations. Hence, $f_{trj,\ i}$ is the integral of the distance between the actual trajectory and the reference one, evaluated over the final part of simulation. The duration of the arch terminating with the EoE that has to be considered is a user defined variable depending on the problem itself and will be here referred with t_{GO}. In this way the agent is gently pushed to follow a path similar to the reference one but without copying it.

$$f_{trj,\ i} = \int_{E_{t_{GO}}}^{E_{EoE}} 10 \cdot e_{ref}\left(E\right)\ dE \tag{5.15}$$

$$e_{ref}\left(E\right) = \left|\left|\underline{\underline{Q}} \cdot \left[\mathbf{x}\left(E\right) - \mathbf{x}_{ref}\left(E\right)\right]\right|\right| \tag{5.16}$$

$$\underline{\underline{Q}} = diag\left(\epsilon_h^{-1},\ \epsilon_\lambda^{-1},\ \epsilon_\phi^{-1},\ \epsilon_v^{-1},\ 0,\ \epsilon_\chi^{-1}\right) \tag{5.17}$$

In Equation 5.15, the integration is performed with respect to the energy because it is the independent variable used as reference for the design of the whole GC subsystem in order to handle trajectories with unconstrained time of flight. The computation of the $e_{ref}\left(E\right)$ term is detailed in Equation 5.16 and Equation 5.17. $E_{t_{GO}}$ in Equation 5.15 refers to the energy the vehicle has at t_{GO} before reaching the EoE. This computation is easily done since it is performed after the full propagation is completed. The energy is linearly scaled to assume values between 0 and 1, where the extremes are the minimum (i.e. E_{EoE}) and maximum energy levels during the trajectory. This is done to keep the results numerically coherent with the other terms.

The last term, $f_{corr,\ i}$, is related to the quality of aerodynamic data: the flight envelope

should belong to the flyable corridor defined in the Mach-α domain shown in Figure 4.8. Having a portion of the trajectory outside this corridor would mean performing that part of simulation with aerodynamic data obtained via extrapolation from the CFD ones, lowering the reliability of the obtained results. However, if the trajectory leaves the corridor for a short time and stays in the close proximity of its boundaries, the estimated aerodynamic coefficients can still be considered reliable.
The $f_{corr,\ i}$ contribution is computed as follows:

$$f_{corr,\ i} = \int_{E_0}^{E_{EoE}} d\left(E\right)\ dE \tag{5.18}$$

$$d\left(E\right) = \begin{cases} w_O \cdot D\left(E\right)^2 & \text{if} \quad \alpha < \alpha_{c,\ min} \vee \alpha > \alpha_{c,\ max} \\ w_I \cdot D\left(E\right) & \text{if} \quad \alpha_{c,\ min} \leq \alpha \leq \alpha_{c,\ max} \end{cases} \tag{5.19}$$

$$D\left(E\right) = \left| 2\frac{\alpha\left(E\right) - \alpha_{c,\ min}\left(E\right)}{\alpha_{c,\ max}\left(E\right) - \alpha_{c,\ min}\left(E\right)} - 1 \right| \tag{5.20}$$

Equation 5.20 represents the normalized distance from the center of the corridor during the trajectory. At each timestep, the mach number M is computed and according to it, the corridor boundaries in terms of angle of attack are retrieved: $\alpha_{c,\ min}$, $\alpha_{c,\ max}$.
The values of the coefficients w_I (inner) and w_O (outer) in Equation 5.19 are tunable parameters.
As in the $f_{trj,\ i}$ case, the integral in Equation 5.18 is performed using a normalized value of energy between 0 and 1.

$f_{uc,\ i}$ is finally computed by summing up the terms $f_{EoE,\ i}$, $f_{trj,\ i}$ and $f_{corr,\ i}$ as shown in Equation 5.21. The weights values w_{corr} and w_{trj} are defined by the user.

$$f_{uc,\ i} = \begin{cases} \sigma\left(f_{EoE,\ i} + w_{trj}\ f_{trj,\ i} + w_{corr}\ f_{corr,\ i}\right) & \text{if the i}^{\text{th}} \text{ uncertainty case is successful} \\ f_{EoE,\ i} + w_{trj}\ f_{trj,\ i} + w_{corr}\ f_{corr,\ i} & \text{if the i}^{\text{th}} \text{ uncertainty case is failed} \end{cases} \tag{5.21}$$

The saturation function $\sigma\left(\cdot\right)$ is implemented to prevent the fitness value associated to successful uncertainty case from being greater (i.e. worse) that the one associated to a failed case. This property must be enforced while always maintaining the proper ordering of the fitness values associated to different uncertainty cases: $f_{uc,\ j} \lesseqgtr f_{uc,\ k} \Leftrightarrow \sigma\left(f_{uc,\ j}\right) \lesseqgtr$

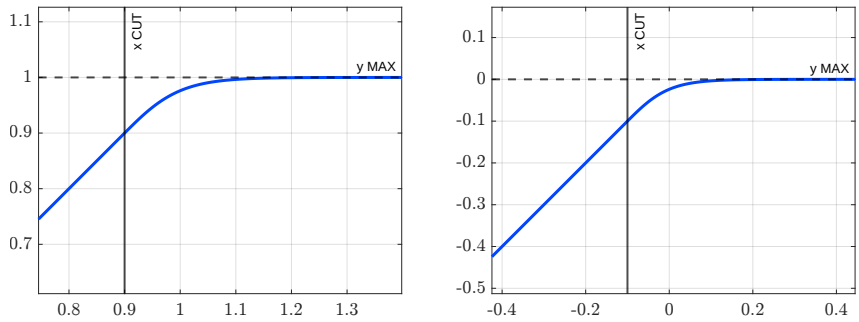
$\sigma\left(f_{uc,\ k}\right)$. The saturation function is then constructed as follows:

$$\sigma_{inner}\left(x\right) = \begin{cases} x & \text{if } x \leq x_{cut} \\ x_{cut} + \left(y_{max} - x_{cut}\right) \tanh\left(\dfrac{x - x_{cut}}{y_{max} - x_{cut}}\right) & \text{if } x \geq x_{cut} \end{cases} \tag{5.22}$$

where $x_{cut} = 0.9$ and $y_{max} = 1$.
Moreover, since according to the specif case of failure rate the fitness value of a successful uncertainty case can be maximum equal to 0 (in the subcase in which $FR = 0$) or equal to 1 (when $0 < FR < 1$), the function described in Equation 5.22, can be translated to saturate at the same value (i.e. 0 or 1), therefore the final implemented expression is the following one:

$$\sigma\left(x\right) = \sigma_{inner}\left(x - x_{offset}\right) + y_{offset} \tag{5.23}$$

which results in the two functions plotted in Figure 5.4.



(a) Nominal function — saturates at 1 $x_{offset} = 0$, $y_{offset} = 0$

(b) Shifted function - saturates at 0 $x_{offset} = -1$, $y_{offset} = -1$

Figure 5.4: GP training saturation functions

Finally, the various $f_{uc,i}$ are aggregated in final expression of g reported in Equation 5.24.

$$g = \begin{cases} F_0\left(\epsilon_{FR}\right) + \displaystyle\sum_{i=1}^{n_{uc}} f_{uc,i} & FR = 0 \\ M \cdot \tanh\left[K_g \cdot \left(median\left(f_{uc}^{fail}\right) + median\left(f_{uc}^{success}\right) - 1\right)\right] & 0 < FR < 1 \\ \dfrac{1}{2}\left(median\left(f_{uc}\right) + max\left(f_{uc}\right)\right) & FR = 1 \end{cases} \tag{5.24}$$

Where $M = \left(F_0\left(FR + \epsilon_{FR}\right) - F_0\left(FR\right)\right)$ and $K_g = \dfrac{arctanh\ 0.99}{h_{0.99}}$. $h_{0.99}$ is a tuning parameter defining the value of the tanh argument when the function returns 0.99, its value is user defined.

Equation 5.24 is divided in three cases to preserve the layer subdivision of the individual fitness value in Equation 5.12. For this reason, the case with $0 < FR < 1$ is bounded in the interval $[0, \quad F_0(FR + \epsilon_{FR}) - F_0(FR))$ while the cases with $FR = 0$ and $FR = 1$ are respectively inferiorly and superiorly unbounded.

5.3.2. Reinforcement Learning Reward Signal

The RL training is driven by the reward signal value; its computation is divided into two terms: a dense term and a terminal one. The dense reward term is given at each timestep of the propagation and it is used to direct the agent to the target. In the present implementation, this term is used to enforce some constraints and to improve the training performances even if it does not specify the actual objective of the optimization. On the other hand, the terminal reward term is given only once the episode reaches the terminal conditions and it is used to evaluate the terminal state.

Dense Reward Signal

This term aims to satisfy three drivers during the propagation:

1. Keep the agent in proximity of the reference trajectory;
2. Avoid flying in regions in which aerodynamic data are not reliable;
3. Promote a smooth guidance function.

The first driver is included to give the agent a sense of which trajectory shapes are likely to reach the target. Since no a-priori regularity can be exploited in the state trends (e.g., altitude or speed consistently decreasing), the reference trajectory computed during the planning phase is used to guide the agent toward a plausible path.

To avoid encouraging the agent to simply replicate the reference trajectory while still providing some feedback, this term is designed to return a positive reward when the agent remains within a predefined distance from it. If the agent exceeds this limit, the reward drops to zero.

Notably, the episode is never terminated only because the agent deviates too much from the reference, preserving its freedom to explore alternative strategies. The mathematical formulation of this term is as follows:

$$R_{trj,\ i} = \sum_{i=1}^{3} \frac{1}{2} \left[\tanh \left(k_{trj} \left(1 - \left| \frac{x_i(t) - x_i^{ref}(E(t))}{w_i} \right| \right) \right) \right] \tag{5.25}$$

where the i suffix refers to the component of the state vector to which the reward term

is related. In the present implementation, the terms $R_{trj,\ h}$, $R_{trj,\ \lambda}$ and $R_{trj,\ \phi}$ related respectively to altitude, longitude and latitude have been computed. The term w_i refers to the distance in the i components after which the corresponding reward term drops to zero and its value, together as the value of k_{trj} is defined by the user according to the characteristics of the problem.

The objective of the reward term in Equation 5.26 is to prevent the agent from flying in regions outside the boundaries of the aerodynamic database corridor.

$$R_{corr} = \begin{cases} 0 & \text{if } D(E) \leq 1 \\ -k_{corr} \cdot D(E) & \text{if } \ D(E) > 1 \end{cases} \tag{5.26}$$

where $D(E)$ is computed according to Equation 5.20 and the value of the constant k_{corr} is a tunable parameter.
In order to allow the agent to detect when it is flying in regions close to the boundaries of the flyable region, a further term is computed as:

$$h_{corr} = -2 \cdot D(E)^2 + 2 \tag{5.27}$$

and it is used to weight the $R_{trj,\ i}$ value lowering it (see Equation 5.32) when the agent is flying close to the limit of the flyable region. Similarly to what has been done in the GP framework, this requirement is enforced by highly penalizing reward value obtained during the episode but without ever interrupting it.

The last element belonging to the dense reward function aims to promote a smooth guidance signal. This is particularly important in the RL since it is known to suffer from the so called action-shaking problem which consists in associating very different action values to close observations [92]. This results from the fact that the training is performed by associating to each state the optimal action but there is no term in the PPO algorithm that promotes that kind of regularity (i.e. similar actions for similar states).
As will be shown in section 6.4, a noisy signal can be dangerous for the performances of the final system in the 6DOF environment therefore the smoothing term detailed in Equation 5.28 is added to the reward signal.

$$R_{smooth} = -k_{\dot{\alpha}} \left| \frac{\alpha(t_k) - \alpha(t_{k-1})}{t_k - t_{k-1}} \right| - k_{\dot{\mu}} \left| \frac{\mu(t_k) - \mu(t_{k-1})}{t_k - t_{k-1}} \right| \tag{5.28}$$

By considering this expression only, there is the risk of converging to a controller with bang-bang structure since when the signal is saturated, R_{smooth} will be 0. To avoid this,

a further term is added with the objective of penalizing the saturation of the guidance command.

$$R_{saturation} = -k_{saturation} \cdot (s_\alpha + s_\mu) \tag{5.29}$$

where

$$s_\theta = \begin{cases} 1 \text{ if } u_\theta = u_{\theta,max} \vee u_\theta = u_{\theta,min} & \text{i.e. command saturated} \\ 0 \text{ if } u_{\theta,min} \leq u_\theta \leq u_{\theta,max} & \text{i.e. command not saturated} \end{cases} \tag{5.30}$$

Notice that in the previous equation the suffix θ refers both to α or μ since they are elaborated in the same way. $k_{\dot{\alpha}}$, $k_{\dot{\mu}}$ and $k_{saturation}$ are tuning constants needed to adjust the relative values of different terms.

Terminal reward Signal

The terminal reward accounts for the position of the vehicle at the end of the episode with respect to the target coordinate. The episode is terminated when the specific energy of the vehicle goes below the target one or when the closest approach to the target occurs. Once the episode reaches the termination, the final reward value is computed as follows:

$$R_{terminal} = \begin{cases} \sum_{i=1}^{5} -k_{log} \cdot \ln e_{EoE,\ i} & \text{if } \max_i e_{EoE,i} < 1 \\ \sum_{i=1}^{5} -k_{quad} \cdot e^2_{EoE,\ i} + k_{quad} & \text{if } \max_i e_{EoE,i} \geq 1 \end{cases} \tag{5.31}$$

where $\mathbf{e}_{EoE} = [e_{EoE\ h},\ e_{EoE\ \lambda},\ e_{EoE\ \phi},\ e_{EoE\ v},\ e_{EoE\ \chi}]$ since there is no requirement in terms of flight path angle at the EoE conditions. As in the previous cases, the value of k_{log} and k_{quad} is defined by the user.

Summary

Each term in the dense and terminal reward signal features some adimensional constants which are included to balance their numerical value. In particular, the terms $R_{trj,\ i}$, R_{smooth} and $R_{saturation}$ are designed to return numeric values all in the same order of magnitude, which is selected to be one order of magnitude lower than the $R_{terminal}$ term that shall be the main contribution to the reward value. On the other hand, the $R_{corridor}$ term is designed such that it returns values in the same order of magnitude as $R_{terminal}$ so that it can compensate it in case of successful episode but obtained flying far from the flyable corridor.

Finally, the full reward value expression is the following:

$$R = h_{corr} \cdot (R_{trj,\ h} + R_{trj,\ \lambda} + R_{trj,\ \phi}) + R_{corr} + R_{smooth} + R_{saturation} + R_{terminal} \quad (5.32)$$

in which every term excluding $R_{terminal}$ is computed at each propagation step, while $R_{terminal}$ is evaluated only at the end of an episode.

5.4. Training Postprocessing

A typical problem of ML applications is overfitting. This phenomenon occurs when a model learns not only the underlying patterns in the training data but also the noise and random fluctuations. As a result, the model performs very well on the training set but poorly on unseen data because it has essentially memorized the training examples instead of generalizing from them. In order to mitigate this effect, a postprocessing stage has been included in both the RL and GP pipelines aiming to select the training iteration with the best generalization capabilities (on the 3DOF environment) on a validation dataset which contains only uncertainty cases that haven't been considered during the training.

Genetic Programming Postprocessing

The output of the GP training is the population at each generation of the evolution with its corresponding fitness and penalty values computed during the training. The postprocessing stage works by evaluating the best individual at each generation on a validation dataset of 100 uncertainty cases unseen during the training. The size of this initial validation dataset is selected to be a trade off solution between the computational time required to evaluate the best individual on each generation on the full dataset, and having some meaningful results.

However, once the first stage is performed, all the individuals scoring a success rate greater than 65% are then tested in the full 500 elements validation dataset in the 3DOF environment.

At this point the individual with the highest success rate is selected, in case several individuals have the same performances, the one with the smaller number of nodes is preferred.

Reinforcement Learning Postprocessing

Due to the higher complexity of the employed software library, the postprocessing is performed using the tool already implemented within `pyrlprob`[1].

[1] https://github.com/LorenzoFederici/pyrlprob

The `Ray-RLlib`[2] implementation of the PPO algorithm works by evaluating a set of full episodes selected from the training dataset after each training iteration. The performances of the full evaluation are then logged and the above mentioned tool selects the iteration that maximize or minimize a selected metric in this evaluation stage.
The actual postprocessing is then performed by selecting two checkpoints:

1. Checkpoint that maximizes the mean reward value;
2. Checkpoint that minimizes the maximum distance from the target at the EoE.

Once these two checkpoints are available, they both are evaluated on a validation dataset of 500 unseen uncertainty cases on the 3DOF environment. The best is then selected to be used on the final 6DOF environment.

[2]https://docs.ray.io/en/latest/rllib/index.html

6 Results and Comparison

This chapter presents the results of this study. The first section details the structure of the experiments performed for both RL and GP techniques, taking into account every phase: from the creation of the training dataset to the final testing on the 6DOF environment. The second section illustrates the performances of the baseline guidance strategy internally developed by DLR for the actual mission. This represents the benchmark against which the ML techniques are compared and validated. The chapter goes on by providing the details of the experiments performed with GP and RL respectively in section 6.3 and section 6.4. These sections include both the 3DOF and 6DOF performances. The consistency between the behavior of the guidance laws in the Python and Simulink environment has been properly verified before proceeding with the computations, this process has been described in Appendix C. Finally, the chapter ends with a comparison between the GP and RL based guidance laws (section 6.5), which are further compared with the baseline algorithm to provide a comprehensive overview.

6.1. Experiment Description

In the present dissertation the combination of training with a given algorithm and corresponding testing on the 6DOF is called **experiment**. The pipeline is general and it is applied identically for both RL and GP experiments.
Each experiment is structured as reported in Figure 6.1 and consists in 4 main phases:

1. Generation of the training dataset;
2. Execution on the training;
3. Postprocessing of the training results;
4. Testing on the 6DOF simulator.

The first step consists in generating the training dataset according to the methodologies explained in section 5.2. A subset of the initial 500 elements dataset is selected, with the objective of reducing the computational times of the training which otherwise would have

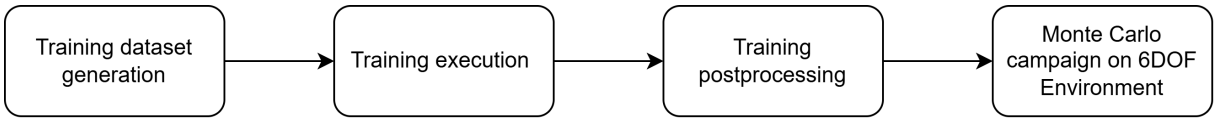


Figure 6.1: Experiment pipeline

been unacceptable.

Once the dataset is produced using the algorithm in subsection 5.2.2, the training is performed using in the 3DOF environment detailed in section 4.3. Its lower complexity results in a much smaller evaluation time (approximately 3 orders of magnitude faster if compared to the 6DOF environment) which makes the training actually feasible in reasonable time. In both GP and RL, the training is terminated when a predetermined number of iterations is reached.

A first assessment of the trained agent's performance is done on the same 3DOF environment used for training but considering a different dataset. This phase is necessary in order to select, between the several checkpoints saved during the training, which iteration has the best generalization capabilities and it is more likely to perform better on the final 6DOF environment. For instance, the last training iteration almost never results to be the best performing one on the latter evaluation dataset due to the overfitting phenomenon. This process has been described in section 5.4 and slightly differs between RL and GP.

Finally, the selected checkpoint (i.e. the GP guidance law or the NN at the selected training iteration) is exported and tested in the 6DOF Simulink environment. This latter testing phase is arguably the most important of the full experiment itself because it is representative of the performances of the guidance law on a realistic environment. Moreover this allows to compare the ML guidance laws to the baseline guidance strategy which relies on the optimal control formulation. During the study, several experiments have been produced both with the GP and RL, then for each of them a Monte Carlo campaign of 100 runs is performed on the 6DOF simulator. This number has been chosen as a trade off between the evaluation time of the testing simulator, and the statistical significance. However, the best experiment with each of the two methodologies has been tested on a Monte Carlo campaign of 500 runs; the same number is used to quantify the performances of the baseline guidance law as well. Notably, this amount of runs is comparable to the one used by DLR to asses the guidance performances for the Critical Design Review of the mission itself. Notice that the 100 runs dataset is a subset of the 500 runs one, since the latter is constructed by adding 400 uncertainty cases to the former.

In general, within the framework of this study, a given run is considered successful if the

vehicle manages to reach the EoE within an hypercube defined by altitude, longitude, latitude, velocity and heading angle, with respect to the GEOC reference frame. According to the initial condition, two possible target states are defined for the mission [64] and the same tolerance is applied. These are detailed in Table 6.1.

State Vector Component GEOC reference frame	**Target 1 (primary)**	**Target 2 (secondary)**	**Absolute Tolerance**
$h\ [m]$	1513	1513	1000
$\lambda\,[rad]$	2.3180	2.3240	$1.3 \cdot 10^{-3}$
$\phi\,[rad]$	-0.5090	-0.5049	$1.2 \cdot 10^{-3}$
$v\,[m/s]$	190	190	50
$\chi\,[rad]$	3.1108	3.1108	0.35

Table 6.1: Target states and absolute tolerance

6.2. Baseline Performances

One of the most important features of the present study is the comparison with the baseline guidance strategy on the same 6DOF validation simulator (i.e. the one described in section 4.4). To assess the baseline performances, a Monte Carlo campaign composed of 500 runs has been conducted and leading to the results in Table 6.2 .

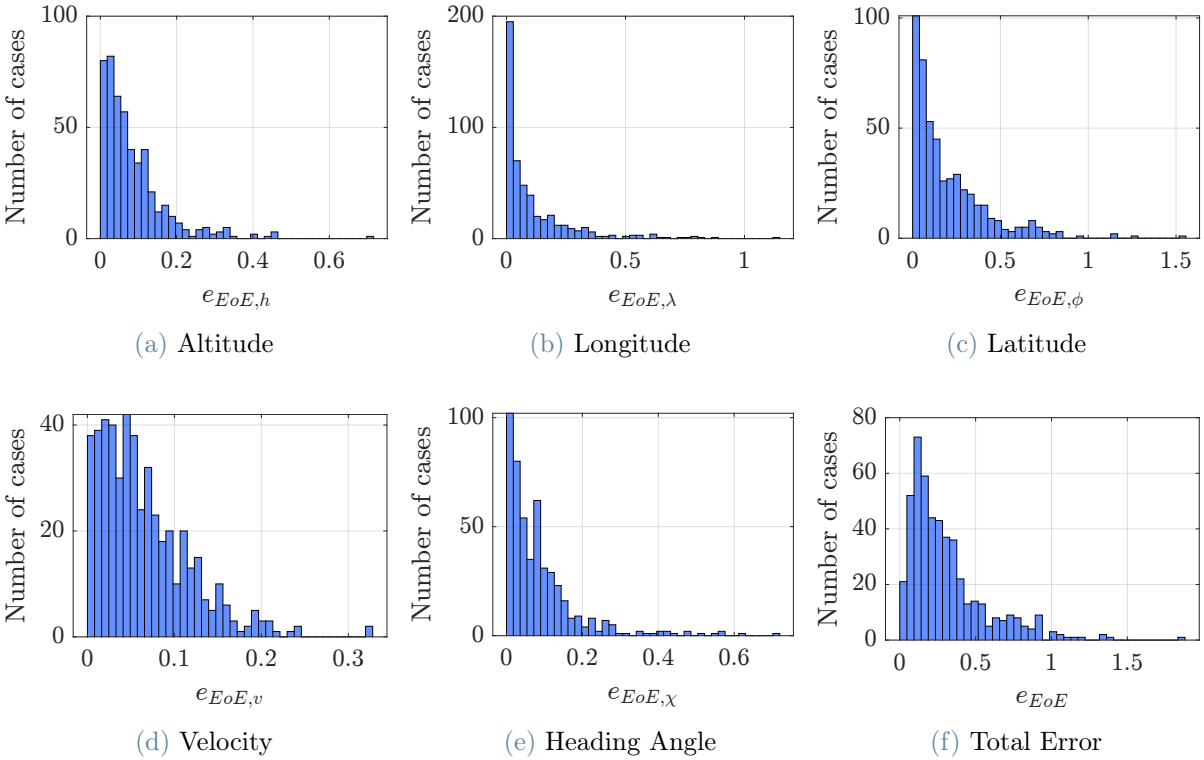
Dataset size	**Success Rate (%)**	**Failure Rate (%)**	**Instability Rate (%)**
100 Runs	97	2	1
500 Runs	97.8	1	1.2

Table 6.2: Baseline Guidance Performances

To properly interpret the results discussed in this chapter, it is important to know that for the sake of the mission outcome, an instability case is worse than a failure since the former implies the complete loss of the vehicle, while the latter only affects the proper achievement of the terminal conditions as described in Table 6.1. However, even if the target is not met, the vehicle managed to fly the complete trajectory from the EI to the EoE, satisfying the mission's scientific goals.

To better characterize the conditions at the EoE, the state error at the has been computed according Equation 5.5 and Equation 5.5 retrieving both the relative error component-

wise and the norm of the vector composed by the relative error components (i.e. the total error). A run is considered failed in meeting the target state if, at the EoE, any state vector component has relative error greater than 1. Figure 6.2 shows the distribution both the state error components and the norm of it.



(a) Altitude (b) Longitude (c) Latitude

(d) Velocity (e) Heading Angle (f) Total Error

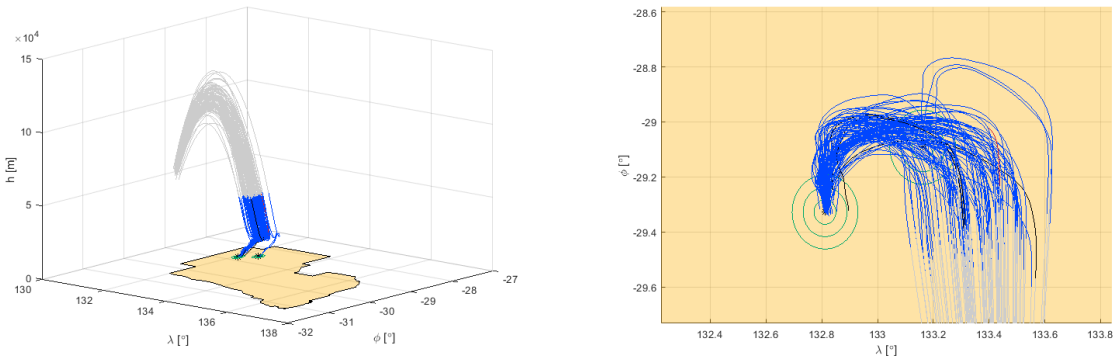
Figure 6.2: ReFEx states error at the EoE

It was observed that failed cases manage to reach the EoE while being still close to the target state. Moreover, the failed runs are outside the target only on the longitude and latitude coordinates.

In Figure 6.3 it is shown the full trajectory (in the Geodetic (GEOD) reference frame) from the separation the the EoE. The arc highlighted in blue represents the part of trajectory considered in this study (i.e. from EI to ac EoE), in particular the following color code is used according to the run outcome:

- Blue: successful run;
- Black: failed run;
- Red: unstable run. Unstable runs are interrupted when the vehicle angular rate goes beyond a certain threshold, once this condition is met, the propagation is stopped.

Only the first 100 runs are depicted, for the sake of the readability of the plot itself. The complete simulation results of the 500 runs 6DOF Montecarlo campaign are reported in section D.1.



(a) Full Trajectory

(b) Zoom From EI to EoE

Figure 6.3: 6DOF trajectories with the baseline guidance strategy, 100 runs, GEOD reference frame

6.3. Genetic Programming Experiments

This section details the settings and results obtained using the IGP algorithm described in section 3.6. The IGP evolution is driven by the genetic operations described in section 3.4 which, in turn, are driven by some random processes. In particular, the individuals are randomly generated at the beginning of the evolutionary process, and they are randomly selected to take part in the genetic operations (i.e. selection, mutation and crossover). Therefore, to assess the actual properties of the IGP algorithm, it is necessary to perform multiple evolutions with the same settings. The initial seed of the RNG gets changed between different evolutions, making it possible to assess the statistical properties of the IGP algorithm.

This is a major difference with respect to the RL, where the stochasticity has a minor impact on the results, allowing to test different setups, without the need of repeating multiple trainings with the same one.

6.3.1. Algorithm Settings

The individuals are trained on a dataset composed by the 25 uncertainty cases provided in Figure 5.2, while table Table 6.3 contains the specific parameters used to set the IGP algorithm. The evolutionary strategy used is the $\mu+\lambda$ in which μ is equal to the population

size and λ is equal to 1.3μ.
The population is initialized using the ramped half-and-half methodology. To increase as much as possible the initial diversity, 100 populations are generated and the one with the highest entropy is used as initial population.

Parameter	Value
Population Size	350
Generations Number	250
Mutation Probability Range	[0.8, 0.3]
Crossover Probability Range	[0.2, 0.7]
Mutation/Crossover Probability Step	0.01
Number of niches	10
Hall of Fame size	70
Individual Max Size	50
Individual Max Depth	40
Ephemeral constants number	3
Ephemeral constants max value	10
Primitive Set	+, -, ×, sin, cos, tanh *add3*, *plog*, *pexp*, *psqrt*

Table 6.3: IGP parameters

The crossover probability is increased by 0.01 while the mutation probability is reduced by the same value at each iteration in which the penalty is 0. Once the crossover probability reaches the value of 0.7 and the mutation probability reaches the value of 0.3 the update process is interrupted and they are kept constant until the end of the evolution.

In Table 6.3 the function *add3* in the "Primitive Set" row is defined as $add3(x, y, z) = x + y + z$ while the functions *plog*, *pexp* and *psqrt* are defined as protected versions of their natural counterparts log, exp and *sqrt*. The protected versions have been used instead then the nominal ones to prevent the generation of numerical errors since during the evolution there is no control on the inputs provided to each function, they are defined as follows:

$$plog\,(x) = \begin{cases} \log\,(x) & \text{if} \quad x > 0 \\ 10^{-300} & \text{if} \quad x \leq 0 \end{cases} \tag{6.1}$$

$$pexp\,(x) = \begin{cases} e^{x} & \text{if} \quad -100 < x < 100 \\ e^{100} & \text{if} \quad x \geq 100 \\ e^{-100} & \text{if} \quad x \leq -100 \end{cases} \tag{6.2}$$

$$psqrt\,(x) = \begin{cases} \sqrt{x} & \text{if} \quad x > 0 \\ 0 & \text{if} \quad x \leq 0 \end{cases} \tag{6.3}$$

Moreover, to improve the performance during test, the tolerances on the target state related to latitude and longitude considered during training are the 80% of the ones reported in Table 6.1. This reducing in the target area aims to train a model capable of solving a more difficult problem with respect to actual one. Such settings have been used to perform 15 full evolutions, the best individual of each of them has then been tested on the 6DOF simulator.

Finally, Table 6.4 contains the settings related to the algorithm described in section 5.2 necessary to generate the corresponding dataset and Table 6.5 contains the parameters related to the fitness and penalty functions described in subsection 5.3.1.

Parameter	**Value**
n_{uc}	25
n_e	20
e_{min}	1
$\mathbf{w}$	[0.436, 0.436, 0.436, 0, 0, 0, 0.655]

Table 6.4: GP training database generation settings

Parameter	**Value**
h_{ref}	1000 m
α_{ref}	20 deg
μ_{ref}	20 deg
t_{ref}	1 s
K_{F_0}	5
w_I	0.1
w_O	1
t_{GO}	90 s
w_{trj}	10 if the failure ratio is 0 1 otherwise
w_{corr}	1

Table 6.5: GP fitness and penalty functions parameters

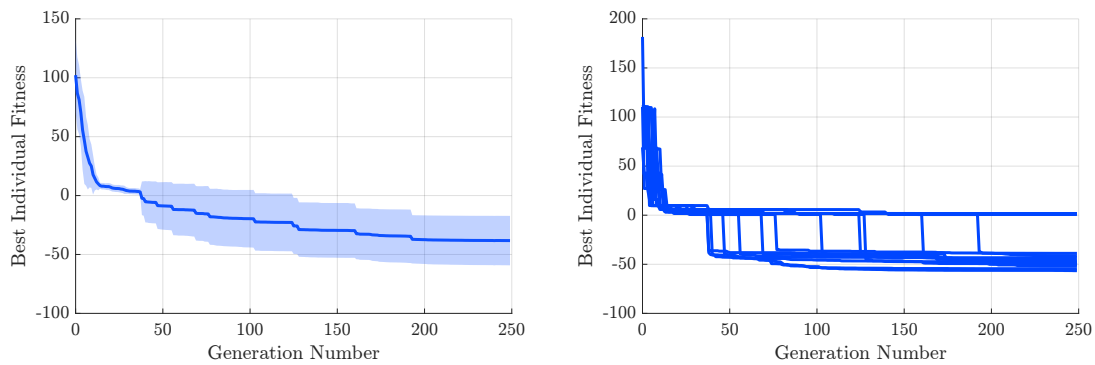
The IGP algorithm[1] has been implemented using the framework offered by the state-of-the-art open-source Python library DEAP [18] which constitutes a reliable environment

[1]https://github.com/Framarc93/GeneticProgrammingAlgorithms

for customized GP applications.

6.3.2. Training Performances

As previously mentioned, 15 full evolutions have been performed to gain an insight on the performance of the algorithm. Figure 6.4 shows the fitness value of the best individual at each generation in the 15 experiments. In particular, Figure 6.4a contains the average fitness value at each generation (solid line) together with the standard deviation to it associated (shaded area) whereas in Figure 6.4b the fitness values associated to the best individual et each generation in each single evolution are reported. Notice that a below zero fitness value means that each of the 25 uncertainty cases is successfully solved during the training.



(a) Fitness value mean and standard deviation (b) Fitness value of each evolution

Figure 6.4: Fitness value of the best individual at each generation

By comparing Figure 6.4a with Figure 6.4b it is possible to understand the reason behind the quick increasing in the standard deviation around the 40th generation. This is due to the nature of the fitness function which changes sign while producing a discontinuity. Moreover, the few runs which do not manage to solve all the uncertainty cases shown in 6.4b contribute to skew the distribution of the fitness values which is not actually centered on its mean. Table 6.7 lists the results of the 15 evolutionary processes performed on the training 3DOF environment, evolutions labeled with (*) cannot manage to successfully solve each of the uncertainty cases during the training.
Performances are divided according to which test dataset has been used (i.e. the 100 or 500 runs) since these datasets are the same that will be employed in the 6DOF validation. Finally the selection of the individual to be used in the 100 runs 6DOF Monte Carlo campaign has been done according to the postprocessing methodology described in

section 5.4.

Evolution Number	100 runs success rate (%)	500 runs success rate (%)
EVOL 1	98	98.6
EVOL 2 (*)	95	95.6
EVOL 3	96	96.0
EVOL 4 (**)	97	98.6
EVOL 5	98	98.4
EVOL 6	97	98.2
EVOL 7	99	99.2
EVOL 8	99	99.2
EVOL 9	98	98.2
EVOL 10 (*)	94	95.0
EVOL 11	96	97.8
EVOL 12	96	96.6
EVOL 13 (**)	99	99.0
EVOL 14 (*)	97	97.4
EVOL 15	97	97.4
Median	97	98.2
Standard Deviation	1.436	1.285

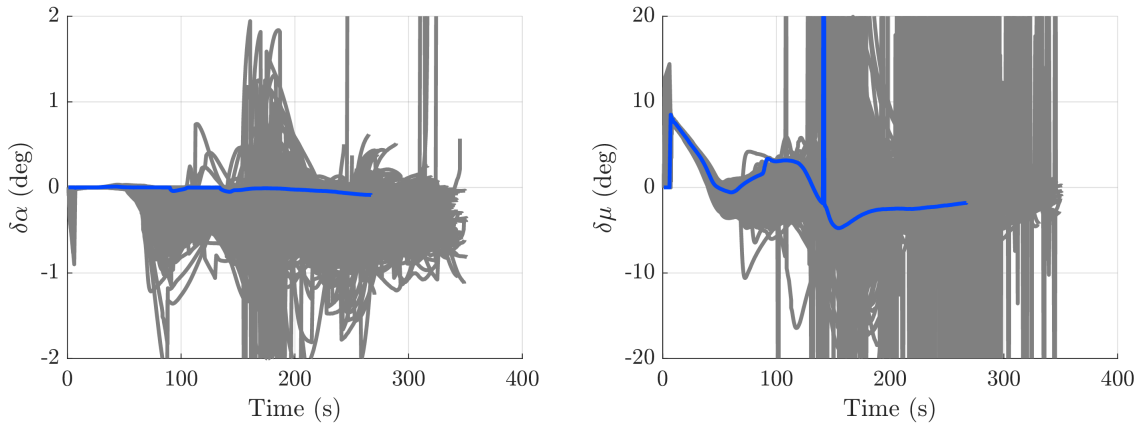
Table 6.7: IGP performances on the 3DOF training environment.
(*): evolutions which terminate without solving each uncertainty case
(**): selected for the 500 runs 6DOF Monte Carlo campaign

As it will be described in detail in the next sections, the two evolutions marked with (**) outperforms all the others in the 6DOF framework.
For the sake of the readability of the chapter, only the plots related to **EVOL 4** and **EVOL 13** will be here shown.

Figures 6.5 and 6.6 depicts the corrective commands issued by the GP guidance laws in **EVOL 4** and **EVOL 13** respectively. From these plots, it can be observed how the GP manages to produce a smooth guidance command signal without the need of penalizing the derivative of the command itself as it is done on the RL.

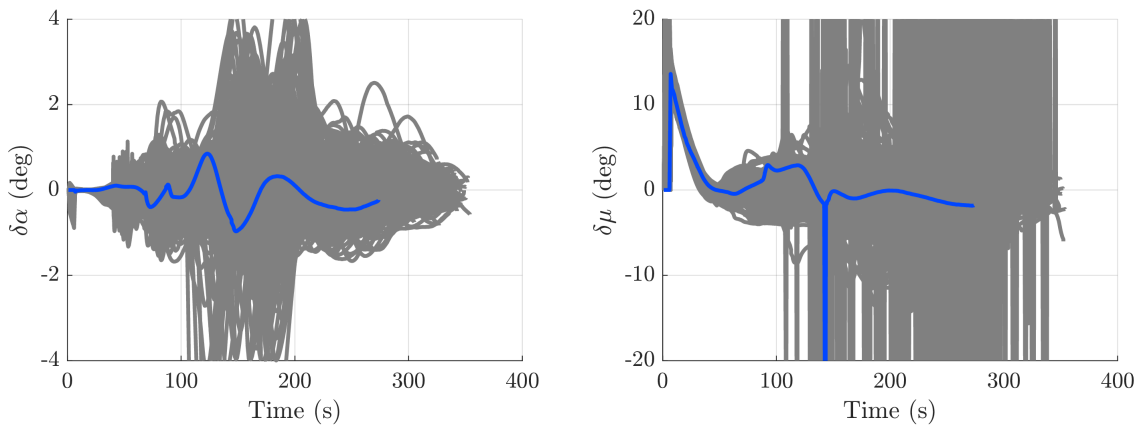
This results is due to two reasons:

1. The GP model is a relatively small (if compared to the network size) scale symbolic function which is defined by combining other elementary smooth functions as shown



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.5: **EVOL 4**, Corrective commands, 3DOF environment - in blue an example run



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

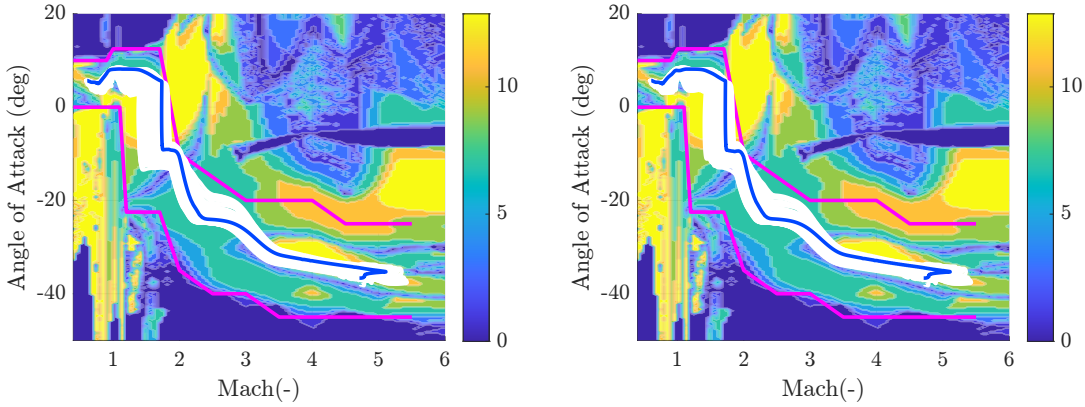
Figure 6.6: **EVOL 13**, Corrective commands, 3DOF environment - in blue an example run

in subsection 6.3.1;

2. The GP algorithm can dynamically modify the structure of the output function during the evolution. It can therefore apply for example a tanh function in one of the uppermost nodes (i.e. close to the root one) with a smoothing effect on the final output.

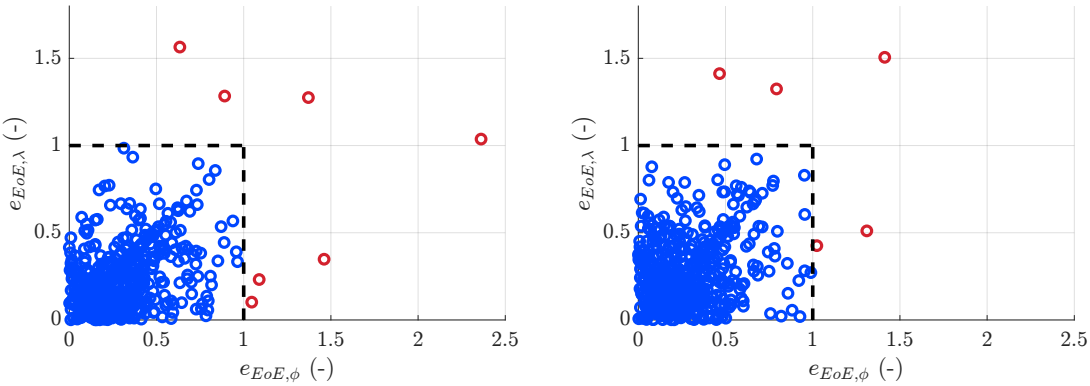
It might happen that during the propagation the function generates some asymptotes, as can be clearly seen in Figure 6.5b and 6.6b, but these asymptotes acts as spikes in time and are filtered by the the control logic which is applied to the output of the guidance law described in subsubsection 4.3.4 and represented by line 15 in Algorithm 4.1.

In Figure 6.7 it is shown the trajectory with respect to the flyable region. The GP guidance law manages to maintain the vehicle away from the highly unstable regions of the Mach-α plane while remaining within the limits of the aerodynamic database boundaries.
Figure 6.8 shows the longitude and latitude relative errors at the EoE which are computed according to Equation 5.4. The simulations going to both targets are considered.



(a) **EVOL 4** (b) **EVOL 13**

Figure 6.7: Flyable region, 3DOF environment - in blue an example run



(a) **EVOL 4** (b) **EVOL 13**

Figure 6.8: Longitude and Latitude relative errors at the EoE, 3DOF environment

6.3.3. Testing Performances

The performances in the 6DOF testing environment are now reported in Table 6.8, the table accounts for success rate and instability rate since the latter is the worst failure mode resulting in the complete loss of the vehicle.

Evolution Number	100 runs dataset		500 runs dataset	
	Success (%)	Instability (%)	Success (%)	Instability (%)
EVOL 1	95	1		
EVOL 2 (*)	94	1		
EVOL 3	94	4		
EVOL 4 (**)	96	0	96.6	1.2
EVOL 5	96	1		
EVOL 6	93	4		
EVOL 7	96	2		
EVOL 8	97	2		
EVOL 9	95	2		
EVOL 10 (*)	84	3		
EVOL 11	96	0		
EVOL 12	94	3		
EVOL 13 (**)	97	1	96.2	1.2
EVOL 14 (*)	94	3		
EVOL 15	91	6		
Median	95	2	N/A	
Standard Deviation	3.23	1.66		

Table 6.8: IGP performances on the testing environment.
(*): evolutions which terminate without solving each uncertainty case
(**): selected for the 500 runs 6DOF Monte Carlo campaign

These results showcase that on average the IGP provides acceptable performances, yet inferior with respect to the baseline guidance strategy (see Table 6.2) while in the best cases (Evolution 4 and Evolution 13), the IGP performances are comparable with the baseline one in particular regarding the instability rate, which is identical to the one obtained with the baseline guidance on the 500 runs dataset. Since the subdivision in successes, failures and instabilities is discrete, the full probability density function associated to the condition at the EoE has been computed and is reported in Figure 6.9. The baseline guidance performances have been also included to offer a better comparison.

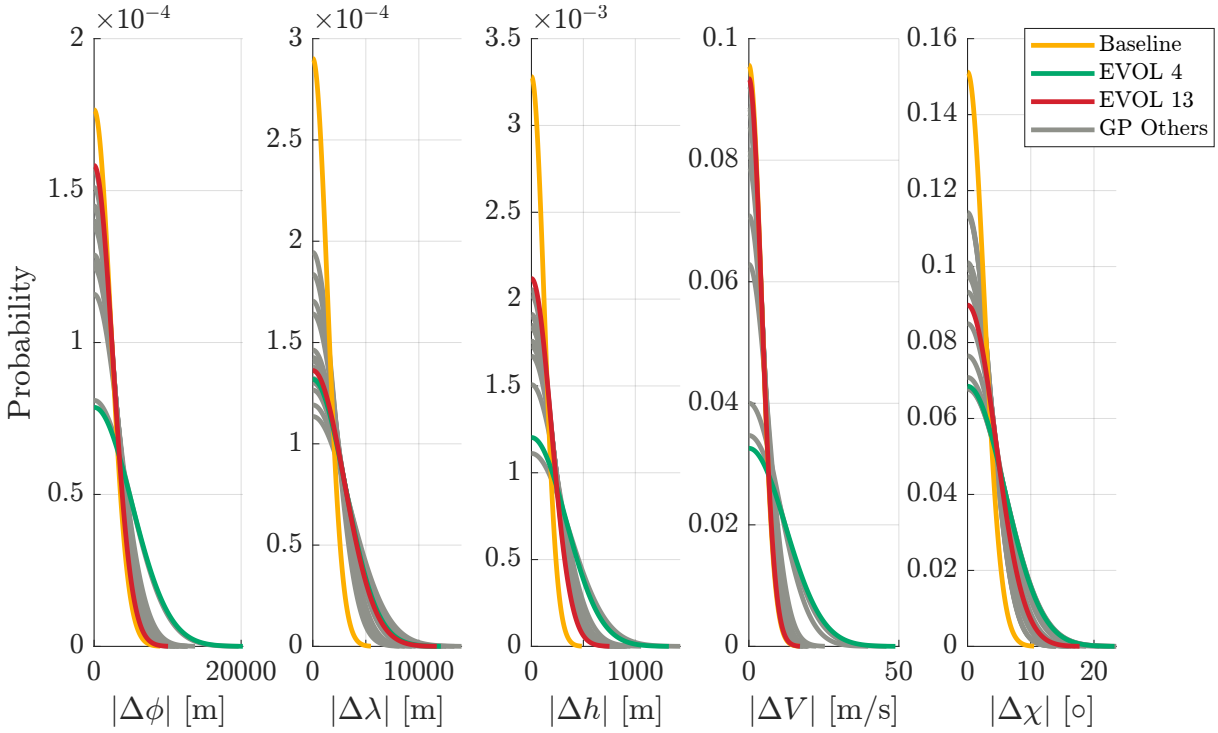
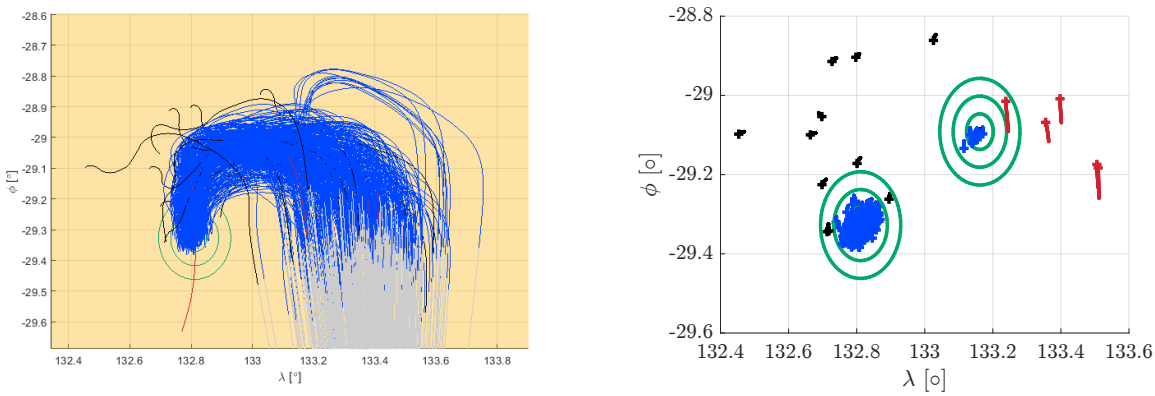


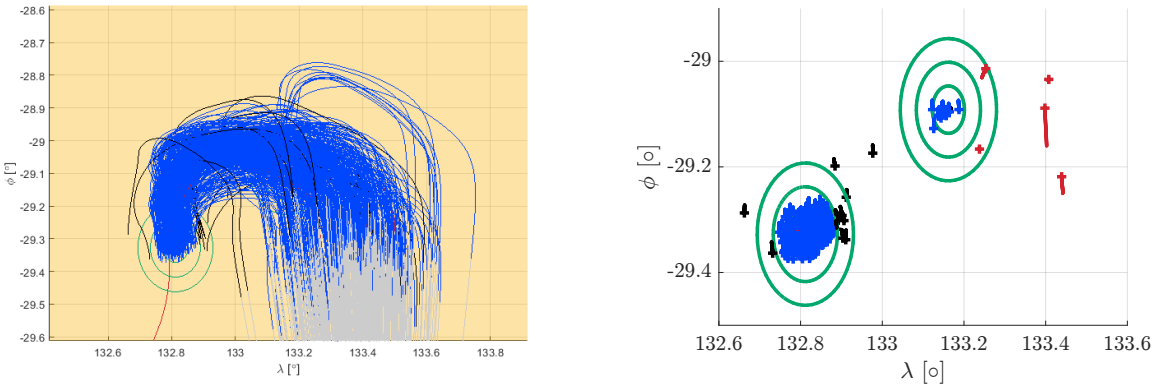
Figure 6.9: Dispersion at the EoE, IGP, 6DOF environment

It is possible to notice how even if the **EVOL 4**, according to Table 6.8, has better performances than **EVOL 13**, when it comes to the distributions of the errors, it underperform with respect to the latter. This is related to the fact that the failed cases in **EVOL 4** miss the target by a large distance, while **EVOL 13** always manages to stay in proximity of it. This phenomenon is confirmed by the plot of the actual simulations in Figure 6.10 and Figure 6.11.



(a) Flight trajectories (b) EoE dispersion

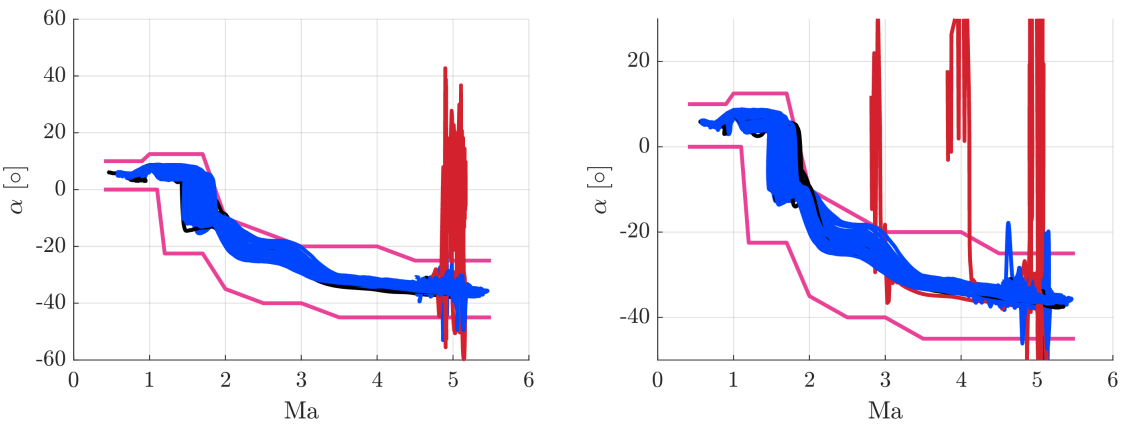
Figure 6.10: **EVOL 4**, 6DOF trajectories, 500 runs, GEOD reference frame



(a) Flight trajectories (b) EoE dispersion

Figure 6.11: **EVOL 13**, 6DOF trajectories, 500 runs, GEOD reference frame

The corrective command signals in the testing environments are reported in Figure 6.13 and Figure 6.14 confirming the GP increased signal smoothness with respect to the RL when the term promoting the smoothness of the guidance function is excluded from the reward formulation, as will be shown in Figure 6.18. Finally, the conditions about the flyable region are satisfied in the 6DOF simulations too (see Figure 6.12), which ensures the accuracy of all the previous results since each run is performed using reliable aerodynamics data.

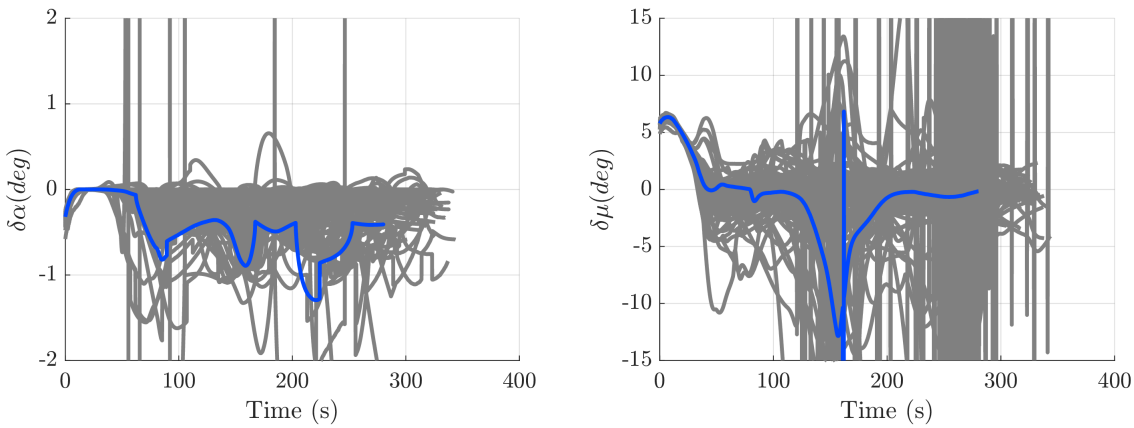


(a) **EVOL 4** (b) **EVOL 13**

Figure 6.12: Flyable region, 6DOF trajectories, 500 runs

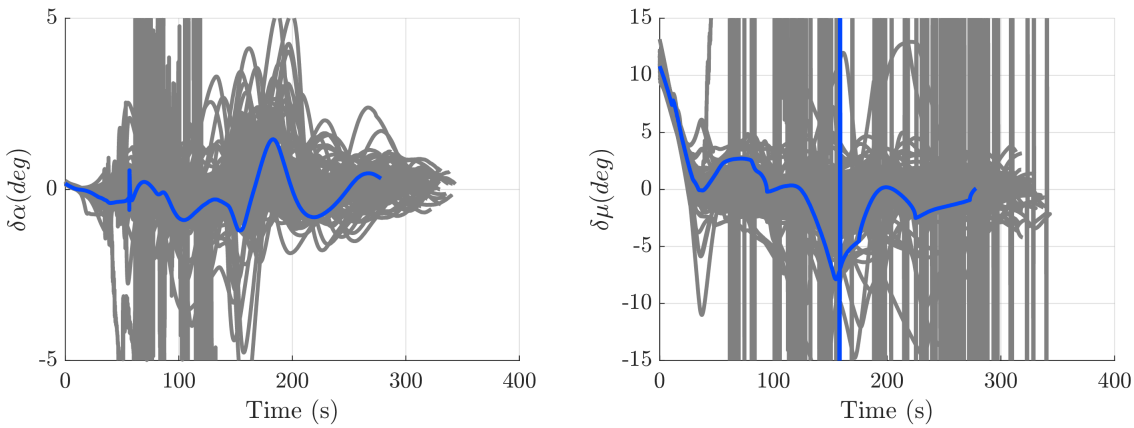
In Figure 6.12 red lines represents unstable runs and it is expected that they will leave the flyable region since the propagation is continued until a predetermined threshold on the angular rate is reached. Being this value much higher than the nominal value, is is expected that the instability will be seen in that way from a simulation point of view. The

complete simulation results of the 500 runs 6DOF Montecarlo campaign for the **EVOL 13** experiment are reported in section D.3.



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.13: **EVOL 4**, Corrective commands, 6DOF environment - in blue an example run



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.14: **EVOL 13**, Corrective commands, 6DOF environment - in blue an example run

6.3.4. Guidance Law Size

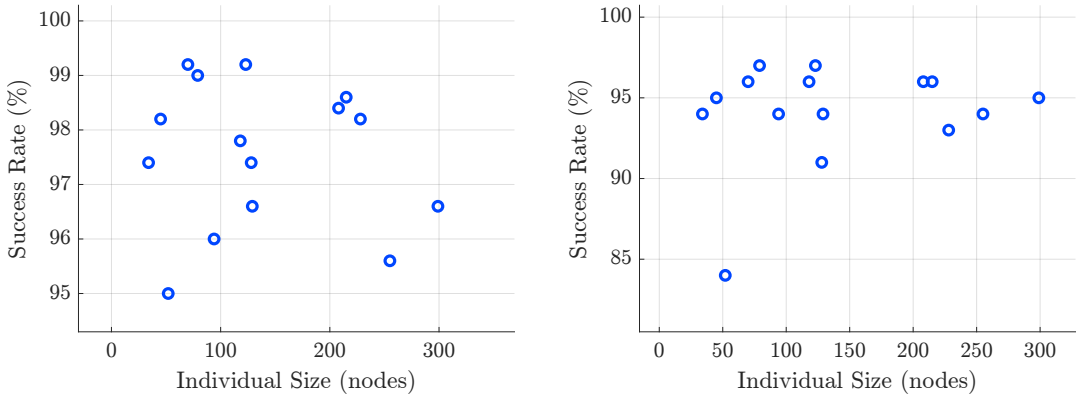
Each GP individual produces two guidance laws to compute respectively the $\delta\alpha$ and $\delta\mu$ commands. The size of these two laws, intended as number of nodes, is reported in Table 6.9 which distinguishes between size of the two guidance laws separately ($N_{\delta\alpha}$ and $N_{\delta\mu}$) and the sum of the sizes of both ($N_{\delta\alpha+\delta\mu}$). Figure 6.15 compares the total number of nodes in each tested guidance law with its performances in both the training and testing

simulator, no evident correlation is noticeable since laws with very different size can lead to similar performances and vice versa.

Evolution number	1	2	3	4	5	6	7	8
$N_{\delta\alpha}$ (nodes)	139	155	52	55	117	147	7	63
$N_{\delta\mu}$ (nodes)	160	100	42	160	91	81	63	60
$N_{\delta\alpha+\delta\mu}$ (nodes)	299	255	94	215	208	228	70	123

Evolution number	9	10	11	12	13	14	15	-
$N_{\delta\alpha}$ (nodes)	28	33	48	115	48	10	72	-
$N_{\delta\mu}$ (nodes)	17	19	70	14	31	24	56	-
$N_{\delta\alpha+\delta\mu}$ (nodes)	45	52	118	129	79	34	128	-

Table 6.9: Comparison between GP-law number of nodes and guidance performances



(a) 3DOF simulations results (b) 6DOF simulations results

Figure 6.15: Individual size with respect to flight performances

6.4. Reinforcement Learning Experiments

Differently from GP, RL is not affected by stochastic behaviors. Even if there are some probabilistic operations within the full training algorithm, in particular the NN weights and biases initialization, these operations never have a first order influence on the training direction. Moreover, the iterative improvement applied to the NN, which parameters are optimized during the training, concurs to strongly mitigate the effect of the stochastic operations. For these reasons, stochastic effects have been considered negligible and were not analyzed as it was done for the GP.

6.4.1. Algorithm settings

Five experiments have been performed to gain an insight on the RL performances and how it compares to the GP and baseline guidance. Three parameters have been considered variable between different experiments:

- Size of the training dataset: the label *xxUC* refers to the number of uncertainty cases in the training dataset;
- Reward signal formulation, with the objective of testing the effect of the smoothing term;
- Size of the NN: a standard and a larger NN have been tested, in the following table, each network is assumed to have 13 input and 2 output nodes while the number of nodes of each hidden layer is written in the corresponding column of Table 6.10. Both the actor and the critic network have been designed to have the same size.

The parameters of the selected experiments have been reported in Table 6.10 that contains only the varying ones, while all the others are kept constant. Table 6.11 and Table 6.12 contain the other fixed tuning hyperparameters referred to the RL algorithm in the first case, and to the reward signal described in subsection 5.3.2 in the second case. Their numerical value has been chosen according to common practice rules or via iterative empirical tuning. As done for the GP, Table 6.13 contains the parameters used to configure the dataset generation algorithm described in section 5.2.

Name	**Training Dataset Size**	**Reward Signal**	**Neural Network Size**	**PPO iterations**
60UC Standard	60 uncertainty cases	Standard, non smooth	256, 128, 128	8000
25UC Smooth	25 uncertainty cases	Smooth	256, 128, 128	8000
60UC Smooth	60 uncertainty cases	Smooth	256, 128, 128	8000
500UC Smooth	500 uncertainty cases	Smooth	256, 128, 128	8000
500UC Smooth Large	500 uncertainty cases	Smooth	400, 200, 150	10000

Table 6.10: RL experiments variable parameters

Parameter	Value
Training Iterations	8000/10000
Batch Size	10240
SGD minibatch size	128
SGD iterations	15
Number of workers	10
Environments per worker	8
Critic network size	see Table 6.10
Actor network size	see Table 6.10
Activation function	tanh
Discount factor	0.99
Clip parameter	0.01

Table 6.11: RL training fixed parameters

Parameter	Value
w_h	2000 m
w_λ	0.01 rad
w_ϕ	0.01 rad
k_{trj}	15
$k_{\dot{\alpha}}$	100
$k_{\dot{\mu}}$	100
k_{corr}	900
$k_{saturation}$	1.5
k_{log}	$\frac{1000}{6}$
k_{quad}	$\frac{500}{6}$
$\delta\alpha$	$[-10°,\ +10°]$
$\delta\mu$	$[-10°,\ +10°]$

Table 6.12: RL reward signal parameters

Parameter	Value
n_{uc}	60
n_e	400
e_{min}	0
$\mathbf{w}$	$[0.097,\ 0.097,\ 0.097,\ 0.097,\ 0.097,\ 0.097,\ 0.971]$

Table 6.13: RL training database generation settings

Given that the objective of the thesis is not to develop new algorithms but to apply existing ones to the reentry guidance problem, already available tools have been used to carry out the present work. Multiple reliable libraries exists and are recognized as state-of-the art, `Ray-RLlib`[2] is chosen. It is an open industry-graded open-source library, able to interface with a custom environment to train a NN. It allows to simply parallelize the training between CPUs reducing the training times. Moreover, the open-source Python library `pyrlprob`[3] has been used because it offers a convenient interface with `Ray-RLlib` to train, evaluate and postprocess custom `Gym`-based [81] environments.

6.4.2. Experiments Results

Figure 6.16 depicts the reward signal evolution during the full training for each of the experiments described in Table 6.10. The **60UC Standard** configuration consistently

[2]https://docs.ray.io/en/latest/rllib/index.html
[3]https://github.com/LorenzoFederici/pyrlprob

outperforms the other configurations during training. This can be explained by the fact that there are no requirements on the shape of the guidance signal, therefore the agent can provide high frequency commands which are very effective on the 3DOF environment. This is confirmed by the 99.6% success rate listed in Table 6.14.

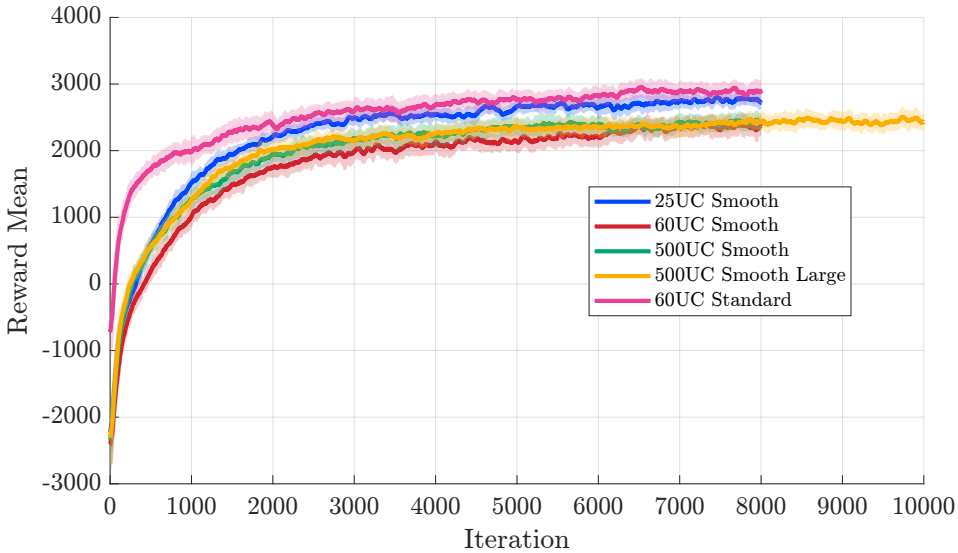
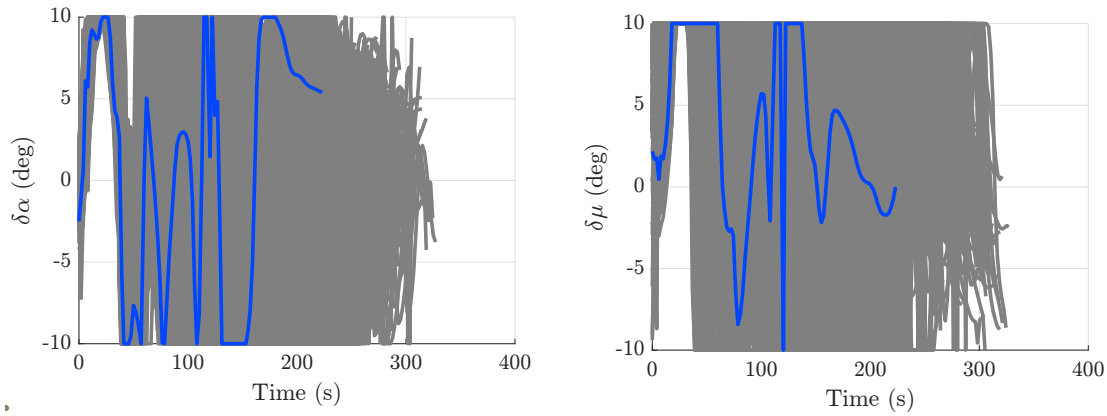


Figure 6.16: Reward value during the training

However, the produced corrective guidance command is very noisy and it generally shows a very irregular behavior (see Figure 6.17) which is problematic when the same network is tested in a more complex environment. This is a consequence of the so called action shaking problem [92] for which the agent can select very different actions associated to close states.
For instance, Figure 6.17 depicts the evolution of the corrective command during time in the 3DOF simulations.

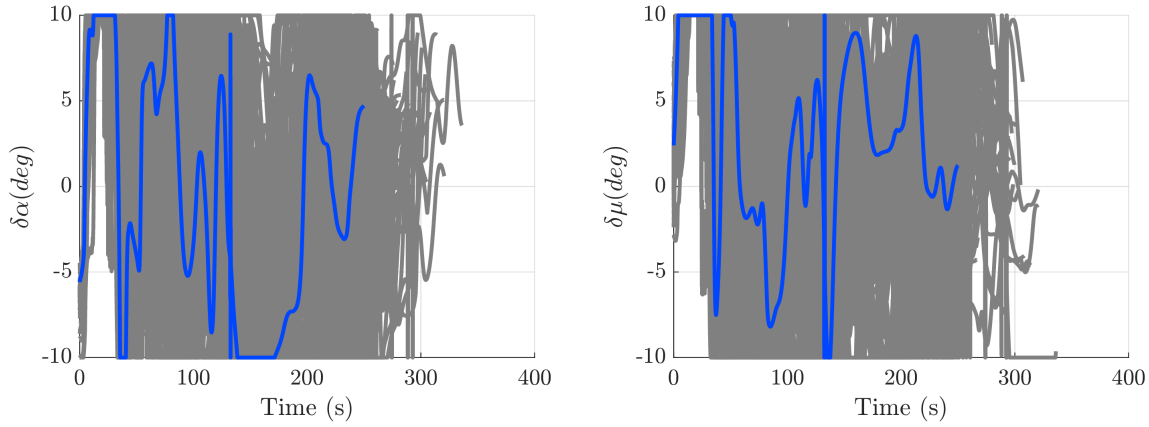
The same behavior is present also in the 6DOF environment in which, the same unstable pattern is present in the corrective guidance command as can be seen in Figure 6.18. Differently with respect the 3DOF environment, in which this command pattern was derived to maximize the performances of the agent, when it comes to the 6DOF evaluation this leads to poor performances. In particular the vehicle is unstable in the 8% of the simulations. This result is much worse than the baseline guidance algorithm one, where the unstable cases are the 1.2% (see Table 6.2).

To overcome this problem, the component shown in Equation 5.28 has been added to the complete reward expression. By penalizing the time derivative of the control action during the training, it is possible to obtain a regularization effect on the produced guidance signal. In this way, even if the performances on the training environment gets slightly



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.17: **60UC Standard**, Corrective commands, 3DOF environment - in blue an example run



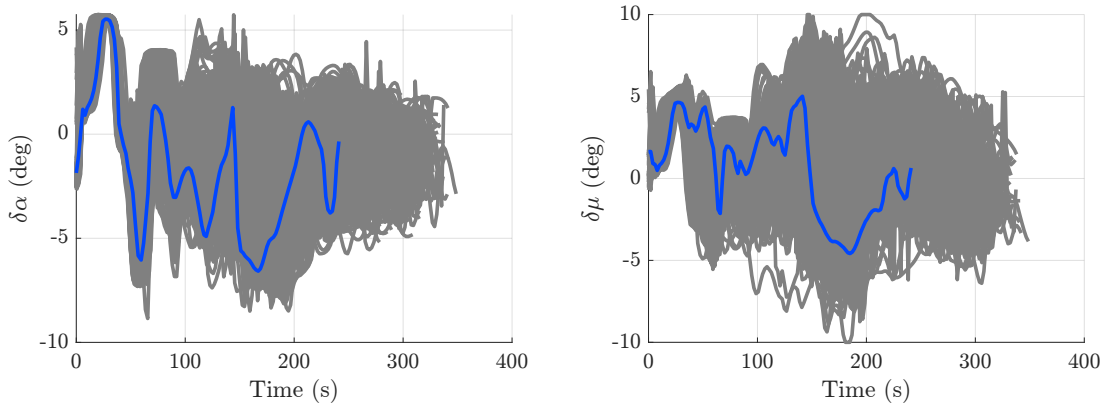
(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.18: **60UC Standard**, Corrective commands, 6DOF environment - in blue an example run

worse due to the increased amount of constraints, the produced guidance commands are smoother and more stable as depicted in Figure 6.19 and Figure 6.20 for the 3 and 6 DOF environments respectively.

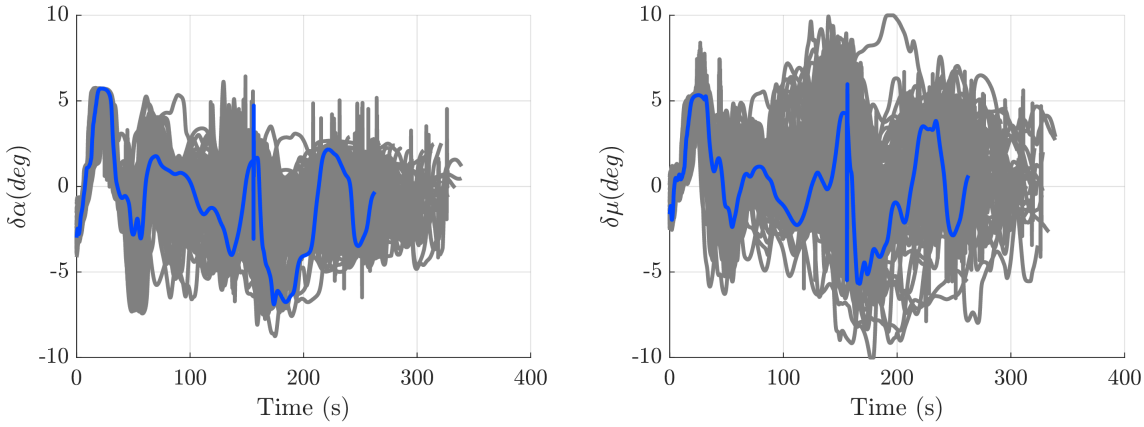
This results in a decrease of the unstable cases from 8% to 1%.

As aforementioned, five experiments have been performed: four of them using the control smoothing term formulation due to the increased performances as already proven. The objective of these tests was to analyze the effects of both the network and training dataset size on the guidance performances. Table 6.14 contains a summary of the results of the five experiments. Results are summarized reporting for the 3DOF case the percentage



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.19: **60UC Smooth**, Corrective commands, 3DOF environment - in blue an example run



(a) Angle of Attack Corrective Command (b) Bank Angle Corrective Command

Figure 6.20: **60UC Smooth**, Corrective commands, 6DOF environment - in blue an example run

of runs managing to reach the target within the tolerances (Success Rate) while in the 6DOF case both the Success Rate and the percentage of unstable runs (Instability Rate) is reported. Each of the five experiments has been tested on the 500 runs dataset within the 3DOF environment, and on the 100 runs dataset in the 6DOF one.

As can be seen from Table 6.14 the experiments implementing the smoothing term consistently outperform the case without. It is important to remember that from the guidance point of view, an instability is worse than a failure since the latter consists in failing to meet the state tolerances at the EoE while in the former case the vehicle is completely lost mid flight. Hence, the first driver to evaluate the quality of the 6DOF performances is the instability rate.

Experiment Name	3DOF environment Success Rate (%)	6DOF environment Success Rate (%)	Instability Rate (%)
60UC Standard	99.6	91	8
25UC Smooth	99	97	3
60UC Smooth	99	98	1
500UC Smooth	98.8	98	1
500UC Smooth Large	99	96	2

Table 6.14: RL experiments performances recap

Experiments discussion

Looking into the single experiment details, **25UC Smooth**, **60UC Smooth** and **500UC Smooth** where conducted to analyze the influence of the training dataset size on the guidance performance while **500UC Smooth Large** aims to investigate the effects of increasing the NN size.

Fixed the number of training iterations, the dataset size influences its performances in two ways with opposite effects:

1. The larger the dataset is, the more representative of the real scenario it is;
2. A smaller dataset allows to train the agent for more episodes on the same uncertainty case improving its performance on cases similar to those seen during the training.

Moreover, the perturbations associated to each uncertainty case have a different degree of severity with respect to the ones associated to another case making it more or less challenging for the GC subsystem to drive the mission to the target.

25UC Smooth features the smallest training dataset (Figure 5.2) selected according to the criteria in subsection 5.2.3 and allows the agent see a larger amount of times the same uncertainty cases and to optimize the guidance law performances based on them. The **500UC Smooth** configuration contrarily, includes the maximum number of uncertainty cases (Figure 5.1). It represents the theoretically optimal configuration allowing for the best representation of the environment during the training. Finally, **60UC Smooth** is a tradeoff solution between the two configurations above mentioned, featuring an intermediate amount of uncertainty cases (Figure 5.3) but they are still selected to include extreme cases as well as the baseline ones in terms of initial conditions and environmental perturbations.

25UC Smooth has the best performances in terms of reward value during the training (Figure 6.16) due to the reduced size dataset. **500UC Smooth** and **60UC Smooth** have poorer training performances. **500UC Smooth** outperforms **60UC Smooth** in the initial phase due to the higher number of UC with low severity perturbations present

in the former even if they eventually converge to comparable reward values.
When it comes to the testing, the three considered configurations perform equally on the 3DOF environment while on the 6DOF, the **25UC Smooth** underperforms with respect to the others probably due to the smaller dataset used during training, which influences the network's capability of dealing with unseen scenarios in a different environment.

The last experiment left to analyze is the **500UC Smooth Large**. This experiment has been configured with a larger NN and more training epochs as it is reported in Table 6.10. The objective of this test was to improve the performances of **500UC Smooth** by increasing the agent generalization capabilities (i.e. larger NN) as well as the training time by performing 10000 iterations instead than the usual 8000. Even if the training performance in terms of reward signal over epochs is very similar to the **500UC Smooth** one, when it comes to the testing, this configuration shows a slightly worse performance both in the 3DOF and in the 6DOF environments. This result can be explained by considering that we are dealing with a very complex problem therefore there are a lot of concurring factors that can influence the outcome of an experiment. For instance, a proper tradeoff between training iterations, network size, dataset size and all the other hyperparameters of the PPO algorithm is always very difficult and time consuming to obtain. Hence, to gain more insights on the effects of the network size, more tests should be performed increasing and decreasing its size gradually.

Figure 6.21 summarizes the results of the described experiments in terms of PDFs computed on the state errors at EoE. To obtain the curves in Figure 6.21, only the runs reaching the target have been included since unstable cases cannot be used. Moreover, the **60UC Standard** configuration has not been included being discarded due to the high number of instabilities. Regardless from successes and failures distinction, all the experiments in Figure 6.21 manage to achieve similar performances on each state, with the **60UC Smooth** and the **500UC Smooth** configurations performing slightly better than the others.

60UC Smooth and **500UC Smooth** exhibit the same performances in the 100 runs 6DOF Monte Carlo campaign outperforming the other configurations. For this reason they have been tested on the larger 500 runs Monte Carlo campaign. Table 6.15 contains the results of the above mentioned simulations, the results both in terms of percentage and absolute number of runs (# Runs) have been included.

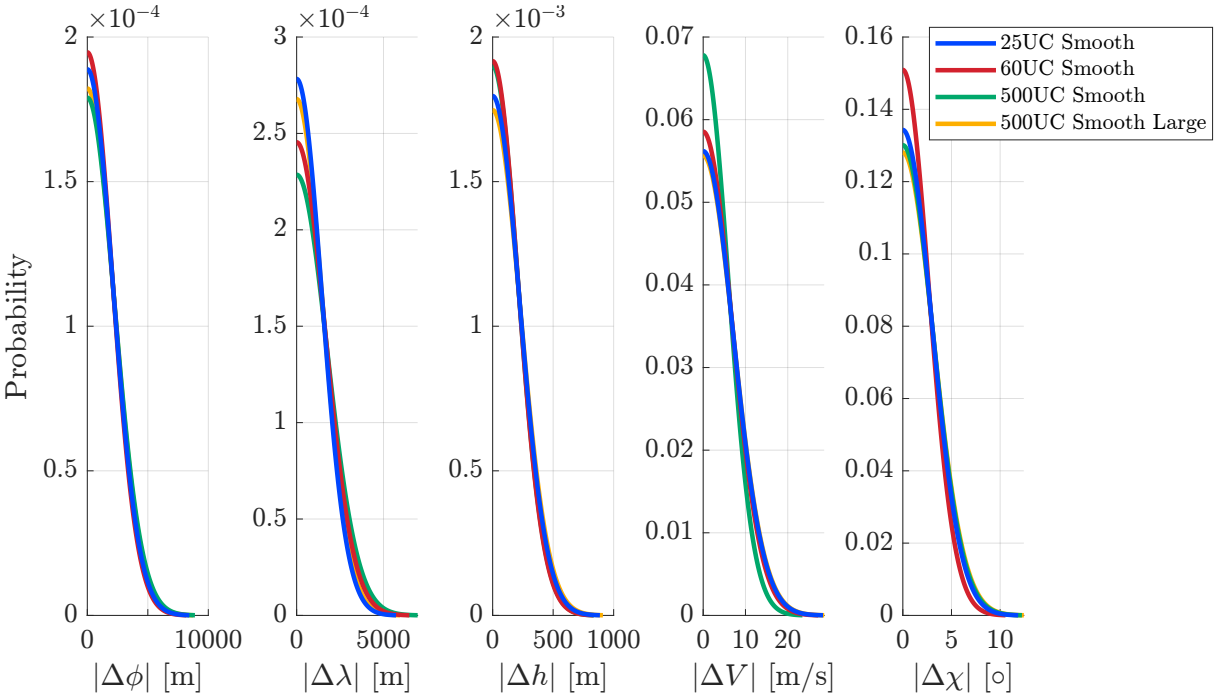


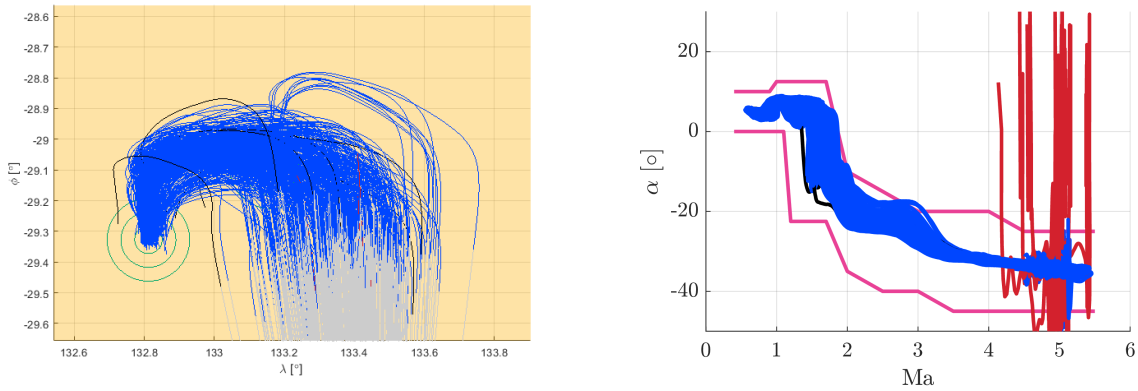
Figure 6.21: RL experiments performances in the 6DOF environment

Experiment Name	Successes		Failures		Instabilities	
	(%)	(# Runs)	(%)	(# Runs)	(%)	(# Runs)
60UC Smooth	97.4	487	1.2	6	1.4	7
500UC Smooth	97	485	1.2	6	1.8	9

Table 6.15: RL experiments performances on the 500 runs Monte Carlo campaign

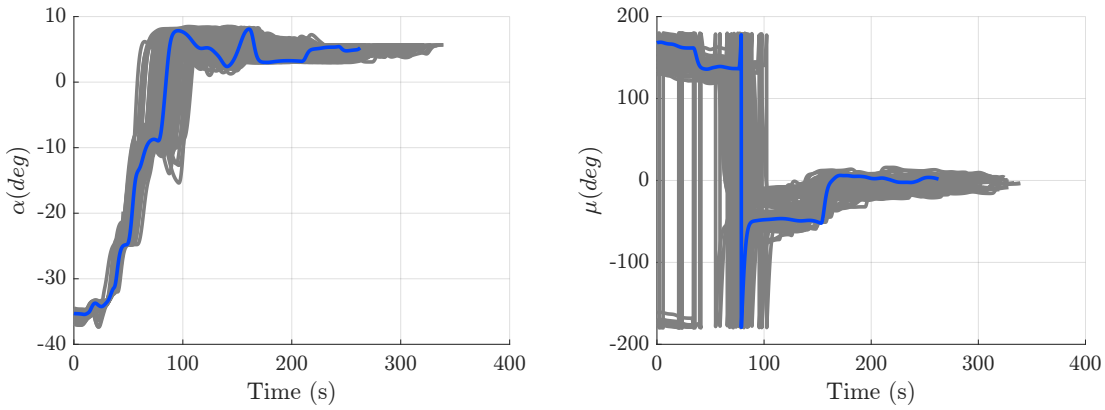
As can be seen, both solutions can deliver very similar performances in terms of instabilities and success rate. However, the **60UC Smooth** configuration shows slightly better results and therefore has been selected for the final comparison with the GP and the reference guidance.

The resulting trajectories projected in the longitude/latitude plane are reported in Figure 6.22a, while Figure 6.22b shows that the RL guidance model manages to keep the vehicle within the flyable region, i.e. where aerodynamic data are available, ensuring the quality of the simulations themselves. The control actions plots are depicted in Figure 6.20 while the full guidance signal intended as $\theta = \theta_{ref} + \delta\theta$ is reported in Figure Figure 6.23. Notice that the bank angle in Figure 6.23b is wrapped between $\pm 180°$ causing the apparent discontinuity before 100 s, in particular this happens in correspondence on the roll maneuver to pass from the belly-up to the belly-down configuration.



(a) Flight trajectories, GEOD reference frame (b) Flyable region

Figure 6.22: **60UC Smooth**, 6DOF environment, 500 runs



(a) Angle Of Attack (b) Bank Angle

Figure 6.23: **60UC Smooth**, 6DOF environment, Guidance Signal - in blue an example run

To contextualize the performances of the selected configuration, Figure 6.24 depicts the PDFs computed from the state errors at EoE obtained with the best and the other RL configurations (excluding **60UC Standard**) as well as with the reference guidance.

Figure 6.24 demonstrates that the selected RL configuration achieves overall performance comparable to the baseline guidance algorithm, with a similar error distribution across the five components of the target state vector. The complete simulation results of the 500 runs 6DOF Montecarlo campaign for the **60UC Smooth** configuration are reported in section D.2.

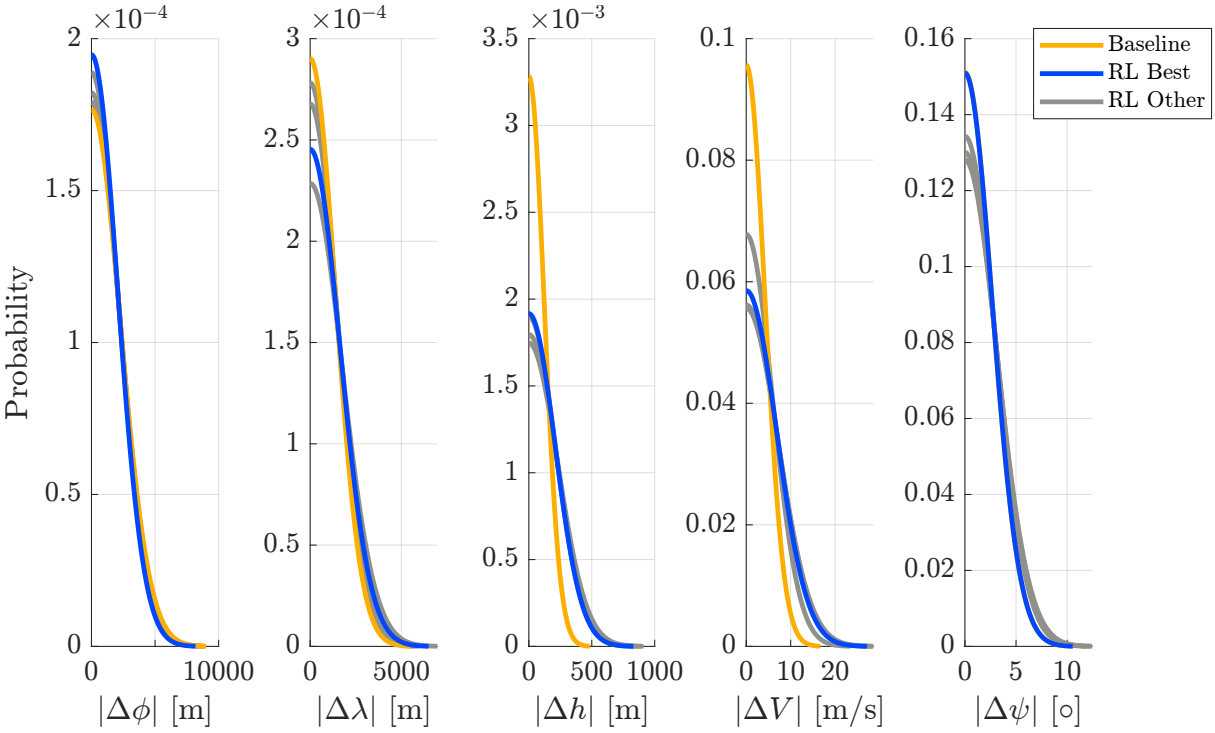


Figure 6.24: Dispersion at the EoE, RL, 6DOF environment

6.5. Comparison

In this section the performance of RL and GP approaches is compared both to each other and to the baseline algorithm. The comparison focuses first on the raw guidance performance. Subsequently, the analysis shiftes to additional parameters that are not directly related to the flight simulations, such as inference time, training time, interpretability of the model, generalization capabilities and implementation aspects.

6.5.1. Guidance Performances

The best results obtained with GP and RL are selected and compared with the baseline guidance algorithm. Concerning the GP, two evolutions have been selected for the final comparison: **EVOL 4** and **EVOL 13**. They have been chosen since are those evolution which outperforms all the others in terms of successes/instabilities in the testing environment. Both of them shows the same number of instability cases equal to the reference algorithm, however the guidance law produced by **EVOL 4** leads to worse failure cases than the one from **EVOL 13**, despite having a higher success rate. This is also visible from the PDFs in Figure 6.9 and from the flight trajectories in Figure 6.10 and Figure 6.11. For this reason both of them are considered in the final comparison with the

RL and the baseline strategy.

Regarding the RL, the **60UC Smooth** configuration outperforms all the others both in terms of success and instability rates, as can be seen in Table 6.14 and Table 6.15, and in terms of EoE errors as can be seen in Figure 6.21.

The final performances indicators are summarized in Table 6.16: the results are reported both in terms of percentage and in terms of number of runs (# Runs) with respect to the 500 total runs which have been performed. This is done to give a better idea of the proportions of the differences in performances. The comparison is completed by Figure 6.25 and Figure 6.26: the first contains the PDF of the distance from the target for each state component while the second accounts for the total relative error at the EoE.

Experiment Name	Successes		Failures		Instabilities	
	(%)	(# Runs)	(%)	(# Runs)	(%)	(# Runs)
Baseline	97.8	489	1	5	1.2	6
RL - 60UC Smooth	97.4	487	1.2	6	1.4	7
GP - EVOL 4	96.6	483	2.2	11	1.2	6
GP - EVOL 13	96.2	481	2.6	13	1.2	6

Table 6.16: Guidance performances comparison, 500 runs, 6DOF environment

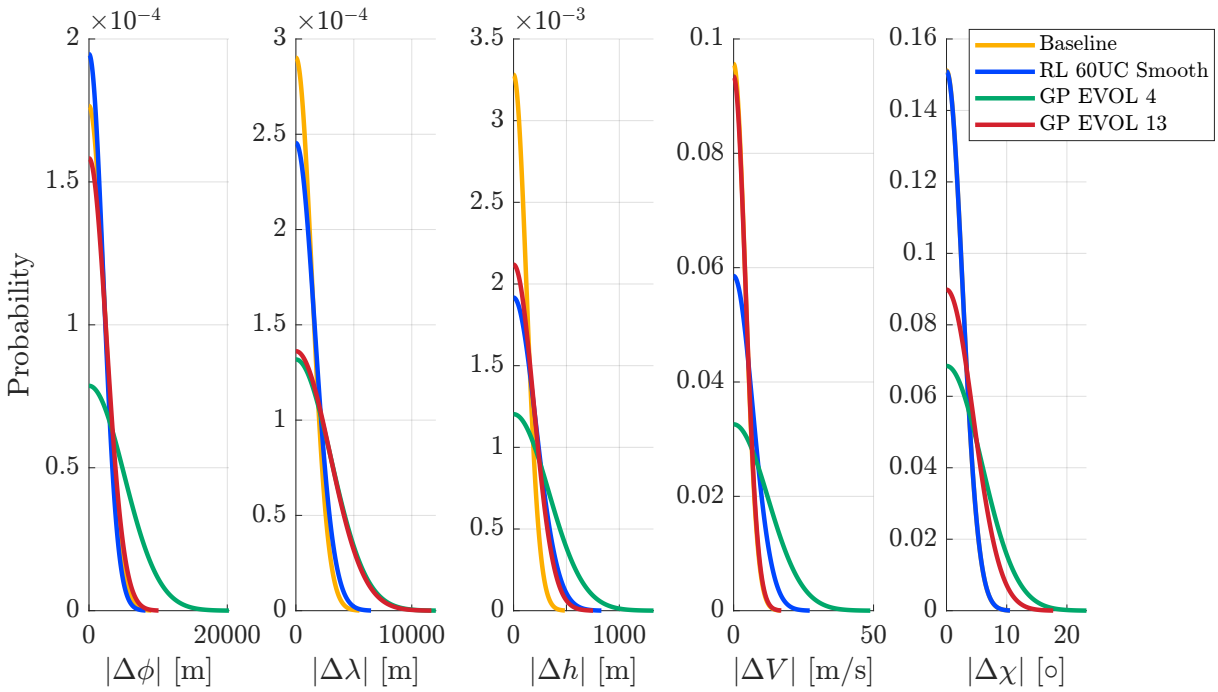
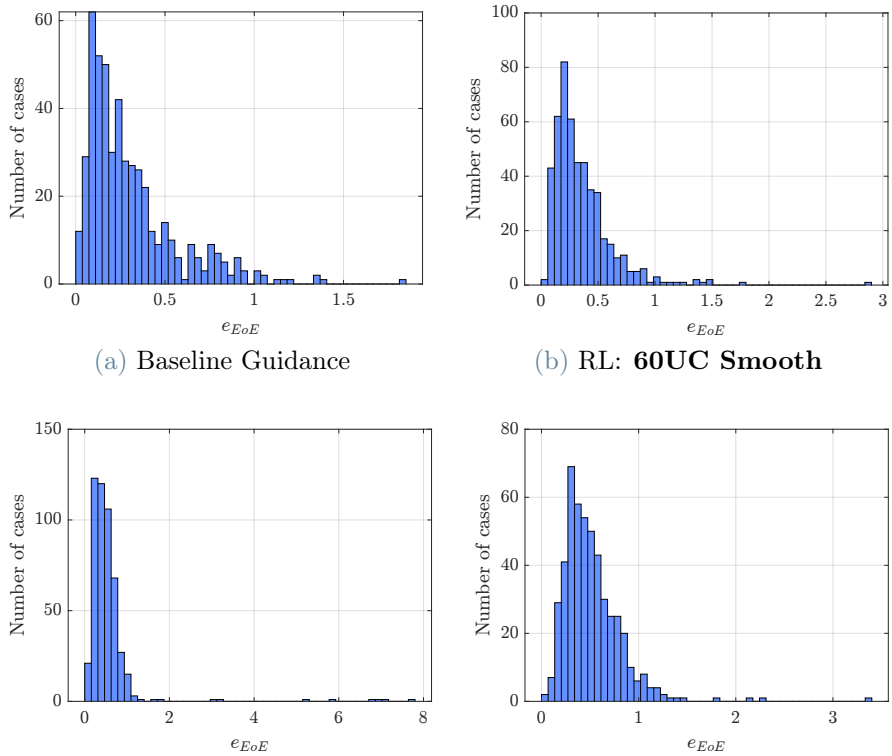


Figure 6.25: State errors distribution at EoE comparison, 500 runs, 6DOF environment

The RL based solution manages to reach the same performance level of the baseline guidance strategy. Notice that a single run of difference in terms of instabilities and



(a) Baseline Guidance (b) RL: **60UC Smooth**

(c) GP: **EVOL 4** (d) GP: **EVOL 13**

Figure 6.26: Total Error, 500 runs, 6DOF environment

failures is negligible from a statistical point of view. Moreover, looking at the PDF in Figure 6.25 it can be seen how the RL error distribution is comparable to the baseline one. However, when it comes to the total error value, which acts as aggregated index of performances, the baseline guidance is still slightly superior. On the other hand, the RL consistently outperforms the GP solutions.

The GP based guidance laws, according to the results in Table 6.16, manages to match the instability rate of the baseline strategy which is the most important requirement. Looking into the performances of the runs which manage to reach the EoE (i.e. the non unstable cases), the GP is slightly outperformed by the baseline algorithm in both **EVOL 4** and **EVOL 13**. In particular, considering the errors distributions in Figure 6.25, 6.26c and 6.26d, **EVOL 13** consistently outperforms **EVOL 4** even if the latter has an higher success rate in Table 6.16. As observed for the RL, the baseline algorithm can achieve better results although they are statistically comparable with those obtained with the GP.

A further performance comparison can be done by taking into account the RL experiment trained using the same dataset as done for the GP, in this way it is possible to gain some

information about how well the information contained in the training dataset are exploited to train the agent. Table 6.17 compares both of the best GP case (i.e. **EVOL 13**) and the median one with the **25UC Smooth** RL experiment.

Experiment Name	3DOF Environment	6DOF Environment		
	Successes (%)	Successes (%)	Failures (%)	Instabilities (%)
GP - median	98.2	95	3	2
GP - best	99	97	2	1
RL - 25UC Smooth	99	97	0	3

Table 6.17: RL and GP performances using the same training dataset

By fixing the training dataset, it is possible to note that the RL outperforms the median GP case both in the training and in the testing environment. However, if we take as reference the best case, the GP can deliver slightly better performances than the RL counterpart.

6.5.2. Techniques Comparison

A further comparison between RL and GP is done by looking into all of those parameters which do not involve the raw guidance performances. This is necessary to highlight positive and negative aspects of the two algorithms. Limiting the analysis only to the flight simulation results would be misleading since there are further factors to consider in order to have the full picture.
The areas that will be here considered are:

- Computational resources needed during the training;
- Computational times for the online execution;
- Generalization capabilities;
- Interpretability of the models;
- Implementation aspects.

Computational resources during training
It is well known that training ML models is a time consuming task. No supercomputer was available during the study, and a Desktop with a 13th Gen Intel(R) Core(TM) i7-13700K processor with 3.4GHz has been used by exploiting parallel computing features on the 24 cores available. The required computational resources are reported in Table 6.18.

Using the described setup, the RL training requires a shorter computational time (considering the 8000 iterations case since the 10000 one was used only during one experiment) than the GP one. On the other hand, when trained with same dataset, the two methodologies consume similar amounts of Random Access Memory (RAM). As expected, the RAM consumption in the RL experiments is strongly affected by the training dataset size. Anyways, in both cases it is theoretically possible to reduce the memory usage by increasing the computational times allocating less parallel instances. Contrarily reducing the computational times would be very difficult without modifying the hardware specifications (i.e. more cores or more powerful processor) or developing a completely compiled version of the training toolkit.

	Reinforcement Learning	Genetic Programming
Training Time (hours)	8000 Iterations: ≈ 46 10000 Iterations: ≈ 58	≈ 57
Ram Usage (GB)	25UC: ≈ 9 60UC: ≈ 11 500UC: ≈ 34	≈ 7

Table 6.18: Training computational resources

Online execution computational time

What is arguably one of the most interesting features of developing a guidance system based on ML techniques, is obtaining a final guidance program able to generate the command in a very short time. For instance, the baseline guidance strategy requires the solution of an optimal control problem in order to generate the guidance commands. On the other hand, both the GP and the RL require the evaluation of a single function, which in the case of the RL it is represented by the NN.

Given the fact that it was impossible to perform a rigorous analysis including compiled source code for the three guidance laws (i.e. GP based, NN and optimal control-based reference), the present comparison aims to define the order of magnitude of the computational time required to execute onboard each of the considered guidance laws. Table 6.19 contains the results of the analysis, the *ML Based* row refers both to the GP-law and to the NN since their evaluation times belong to the same order of magnitude.

Strategy	Computational Time (ms)
Classical	≈ 10
ML Based	≈ 0.01

Table 6.19: Order of magnitude of the expected average computational time to generate the guidance command

The evaluation time of the GP guidance law and of the NN have been measured through a Monte Carlo campaign in the Python environment[4] leading to a mean evaluation time of 0.156 ms for the GP-law and of 0.223 ms for the NN which is higher with respect to the former due to the greater complexity of the latter. Only the GP-law was available in the compiled version, as well as in the interpreted one. The evaluation time of such compiled version of the GP-law code (i.e. C version) is one order of magnitude lower than the one of its interpreted counterpart (i.e. Python version), resulting in the ≈ 0.01 ms estimate of computational time reported in Table 6.19. The execution time for the NN was also estimated considering a factor 10 when passing from the interpreted to the compiled environment according to the GP result and to the literature [68].

On the other hand, results from the optimal control-based reference guidance solution were available as execution time of a compiled source code in a comparable (with respect to the one used for the GP-law and NN testing) hardware with an average value of 33.661 ms.

It is finally worth to mention that while the computational time required by the optimization based method (i.e. the baseline one) depends on the flight phase, the further from the target, the longer the propagation that needs to be done onboard [63], ML based solutions generate the guidance signal through a function evaluation which execution time will be independent on the flight phase.

Generalization capabilities

An important figure of merit in ML applications is the model's generalization capability. Since in this case the training can be performed only in the simplified 3DOF environment, the analysis focuses on the similarity between the performance on the training and testing environments. This is interesting for real-world applications since independently on how precise a given simulator can be, it will always differ from the real world due to the approximations performed. To perform this task, the Jensen-Shannon Divergence has been considered as figure of merit.

[4]The Python environment has been chosen since the NN implementation was better optimized than the simulink one, hence it could produce more reliable results.

The Jensen-Shannon divergence is a method used to measure the similarity between two probability distributions and it is based on the Kullback-Lieber divergence with the difference of making its value symmetric and always finite [19]. In this context it is used to compare the PDF associated to the total error obtained in the 3DOF environment with the PDF associated to the same parameter computed in the 6DOF environment. This approach does not take into account how the single run performs in the two environments, rather it considers the general behavior of an agent in a given uncertainty scenario. Once the Kullback-Lieber divergence is computed [8], also the the Jensen-Shannon divergence can be retrieved. In particular, the square root of it will be used in order to have a value which can be considered a metric distance satisfying the triangular inequality [54], the closer $\sqrt{JS}$ is to 0, the more similar two PDF will be. Table 6.20 and Table 6.21 contain the values of Jensen Shannon divergence for each experiment performed.

Experiment	25UC Smooth	60UC Smooth (↓)	500UC Smooth	500UC Smooth Large (↑)	60UC Standard (*)
$\sqrt{JS}$	0.0896	0.0734	0.1158	0.1223	0.0721

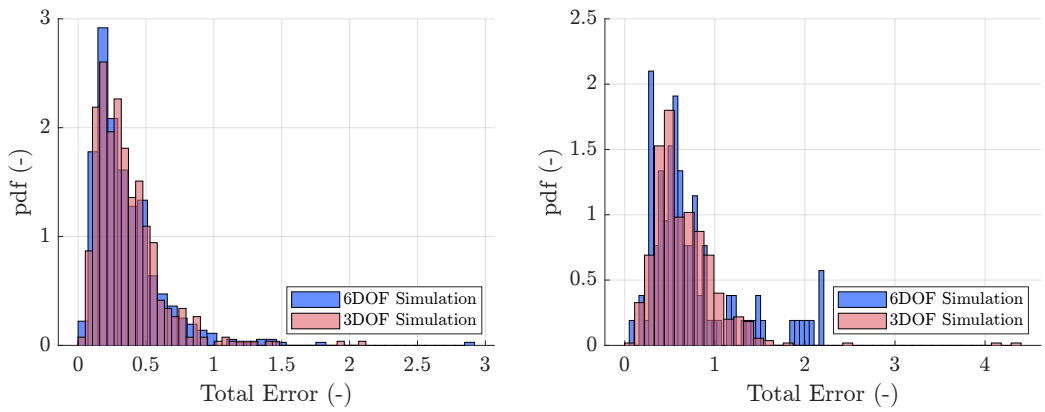
Table 6.20: Jensen-Shannon divergence for RL experiments. Lower means better generalization

Evolution number	1 (↓)	2	3	4	5	6	7	8
$\sqrt{JS}$	0.1086	0.1414	0.1207	0.1716	0.1283	0.1365	0.1237	0.1349
Evolution number	9	10 (↑)	11	12	13	14	15	
$\sqrt{JS}$	0.1195	0.1896	0.1431	0.1859	0.1259	0.1428	0.1674	
median	0.1365							
std dev	0.0249							

Table 6.21: Jensen-Shannon divergence for GP experiments. Lower means better generalization

The highest and the lowest case of each between Table 6.20 and Table 6.21 is marked respectively with (↑) and (↓).
The configuration with the best generalization capabilities wold be **60UC Standard** which however is labeled with (*) because it has been discarded due to its poor flight performances.
Notice that the $\sqrt{JS}$ metrics is computed based on the total error at the EoE which is defined only for stable runs that can reach it. Hence, the **60UC Standard** case PDF has been computed using less samples than in the other cases.

RL experiments outperform the GP ones, **60UC Smooth** is the configuration with the best generalization capabilities in terms of JS divergence while the **EVOL 10** GP case is the worst one.
In Figure 6.27 the histograms corresponding to the PDF of the total error distribution associated the best and worst case in terms of JS divergence are reported.



(a) Lowest $\sqrt{JS}$ case: **60UC Smooth** (b) Highest $\sqrt{JS}$ case: **EVOL 10**

Figure 6.27: Experiments with the highest and lowest JS divergence value

Finally, by comparing the **25UC Smooth** configuration with the results of the 15 IGP experiments, in other words the experiments performed with the same dataset, it is possible to confirm that the RL maintains superior generalization capabilities.

Model interpretability

One of the strengths of the GP is the fact that, differently from the RL, it produces as output an interpretable model (i.e. a symbolic expression which can be read and understood by a human) which is much simpler than the structure of the NN. For instance, the guidance law generated by the **EVOL 13** experiment which has been proven to have performances comparable to the baseline guidance algorithm (Table 6.16) is reported in Equation 6.4 and Equation 6.5 describing respectively the Angle of Attack and Bank Angle correction commands. The equations inputs are the components of both the absolute and relative (identified by the *rel* suffix) GEOC state vector as described in section 5.1. The

symbols are marked with a hat ($\hat{\cdot}$) because they have to be intended as normalized.

$$\begin{aligned}\delta\alpha = \sin\Big(\log\Big(\Big(\sin(\tanh(\hat{\lambda}_{rel}))(\hat{h}_{rel} - \hat{\phi}_{rel})\Big) - (\sin(-8.6727) + \sin(\hat{\chi}))\Big)\Big) \\ \cdot \Bigg(\Bigg(\Bigg(\exp(\hat{\phi}_{rel}) + \hat{\phi}_{rel} + \sqrt{\tanh(\tanh(\hat{E})) - (\sin(\hat{\phi}_{rel}) - \sqrt{\hat{\phi}_{rel} - \hat{\lambda}})}\Bigg) \\ -(\sin(\hat{h}_{rel})(\hat{E} + \hat{\phi} + \hat{v}_{rel}))\Bigg) \cdot (\sin(\hat{\gamma}_{rel})(\hat{E} + \hat{\phi} + \hat{v}_{rel}))\Bigg)\end{aligned} \tag{6.4}$$

$$\begin{aligned}\delta\mu = \sin\Big(\sin\Big(\sin(\hat{\chi}_{rel}) + \Big(\sqrt{\sin(\hat{v}_{rel})} + (\hat{\phi}_{rel} - \hat{v}_{rel})\Big)\Big) \\ + \Big(\hat{\chi}_{rel} + \hat{h}\Big(\Big(\tanh(1.9551) + 2\hat{E}\Big)\sqrt{\hat{h}^2 \cdot \hat{\lambda}}\Big)\Big)\Big)\end{aligned} \tag{6.5}$$

Even though the equations shown in Equation 6.4 and Equation 6.5 is still quite complex, the IGP has been proven effective in generating simpler equations with good performances. Equation 6.6 and Equation 6.7 reports the **EVOL 9** guidance laws that deliver performances above average on the 3DOF environment, and equal to the median result in the high fidelity 6DOF one.

$$\begin{aligned}\delta\alpha = \tanh\Big(\sin\Big(\Big(7.7592\ln(\ln(\sqrt{\hat{\gamma}_{rel}}) + \exp(\hat{v}_{rel})) \\ + \ln(\hat{v}_{rel}) + (-0.0475\hat{v}_{rel} + \exp(\tanh(\tanh(\exp(2.384)))))) \cos(\hat{\gamma}_{rel})\Big)\Big)\hat{\phi}_{rel}\end{aligned} \tag{6.6}$$

$$\delta\mu = (2\hat{h}_{rel} + \hat{\gamma})(\hat{E} + 2\hat{\phi}_{rel} + \tanh((\hat{E} + 8.4941)\hat{\chi}_{rel})) \tag{6.7}$$

Notice that in Equation 6.4 and Equation 6.5 the functions $\sqrt{(\cdot)}$, $\exp(\cdot)$ and $\log(\cdot)$ shall be intended as the corresponding protected versions described in subsection 6.3.1.
On the other hand, the NN used in this study contained 53250 tunable parameters (counting weights and biases), while the larger version used in the **500UC Smooth Large** configuration had 116252 of them, resulting in extremely complex black box models.

In general, having a simpler and white box program structure has three main benefits:

1. Simpler deployment on the flight software;
2. Lower computational time to perform the evaluation;
3. Easier to perform a-posteriori studies on the output of the model.

The first two points have already been proven through this dissertation, while the third one is a viable path for further studies which were outside the scope of the present work.

In general the GP law structure is much simpler than the NN one due to the usage of a full set of nonlinear function and due to the possibility of autonomously evolve the structure of the law itself, while including the capability to select a subset of terminal symbols (i.e. input variables) to be actually used in the function. In the RL case on the other hand, the network structure including input variables and activation functions is defined a-priori without the possibility of updating it while the training goes on.

Implementation aspects

This last paragraph aims to give an overview on some aspects related to the implementation. First of all, it is interesting to recall the way the two methodologies deal with the uncertainties. If on one side, the GP requires to propagate the full trajectory for each uncertainty case in its training dataset to compute the fitness function associated to each individual, on the other, the RL only needs to propagate a subset of them at each iteration. This implies that if the other parameters are kept constant, increasing the size of the training dataset in the RL framework does not modify the computational time to perform a given number of iterations, while the GP will be strongly affected by it. However the RL might require more training iterations with bigger datasets to fully exploit the wider pool of training cases.

Another difference lays in the fact that the GP features a lower number of tuning hyperparameters with respect to the RL making its setup theoretically easier. However, this quality is compensated by the intrinsic stochasticity of the GP which makes necessary to repeat a given training experiment several times in order to properly assess its performances.

Finally, being the GP libraries simpler from an implementation point of view, the quality of the postprocess done on the corresponding trainings is finer with respect to the RL (section 5.4) which could be potentially improved by having more control on the libraries themselves.

7 | Conclusions and Future developments

This chapter aims to outline the conclusions of the study based on the results provided in chapter 6 and answering the initial research questions in section 1.3. Even though chapter 6 contains already a detailed overview of the results, the most important of them will be repeated in section 7.1 to support the final discussion. The chapter ends with section 7.2 in which some unexplored areas or future developments strictly related to the present work will be suggested.

7.1. Conclusions

This study proves the effectiveness of DRL and IGP in solving the ReFEx online reentry guidance problem in the 3DOF training environment. The solution derived within this simplified framework was subsequently proven proven capable of successfully guiding the vehicle to the EoE, as well as in the high fidelity 6DOF simulator.

The performances of DRL and GP based solutions have been assessed using as reference term the baseline algorithm specifically developed for ReFEx by the DLR team, which it is based on the optimal control theory. Results show that both DRL and GP based solutions manage to achieve performances comparable to the baseline ones, but requiring a computational time to generate the guidance signal approximately three orders of magnitude lower. It is worth remarking that this result has been achieved without retraining the ML models, which have then been tested on a different environment and unseen scenarios.

Chapter 4 and chapter 5 explore the the first research question:

> **[RQ 1]**: How can a RL or GP based model be trained to generate the closed loop online guidance signal for a reentry mission?

The training methodology can be conceptually divided into three parts:

Vehicle dynamical model : described in chapter 4, is designed to approximate as well as possible the actual dynamics of testing environment. This has been done by including the navigation errors and by limiting the guidance signal to values which can be actually tracked by the real 6DOF system. Perturbations applied to the initial conditions as well as to the atmospheric and vehicle model have also been considered.

Generation of the training dataset : it contains the uncertainty cases (i.e. instances of perturbations that define the conditions under which flight simulations are conducted) to be used during the training. Careful selection of these cases is crucial to ensure that the dataset is sufficiently representative of real-world scenarios and it is described in section 5.2.

Definition of the reward and fitness functions : this is arguably the most challenging part since its definition requires balancing several and often conflicting objectives in the same scalar value. Extensive testing has been needed to find the proper setup since also the smallest error in its formulation can lead the training to fail. Both functions (i.e. reward and fitness) contain the same terms: minimization of the distance from the target at the EoE, keeping the vehicle within the flight envelope in which aerodynamic data are available, limiting the guidance signal in a feasible range and staying close to the reference trajectory in the final phase of the flight (see section 5.3). The reward function has been further augmented to promote the generation of a smooth guidance signal (Equation 5.28) having a strong beneficial effect in stabilizing the guidance policy in the testing environment (see section 6.4). On the other hand, the biggest challenge faced during the design of the GP fitness function, was the need of specifying a scalar fitness value related to the performances of an individual across each uncertainty case in the training dataset at once. This issue has been tackled in subsection 5.3.1, where a fitness function divided in layers based on the success rate of each individual has been suggested. The fitness value within each layer is associated to the specific flight performances of the individual in each of the uncertainty cases.

The second research question:

> **[RQ 2]**: To what extent can machine learning–based online guidance algorithms (specifically RL and GP) match or exceed the performances of a state-of-the-art guidance law when evaluated in a high-fidelity flight dynamics simulator?

is addressed in chapter 6, where it is shown that even though none of the ML based solutions obtained in this study can outperform the guidance accuracy of the baseline algorithm, both the DRL and GP application can provide solutions comparable with it

(Table 6.16).
The DRL configuration that provides the best results is **60UC Smooth** which features a 60 elements size training dataset and the smooth formulation of the reward signal. It manages to deliver performances analogous to the baseline algorithm: namely 97.4% success rate and 1.4% instability rate with respect to the 97.8% and 1.2% of the latter. The importance of promoting a smooth guidance signal can be noticed by comparing the instability rate of the **60UC Smooth** configuration with its standard counterpart **60UC Standard** in Table 6.14 (1% and 8% respectively). Differently from the DRL, the GP results suffer from the stochasticity effects intrinsic to the evolutionary process. This led to the need to perform several experiments with the same setup to assess the statistical properties of the algorithm itself, and thus, 15 separate evolutions were computed. The average IGP solution underperforms in comparison to the baseline algorithm, with a median success rate of 95% and a median instability rate of 2% (respectively 97% and 1% in the baseline case when evaluated in the 100 runs campaign). However, the best result could achieve performances comparable to those of the baseline: 97% success rate and 1% instability rate in the 100 runs Monte Carlo campaign, which get to 96.2% and 1.2% after completing the 500 runs campaign. Moreover, both ML techniques require approximately 0.01 ms to generate the online guidance command independently from the phase of the flight, proving their online capabilities. This is a major benefit compared to the classical approach optimization-based, that requires ≈ 10 ms to compute each instruction; in particular, the computational time of the latter method depends on the flight phase, the further from the target, the longer the propagation that needs to be done onboard.

Finally, the third research question:

> **[RQ 3]**: What are the comparative strengths and limitations of RL and GP in the context of online guidance for atmospheric reentry missions?

has also been analyzed in chapter 6. Considering exclusively the flight performances, the DRL consistently outperforms the GP, both in accuracy to the target and generalization capabilities. However, when the RL training is executed using the same dataset as in the GP one, namely the **25UC Smooth** configuration, the differences reduce. The **25UC Smooth** experiment delivers a 97% success rate and a 3% instability rate on the 100 runs Monte Carlo campaign, which is slightly better than the median GP performance, but worse than the best one.
When it comes to the comparison between the two techniques, regardless of the flight simulations outcome, the time needed to evaluate the GP-law is lower than its NN counterpart due to the greater complexity of the latter. Moreover, the GP-law intrinsically

produces a smooth guidance signal which is well suited for GC application, whereas the NN tends to suffer from the action shaking problem if not properly limited. Finally, GP returns a white box interpretable model, which is opposite to the black box characteristics of the NN. This is particularly interesting because, even if outside from the scope of this work, it is possible to include at evolution level constraints on the size of the GP-law or to enforce some desired properties (e.g. stability) in order to obtain a human-readable model reliable for critical applications.

DRL could deliver guidance performances superior to the GP ones for two main reasons. Firstly, the most likely reason for the increased performances of the DRL methodology is the training logic typical of the RL and of the PPO algorithm, described in chapter 2, which proceeds by batches of state-action-reward values and aims to associate to each state the corresponding optimal action. In particular, it is possible to use a wider training dataset without strongly affecting the training computational time (i.e. 60 elements in the best RL case with respect to the 25 in the GP one). At each RL training iteration a subset of the uncertainty cases in the dataset is propagated and used to compute the PPO policy rollout, whereas the GP needs to fully propagate each element in the training dataset for each individual in order to evaluate all of them in the same exact conditions to properly conduce the evolution. If the dataset changes between subsequent iterations, the evolution is expected to diverge, therefore it is fundamental to keep the dataset fixed, limiting the amount of uncertainty cases that can be processed.

Secondly, the higher complexity of the network with respect to the symbolic expression of the GP improves its generalization capabilities. However, in subsection 6.3.4 it has been shown that the individual size does not directly affect its performances. In particular, some below average individuals could outperform others above average. Moreover, a lot of large size individuals have not been tested in the 6DOF environment since they were discarded during the postprocessing stage. For this reason, the law complexity seems to play a second order role with respect to the factor mentioned above.

A further point of comparison is the stochasticity of the training: even though both methodologies include some random process in determinate phases, the RL proceeds by iteratively improving the same NN, this results in making the random processes (for example the initialization of the NN itself) negligible with respect to the training output when all the other parameters are kept constant. On the other hand, the GP evolution is strongly driven by random processes, which makes it necessary to perform several evolutions with the same setup to characterize its performances. It is possible to reduce the standard deviation of the result by properly designing the training environment and the fitness function, but the stochasticity is intrinsic in the GP formulation, making the

testing of a given setup way more time consuming than the cost of performing a single evolution.

7.2. Future developments

In this section some areas of possible improvement for further studies will be suggested. Three research directions are identified: the first is about improving the training environment while keeping the RL and GP settings unaltered. On the other hand, the second and third research directions are related to areas of improvement strictly tied to the ML algorithms. Finally, the possibility of creating a hybrid technique able to merge the positive aspects of RL and GP will be outlined.

Training Environment

Three main areas of improvement have been identified with respect to the training environment:

- As described in section 4.3, the training simulator contains a logic aiming to limit the guidance command output to keep the vehicle within the safety corridor shown in Figure 4.8 when propagating the 3DOF dynamic. This is necessary to avoid those regions of the flight envelope in which the vehicle results to be underactuated, since the 3DOF model cannot capture the aerodynamic instabilities which affect the rotational dynamics. A possible development in this direction could consist in removing the safety corridor constraint, while keeping the one related to the availability of aerodynamic data, and replacing it with a penalization proportional to the integral of the trimmability value (shown in the color scale in Figure 4.8). Those trajectories flying through regions at high risk of instability will in this way be penalized with respect to safer ones and the training will eventually converge towards a safe solution without the limitations related to the satisfaction of a predetermined corridor.

- Use a 6DOF model of ReFEx within the training environment. This model would be a simplified version of the high-fidelity one described in section 4.4, to limit the required computational time. However, it would enable greater similarity between the training and testing environments, ultimately making it possible to directly model aerodynamic instabilities during training.

- Improve the RL reward function. An extra term could be added to penalize the partial derivative of the controls with respect to the full observation state instead of considering only their time derivatives. This should promote and improve the

smoothness of the final policy.

DRL usage

- Different NN architectures could be explored. The present work focused only on MLP since the objective was not only to optimize as much as possible the DRL policy guidance performances, but also to have a fair comparison with the GP. In a framework in which the only objective is to have the best guidance law possible, experimenting with more advanced NN architectures, first of all the Transformer, is definitely a reasonable development direction;
- Using the GPU during the training may be beneficial to reduce the training time;
- The way the combination between NN size and training dataset size affects the guidance performances of the trained agent is still partially an open question, which could not be fully analyzed and understood within this work.
- Even if the stability properties of the final policy have been verified empirically during the testing stages, a refined approach aiming to train in parallel the policy network and the corresponding Lyapunov function could be adopted [12].
- Perform a stage of fine-tuning of the several hyperparameters.

GP usage

- Perform a set of GP experiments with a larger training dataset (i.e. 60 or 500 elements) in order to better asses how the size of the training dataset affects the final performances; improving also the quality of the comparison with respect to the RL. This analysis has not been done within the present work because of the high computational time required with the available hardware.
- A further but more complex research direction could be in general to work on improving the IGP algorithm itself to deliver better results and reduce the stochasticity effects. It is noteworthy that the research done in terms of RL algorithm in the past years has been much wider and extensive than what has been done with the GP which can be considered a much less mature technology when it comes to GC applications.
- Similarly to what has been suggested in the RL case, the GP framework can be used to evolve in parallel the controller structure and the corresponding Lyapunov stability function to guarantee the stability of the controller itself [2, 24].

- As for the RL case, a fine-tuning of the algorithm parameters could improve its performances.

Among all the other possible further developments of the present work, the most promising one is deriving a new hybrid ML technique aiming to merge the effectiveness and the framework of RL with the interpretability of the GP model. This hybrid technique would represent the agent of the MDP using the tools developed by the GP but it would be trained and evolved according to the RL methodology, allowing to output a fully interpretable model that potentially could have the same performance of using a NN as in the standard RL formulation. Some research on this area already exists but not applied to problem with this level of complexity [29, 30].

Bibliography

[1] B. Acikmese and S. R. Ploen. Convex programming approach to powered descent guidance for mars landing. *Journal of Guidance, Control, and Dynamics*, 30(5): 1353–1366, 2007. doi: 10.2514/1.27553. URL `https://doi.org/10.2514/1.27553`.

[2] M. M. A. Ali, A. Jamali, A. Asgharnia, R. Ansari, and R. Mallipeddi. Multi-objective lyapunov-based controller design for nonlinear systems via genetic programming. *Neural Computing and Applications*, 34(2):1345–1357, 2021. doi: 10.1007/s00521-021-06453-1. URL `https://doi.org/10.1007/s00521-021-06453-1`.

[3] F. AlMahamid and K. Grolinger. Reinforcement learning algorithms: An overview and classification. In *2021 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*, pages 1–7. IEEE, 2021.

[4] W. Bauer, P. Rickmers, A. Kallenbach, S. Stappert, V. Wartemann, C. H.-J. Merrem, R. Schwarz, M. Sagliano, J. T. Grundmann, A. Flock, et al. Dlr reusability flight experiment refex. *Acta Astronautica*, 168:57–68, 2020.

[5] H.-G. Beyer and H.-P. Schwefel. Evolution strategies–a comprehensive introduction. *Natural computing*, 1:3–52, 2002.

[6] S. R. Bieniawski, B. Lewis, B. Friia, A. Mahajan, K. Somervill, P. Mamidipudi, and D. Dakin. New shepard flight test results from blue origin de-orbit descent and landing tipping point. In *AIAA SCITECH 2022 Forum*, 2022. doi: 10.2514/6.2022-1829.

[7] C. M. Bishop. *Pattern recognition and Machine Learning*. Springer, 2006.

[8] S. L. Brunton and J. N. Kutz. *Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control*. Cambridge University Press, 2019.

[9] L. Bussler, J. L. Redondo Gutierrez, and P. Rickmers. Refex: Reusability flight experiment – trajectory design. 2023. 10th European Conference for Aeronautics and Space Sciences (EUCASS).

[10] J. Carradori. *6-DOF atmospheric rocket landing guidance using meta-Reinforcement Learning*. PhD thesis, Delft University of Technology, 2024.

[11] D. Celestini, D. Gammelli, T. Guffanti, S. D'Amico, E. Capello, and M. Pavone. Transformer-based model predictive control: Trajectory optimization via sequence modeling. *IEEE Robotics and Automation Letters*, 9(11):9820–9827, 2024. doi: 10.1109/LRA.2024.3466069.

[12] P. Cocaul, S. Bertrand, H. Piet-Lahanier, L. Lemazurier, and M. Ganet. Deep reinforcement learning design of safe, stable and robust control for sloshing-affected space launch vehicles. *Control Engineering Practice*, 161:106328, 2025. ISSN 0967-0661. doi: https://doi.org/10.1016/j.conengprac.2025.106328. URL `https://www.sciencedirect.com/science/article/pii/S0967066125000917`.

[13] A. H. Cordesman and J. Kendall. *Chinese space strategy and developments*. Center for Strategic and International Studies (CSIS), 2016.

[14] J. G. Dias Basto da Silva, J. L. Redondo Gutierrez, and P. Bernal Polo. Refex navigation design: Improvements to the navigation filter. In *Proceedings of the International Astronautical Congress, IAC*, 2024. URL `https://elib.dlr.de/207879/`.

[15] DLR. Refex mission, official dlr video,. `https://www.youtube.com/watch?feature=shared&v=eerdVjl8M7I`, 2020. Accessed: 2024-11-21.

[16] D. P. Drob, J. T. Emmert, J. W. Meriwether, J. J. Makela, E. Doornbos, M. Conde, G. Hernandez, J. Noto, K. A. Zawdie, S. E. McDonald, J. D. Huba, and J. H. Klenzing. An update to the horizontal wind model (hwm): The quiet time thermosphere. *Earth and Space Science*, 2(7):301–319, 2015. doi: https://doi.org/10.1002/2014EA000089. URL `https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1002/2014EA000089`.

[17] L. Federici, A. Scorsoglio, A. Zavoli, and R. Furfaro. Autonomous guidance for cislunar orbit transfers via reinforcement learning. 08 2021.

[18] F.-A. Fortin, F.-M. De Rainville, M. Gardner, M. Parizeau, and C. Gagné. Deap: Evolutionary algorithms made easy. *Journal of Machine Learning Research, Machine Learning Open Source Software*, 13:2171–2175, 07 2012.

[19] B. Fuglede and F. Topsoe. Jensen-shannon divergence and hilbert space embedding. In *International Symposium on Information Theory, 2004. ISIT 2004. Proceedings.*, pages 31–, 2004. doi: 10.1109/ISIT.2004.1365067.

[20] S. Fujimoto, H. van Hoof, and D. Meger. Addressing function approximation error in actor-critic methods, 2018. URL `https://arxiv.org/abs/1802.09477`.

[21] J. Gao, X. Shi, Z. Cheng, J. Xiong, L. Liu, Y. Wang, and Y. Yang. Reentry trajectory optimization based on deep reinforcement learning. In *2019 Chinese Control And Decision Conference (CCDC)*, pages 2588–2592. IEEE, 2019.

[22] A. Garcia, S. S. C. Yamanaka, A. N. Barbosa, F. C. P. Bizarria, W. Jung, and F. Scheuerpflug. Vsb-30 sounding rocket: history of flight performance. *Journal of Aerospace Technology and Management*, 3(3):325–330, 2011.

[23] B. Gaudet, R. Linares, and R. Furfaro. Deep reinforcement learning for six degree-of-freedom planetary landing. *Advances in Space Research*, 65(7):1723–1741, 2020.

[24] B. Grosman and D. R. Lewin. Automatic generation of lyapunov functions using genetic programming. *IFAC Proceedings Volumes*, 38(1):75–80, 2005. ISSN 1474-6670. doi: https://doi.org/10.3182/20050703-6-CZ-1902.00873. URL `https://www.sciencedirect.com/science/article/pii/S1474667016368859`. 16th IFAC World Congress.

[25] T. Guffanti, D. Gammelli, S. D'Amico, and M. Pavone. Transformers for trajectory optimization with application to spacecraft rendezvous. In *2024 IEEE Aerospace Conference*, pages 1–13. IEEE, 2024.

[26] J. L. R. Gutierrez, P. B. Polo, B. Gäßler, J. Robens, P. Acquatella, D. Seelbinder, and S. Theil. Refex: Reusability flight experiment-architecture and algorithmic design of the gnc subsystem. In *10th European Conference for Aeronautics and Space Sciences (EUCASS)*, 2023.

[27] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine. Soft actor-critic: Off-policy maximum entropy deep reinforcement with a stochastic actor. In *Proceedings of the 35th International Conference on Machine Learning. July 10th-15th, Stockholm, Sweden*, volume 1870, 1861.

[28] T. Haarnoja, A. Zhou, K. Hartikainen, G. Tucker, S. Ha, J. Tan, V. Kumar, H. Zhu, A. Gupta, P. Abbeel, and S. Levine. Soft actor-critic algorithms and applications. *arXiv preprint arXiv:1812.05905*, 2018.

[29] D. Hein, S. Udluft, and T. A. Runkler. Interpretable policies for reinforcement learning by genetic programming. *Engineering Applications of Artificial Intelligence*, 76:158–169, 2018. ISSN 0952-1976. doi: https://doi.org/10.1016/j.

engappai.2018.09.007. URL https://www.sciencedirect.com/science/article/pii/S0952197618301933.

[30] D. Hein, S. Limmer, and T. A. Runkler. Interpretable control by reinforcement learning. *IFAC-PapersOnLine*, 53(2):8082–8089, 2020. doi: https://doi.org/10.1016/j.ifacol.2020.12.2277. URL https://www.sciencedirect.com/science/article/pii/S2405896320329372.

[31] H. J. Hwang, J. Jang, J. Choi, J. H. Bae, S. H. Kim, and C. O. Kim. Stepwise soft actor–critic for uav autonomous flight control. *Drones*, 7(9), 2023. ISSN 2504-446X. doi: 10.3390/drones7090549. URL https://www.mdpi.com/2504-446X/7/9/549.

[32] Q. Jiang, X. Wang, and Y. Li. Intelligent reentry guidance with dynamic no-fly zones based on deep reinforcement learning. In *International Conference on Computational & Experimental Engineering and Sciences*, pages 291–313. Springer, 2023.

[33] J. R. Koza and R. Poli. A genetic programming tutorial. *Introductory tutorials in optimization, search and decision support*, 8, 2003.

[34] Z. Krevor, R. Howard, T. Mosher, and K. Scott. Dream chaser commercial crewed spacecraft overview. In *17th AIAA International Space Planes and Hypersonic Systems and Technologies Conference*, page 2245, 2011.

[35] S. Li, Y. Yan, H. Qiao, X. Guan, and X. Li. Reinforcement learning for computational guidance of launch vehicle upper stage. *International Journal of Aerospace Engineering*, 2022(1):2935929, 2022.

[36] S. Li, Y. Yan, H. Qiao, X. Guan, and X. Li. Reinforcement learning for computational guidance of launch vehicle upper stage. *International Journal of Aerospace Engineering*, 2022(1):2935929, 2022.

[37] Y. Li. Deep reinforcement learning, 2018. URL https://arxiv.org/abs/1810.06339.

[38] P. Lu. Introducing computational guidance and control, 2017. URL https://doi.org/10.2514/1.G002745.

[39] S. Luke and L. Panait. Fighting bloat with nonparametric parsimony pressure. In J. J. M. Guervós, P. Adamidis, H.-G. Beyer, H.-P. Schwefel, and J.-L. Fernández-Villacañas, editors, *Parallel Problem Solving from Nature — PPSN VII*, pages 411–421, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg. ISBN 978-3-540-45712-1.

[40] Y. Luo, J. Wang, J. Jiang, and H. Liang. Reentry trajectory planning for hypersonic

vehicles via an improved sequential convex programming method. *Aerospace Science and Technology*, 149:109130, 2024. ISSN 1270-9638. doi: https://doi.org/10.1016/j.ast.2024.109130. URL `https://www.sciencedirect.com/science/article/pii/S1270963824002633`.

[41] D. Malyuta, T. P. Reynolds, M. Szmuk, T. Lew, R. Bonalli, M. Pavone, and B. Acikmese. Convex optimization for trajectory generation, 2021. URL `https://arxiv.org/abs/2106.09125`.

[42] Y. Mao, M. Szmuk, X. Xu, and B. Açikmese. Successive convexification: A superlinearly convergent algorithm for non-convex optimal control problems. *arXiv preprint arXiv:1804.06539*, 2018.

[43] F. Marchetti and E. Minisci. A hybrid neural network-genetic programming intelligent control approach. In *International Conference on Bioinspired Methods and Their Applications*, pages 240–254. Springer, 2020.

[44] F. Marchetti and E. Minisci. Genetic programming guidance control system for a reentry vehicle under uncertainties. *Mathematics*, 9:1868, 08 2021. doi: 10.3390/math9161868.

[45] F. Marchetti and E. Minisci. Inclusive genetic programming. In *Genetic Programming: 24th European Conference, EuroGP 2021, Held as Part of EvoStar 2021, Virtual Event, April 7–9, 2021, Proceedings 24*, pages 51–65. Springer, 2021.

[46] F. Marchetti, J. L. Redondo Gutierrez, and D. Seelbinder. Genetic programming guidance for the reentry trajectory of the ReFEx vehicle. In *IAF Space Transportation Solutions and Innovations Symposium*, pages 362–375, Paris, France, 2024. International Astronautical Federation (IAF).

[47] S. Mceowen, A. Kamath, P. Elango, T. Kim, and B. Açıkmeşe. High-accuracy 3-dof hypersonic reentry guidance via sequential convex programming. In *AIAA SCITECH 2023 Forum*, 01 2023. doi: 10.2514/6.2023-0300. URL `https://arc.aiaa.org/doi/abs/10.2514/6.2023-0300`.

[48] K. Mease, D. Chen, P. Teufel, and H. Schonenberger. Reduced-order entry trajectory planning for acceleration guidance. *Journal of Guidance, Control, and Dynamics*, 25 (2):257–266, 2002. doi: 10.2514/2.4906. URL `https://doi.org/10.2514/2.4906`.

[49] P. Menon, M. Yousefpor, T. Lam, and M. Steinberg. Nonlinear flight control system synthesis using genetic programming. In *Guidance, Navigation, and Control Conference*, page 3224, 1995.

[50] C. H.-J. Merrem, V. Wartemann, and T. Eggers. Preliminary aerodynamic design of a reusable booster flight experiment. *CEAS Space Journal*, April 2020. URL https://elib.dlr.de/137174/.

[51] C. H.-J. Merrem, V. Wartemann, H. Elsaesser, T. Ruhe, and T. Eggers. Rolling capabilities of the experimental vehicle refex. 2022.

[52] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

[53] E. Mooij. *Re-entry systems*. Springer Nature, 2024.

[54] F. Nielsen. On the jensen–shannon symmetrization of distances relying on abstract means. *Entropy*, 21(5), 2019. ISSN 1099-4300. doi: 10.3390/e21050485. URL https://www.mdpi.com/1099-4300/21/5/485.

[55] C. K. Oh and G. J. Barlow. Autonomous controller design for unmanned aerial vehicles using multi-objective genetic programming. In *Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753)*, volume 2, pages 1538–1545. IEEE, 2004.

[56] P. Pei, S. Fan, W. Wang, and D. Lin. Online reentry trajectory optimization using modified sequential convex programming for hypersonic vehicle. *IEEE Access*, 9: 23511–23525, 2021. doi: 10.1109/ACCESS.2021.3056517.

[57] J. M. Picone, A. E. Hedin, D. P. Drob, and A. C. Aikin. Nrlmsise-00 empirical model of the atmosphere: Statistical comparisons and scientific issues. *Journal of Geophysical Research: Space Physics*, 107(A12):SIA 15–1–SIA 15–16, 2002. doi: https://doi.org/10.1029/2002JA009430. URL https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2002JA009430.

[58] R. Poli, W. Langdon, N. Mcphee, and J. Koza. *A Field Guide to Genetic Programming*. 01 2008. ISBN 978-1-4092-0073-4. URL http://www.cs.ucl.ac.uk/staff/W.Langdon/ftp/papers/poli08_fieldguide.pdf.

[59] M. L. Puterman. Markov decision processes. *Handbooks in operations research and management science*, 2:331–434, 1990.

[60] A. V. Rao. A survey of numerical methods for optimal control. *Advances in the astronautical Sciences*, 135(1):497–528, 2009.

[61] J. L. Redondo Gutierrez, D. Seelbinder, J. Robens, and P. Acquatella. Design of

a control allocation solution for the winged reusable launch vehicle refex. In *AIAA Scitech 2022 Forum*, page 1842, 2022.

[62] J. L. Redondo Gutierrez, D. Seelbinder, S. Theil, P. Bernal Polo, B. Gäßler, J. Robens, and P. Acquatella B. Refex: Reusability flight experiment-architecture and algorithmic design of the gnc subsystem. 2023.

[63] J. L. Redondo Gutierrez, A. Wenzel, D. Seelbinder, and S. Theil. Flight readiness of refex guidance: Preparation for integration into flight software. In *AIAA SciTech 2025 Forum*, 2025. URL `https://elib.dlr.de/213334/`.

[64] J. L. R. Redondo Gutierrez, D. Seelbinder, and S. Theil. Design of refex guidance: Trajectory correction after ascent. In *AIAA SCITECH 2023 Forum*, page 1995, 2023.

[65] J. L. R. Redondo Gutierrez, D. Seelbinder, and S. Theil. Design of refex guidance: Periodic on-board trajectory updates. In *AIAA Scitech 2024 Forum*, page 0087, 2024.

[66] P. Rickmers, E. Dumont, S. Krummen, J. L. Redondo Gutierrez, L. Bussler, S. Kottmeier, G. Wubbels, H. Martens, S. Woicke, M. Sagliano, J. S. Haseker, L. Witte, M. Sippel, W. Bauer, and H.-J. Peetz. The callisto and refex flight experiments at dlr - challenges and opportunities of a wholistic approach. *Acta Astronautica*, 225, 2024. URL `https://elib.dlr.de/206771/`.

[67] J. Robens, B. Gäßler, and P. Acquatella B. Refex: Reusability flight experiment-design of an exoatmospheric control system. In *Aerospace Europe Conference 2023-10th EUCASS-9th CEAS*, 2023.

[68] P. Rysak. Comparative analysis of code execution time by c and python based on selected algorithms. *Journal of Computer Sciences Institute*, 26:93–99, 03 2023. doi: 10.35784/jcsi.3109.

[69] C. Sánchez-Sánchez and D. Izzo. Real-time optimal control via deep neural networks: Study on landing problems. *Journal of Guidance, Control, and Dynamics*, 41(5): 1122–1135, 2018. doi: 10.2514/1.G002357. URL `https://doi.org/10.2514/1.G002357`.

[70] D. P. Scharf, B. Açıkmeşe, D. Dueri, J. Benito, and J. Casoliva. Implementation and experimental demonstration of onboard powered-descent guidance. *Journal of Guidance, Control, and Dynamics*, 40(2):213–229, 2017. doi: 10.2514/1.G000399.

[71] J. Schulman, S. Levine, P. Moritz, M. I. Jordan, and P. Abbeel. Trust region policy optimization, 2017. URL `https://arxiv.org/abs/1502.05477`.

[72] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

[73] R. Schwarz, D. Kiehn, G. F. Trigo, B. Razgus, A. Wenzel, D. Seelbinder, J. Sommer, S. Theil, M. Markgraf, M. Dumke, M. Reigenborn, M. Bestard Körner, M. Solari, B. Braun, and D. Pfau. Overview of flight guidance, navigation, and control for the dlr reusability flight experiment (refex). In *8th European Conference for Aeronauticas and Space Sciences (EUCASS)*, July 2019. URL https://elib.dlr.de/129086/.

[74] Z. Shen and P. Lu. Onboard generation of three-dimensional constrained entry trajectories. *Journal of Guidance, Control, and Dynamics*, 26(1):111–121, 2003. doi: 10.2514/2.5021. URL https://doi.org/10.2514/2.5021.

[75] P. Simplício, A. Marcos, E. Joffre, M. Zamaro, and N. Silva. Review of guidance techniques for landing on small bodies. *Progress in Aerospace Sciences*, 103:69–83, 2018. ISSN 0376-0421. doi: https://doi.org/10.1016/j.paerosci.2018.10.005. URL https://www.sciencedirect.com/science/article/pii/S0376042118300101.

[76] T. Soule and J. A. Foster. Removal bias: a new cause of code growth in tree based evolutionary programming. In *1998 IEEE international conference on evolutionary computation proceedings. IEEE World Congress on Computational Intelligence (Cat. No. 98TH8360)*, pages 781–786. IEEE, 1998.

[77] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, second edition, 2018. URL http://incompleteideas.net/book/the-book-2nd.html.

[78] M. Szmuk, T. P. Reynolds, and B. Açıkmeşe. Successive convexification for real-time six-degree-of-freedom powered descent guidance with state-triggered constraints. *Journal of Guidance, Control, and Dynamics*, 43(8):1399–1413, 2020. URL https://doi.org/10.2514/1.G004549.

[79] Y. Takubo, T. Guffanti, D. Gammelli, M. Pavone, and S. D'Amico. Towards robust spacecraft trajectory optimization via transformers, 2025. URL https://arxiv.org/abs/2410.05585.

[80] M. Tipaldi, R. Iervolino, and P. R. Massenio. Reinforcement learning in spacecraft control applications: Advances, prospects, and challenges. *Annual Reviews in Control*, 54:1–23, 2022.

[81] M. Towers, A. Kwiatkowski, J. Terry, J. U. Balis, G. D. Cola, T. Deleu, M. Goulão, A. Kallinteris, M. Krimmel, A. KG, R. Perez-Vicente, A. Pierré, S. Schulhoff, J. J.

Tai, H. Tan, and O. G. Younis. Gymnasium: A standard interface for reinforcement learning environments, 2024. URL https://arxiv.org/abs/2407.17032.

[82] L. Vanneschi and R. Poli. *Genetic Programming — Introduction, Applications, Theory and Open Issues*, pages 709–739. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012. ISBN 978-3-540-92910-9. doi: 10.1007/978-3-540-92910-9_24. URL https://doi.org/10.1007/978-3-540-92910-9_24.

[83] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[84] C. Verdier and M. Mazo Jr. Formal controller synthesis via genetic programming. *IFAC-PapersOnLine*, 50(1):7205–7210, 2017.

[85] J. Wang, Y. Wu, M. Liu, M. Yang, and H. Liang. A real-time trajectory optimization method for hypersonic vehicles based on a deep neural network. *Aerospace*, 9(4):188, 2022.

[86] A. Watanabe, K. Yonemoto, T. Fujikawa, T. Matsukami, T. Otsuki, Y. Kitazono, Y. Koshida, M. Murakami, T. Watanabe, H. Atarashi, et al. Design and development status of experimental winged rocket wires# 015. In *Asia-Pacific international symposium on aerospace technology*, pages 323–337. Springer, 2021.

[87] H. Xie. *An analysis of selection in genetic programming.* PhD thesis, Te Herenga Waka-Victoria University of Wellington, 2009.

[88] M. Xu, K. Chen, L. Liu, and G. Tang. Quasi-equilibrium glide adaptive guidance for hypersonic vehicles. *Science China Technological Sciences*, 55:856–866, 2012.

[89] A. Zavoli and L. Federici. Reinforcement learning for robust trajectory design of interplanetary missions. *Journal of Guidance, Control, and Dynamics*, 44(8):1440–1453, 2021.

[90] H. Zhang and T. Yu. *Taxonomy of Reinforcement Learning Algorithms*, pages 125–133. Springer Singapore, Singapore, 2020. doi: 10.1007/978-981-15-4095-0_3. URL https://doi.org/10.1007/978-981-15-4095-0_3.

[91] B. D. Ziebart. Modeling Purposeful Adaptive Behavior with the Principle of Maximum Causal Entropy. pages 45–49, 2018. doi: 10.1184/R1/6720692.v1. URL https://kilthub.cmu.edu/articles/thesis/Modeling_Purposeful_Adaptive_Behavior_with_the_Principle_of_Maximum_Causal_Entropy/6720692.

[92] G. Zou, Y. He, F. R. Yu, L. Chen, W. Pan, and Z. Ming. Multi-constraint deep reinforcement learning for smooth action control. In L. D. Raedt, editor, *Proceedings of the Thirty-First International Joint Conference on Artificial Intelligence, IJCAI-22*, pages 3802–3808. International Joint Conferences on Artificial Intelligence Organization, 7 2022. doi: 10.24963/ijcai.2022/528. URL `https://doi.org/10.24963/ijcai.2022/528`. Main Track.

[93] İbrahim Yazici, I. Shayea, and J. Din. A survey of applications of artificial intelligence and machine learning in future mobile networks-enabled systems. *Engineering Science and Technology, an International Journal*, 44:101455, 2023. ISSN 2215-0986. doi: https://doi.org/10.1016/j.jestch.2023.101455. URL `https://www.sciencedirect.com/science/article/pii/S2215098623001337`.

A | Baseline Guidance Algorithm

The baseline guidance algorithm of the ReFEx mission is designed to perform two main tasks: a major trajectory update to correct the reference trajectory accounting for the actual initial conditions [64], and regular minor trajectories updates [65], to be performed during the atmospheric reentry phase.

Major Trajectory Update:
This algorithm and related analyses are described in [64] and summarized in this section. Several flight constraints need to be enforced during the mission and they can be divided int two main groups: first, constraint that ensure the *safety* of the vehicle and second, constraints that ensure the *controllability* of the vehicle. The first constraint group, including load factor, dynamic pressure, heath flux and heat load, has been accounted during the mission and vehicle design phase, therefore can be considered as implicitly satisfied when it comes to compute the major trajectory update.
On the other hand, the second group is composed by three main constraints:

1. Limitation in the angular rate;
2. Limitation in the angular acceleration;
3. Imposition of a flight-corridor in the *Mach*-α domain.

It is necessary to define a flight corridor since there are flight conditions in which the vehicle is underactuated, therefore the flight corridor profile results from a proper analysis of the trimmability of the Re-Entry segment within the feasible flight envelope as can be seen in Figure 4.8. The definition of this flight corridor imposes the *roll-manoeuvre* (already described in section 4.2) to be performed between $M = 2$ and $M = 1.5$ in order to keep ReFEx within a safe flight envelope.

Since the trajectory update must be computed by the on-board software, its computational cost is one of the most critic requirements; solving an optimal control problem is therefore unfeasible while on the other hand developing a new solver was out of the scope of the mission. The adopted solution therefore features a *reduced* optimal control problem

in order to avoid dealing with the full complexity of the nominal one.
The state vector of the optimal control problem is $\boldsymbol{x}(t) = [r,\ \phi,\ \lambda,\ v_G,\ \gamma_G,\ \chi_G,\ \mu,\ \alpha,\ \dot{\mu},\ \dot{\alpha}]$ that are respectively: distance from the Earth center, latitude, longitude, speed, flight path angle and heading referred to the Center of Mass (CoM) (expessed in GEOC with respect to the ECEF reference frame), bank angle, attack angle and respective time derivatives. The control vector is instead defined as: $\boldsymbol{u}(t) = [\ddot{\mu},\ \ddot{\alpha}]$ while the cost function to be minimized consists in meeting a prescribed final condition expressed in position and velocity.

The idea behind the reduced problem consists in reducing the control vector from a 2-D continuous time signal to a discrete set of parameters $\hat{u}$ such that:

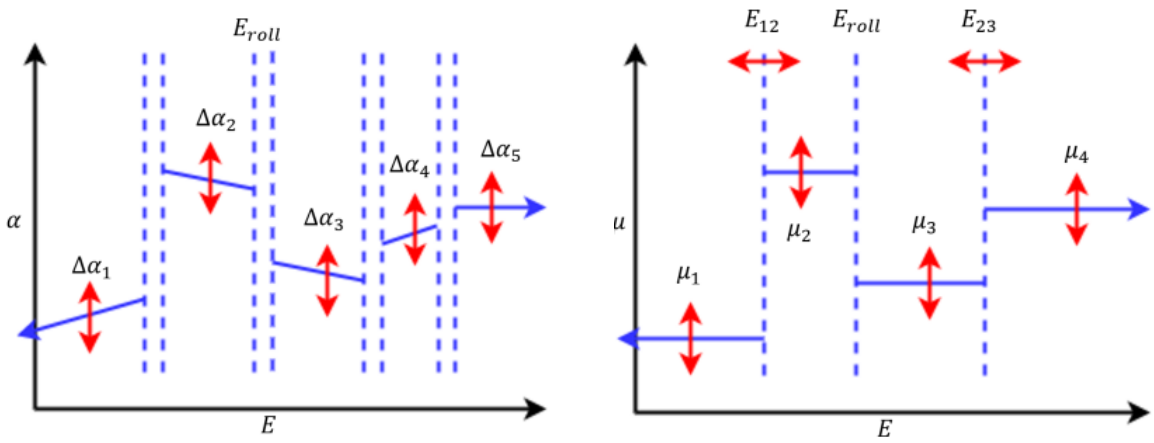
$$\boldsymbol{u}(t) = \boldsymbol{u}(\hat{u}, \boldsymbol{x}(t)) \tag{A.1}$$

where in this case, the state vector is $\boldsymbol{x}(t) = [r,\ \lambda,\ \phi,\ v,\ \gamma,\ \chi]$ and the control vector is $\boldsymbol{u}(t) = [\mu,\ \alpha]$.
In the *reduced* framework, the bank angle μ profile is divided in 4 independent intervals defined in the energy domain (see Equation A.2), within each interval, the bank angle profile is constant, leading to 4 optimization parameters, a schematic representation is given in Figure A.1b.

$$E = E_{\text{kinetic}} + E_{\text{potential}} = \frac{v^2}{2} - \frac{\mu_{\oplus}}{r} \tag{A.2}$$

where $\mu_{\oplus}$ is the Earth standard gravitational parameter.

The angle of attack α is treated similarly: an initial profile in the energy domain is given based on the nominal trajectory, the profile is then divided in 5 intervals in the energy domain with buffers between each other, in each buffer a constant offset is added to the nominal profile as can be seen in Figure A.1a. In this case, the interval boundaries are fixed.
Finally the energy value named E_{roll} is fixed in the energy domain and defines the moment in which performing the roll maneuver, passing from the *belly-up* to the *belly-down* configuration.



(a) Angle of Attack discretization (b) Bank angle discretization

Figure A.1: Control parameters discretization [64]

The resulting parameterized control vector is the following:

$$\hat{\boldsymbol{u}} = [E_{12},\ E_{23},\ \mu_1,\ \mu_2,\ \mu_3,\ \mu_4,\ \Delta\alpha_1,\ \Delta\alpha_2,\ \Delta\alpha_3,\ \Delta\alpha_4,\ \Delta\alpha_5] \tag{A.3}$$

At each iteration, once the control parameters $\hat{\boldsymbol{u}}$ are available, the trajectory is propagated based on its initial state using the computed control parameters, and its terminal condition is based on an evaluation of the specific energy.
The constraints in angular rates, angular accelerations and angle of attack are imposed directly in this phase.
Finally the control parameters are evolved in order to minimize the cost function formalized as difference between the actual terminal point and the target one.
It has been noticed that for a set of initial conditions, it is not possible to find a proper trajectory to match the prescribed final condition; these points have been properly analyzed and classified: when the initial condition belongs to a point in this set, the target is automatically moved to a second option which has been proven to be reachable from each point belonging to the aforementioned set.
The run time for this algorithm is shown in [63], and is considered acceptable since it is executed during the exoatmospheric coasting phase.

Periodic Trajectory Update:
This algorithm and related analyses are described in [65] and summarized in this section. As already mentioned, following the initial trajectory update is still not sufficient to ensure the satisfaction of the terminal requirements due to errors in the modeling, navigation

and control, it is necessary to perform an online correction of the first trajectory update. This process is done in a similar way with respect to the *major update*, also in this case, the optimal control problem is transcripted in a reduced version where the control vector is transformed from a 2D continuous signal to a discrete set of parameters $\hat{\boldsymbol{u}}$.

The updating algorithm, when computing the new trajectory, divides the control horizon in three equal intervals (i.e. from the current timestep to the final condition), and in each time interval, it prescribes a shift in the angle of attach α and in the bank angle σ with respect to the nominal value that comes either from the initial *Major Trajectory Update* or from the current optimization itself. These variation are represented by the letter δ. The purpose of applying these shifts in the whole time horizon is both to determine the immediate control action to be demanded to the control system, and to determine the parameters to be used in the trajectory propagation needed to evaluate the final cost function. The optimization variables are slightly different from the one in the previous paragraph: in this case the $\hat{\boldsymbol{u}}$ set includes:

- The change in the bank angle with respect to the nominal profile in each time interval ($\delta\mu_1,\ \delta\mu_2,\ \delta\mu_3$);
- The global angle of attack variation with respect to the nominal profile to be applied until the control horizon is reached $\delta\alpha$;
- The bank angle μ in the two last segments in the corresponding energy intervals shown in Figure A.1b (μ_3, μ_4);
- A combined parameter accounting for the angle of attack α variation in the last two segments $\Delta\alpha_{4-5}$;

the parameterized control vector will therefore be the following:

$$\hat{\boldsymbol{u}} = [\delta\mu_1,\ \delta\mu_2,\ \delta\mu_3,\ \delta\alpha,\ \mu_3,\ \mu_4,\ \Delta\alpha_{4-5}] \tag{A.4}$$

After the current iteration on $\hat{\boldsymbol{u}}$ is available, the algorithm propagates the trajectory until the specific-energy final condition is met by imposing the path constraints during the propagation itself as in the previous case.
The objective is again to minimize the error with respect to the target final condition but in this case, differently from before, each update step is computed by running the algorithm for a single iteration using as initial guess the previous result but with a relatively high frequency (0.5 Hz). This allows in every case to improve the previous solution, also when the single iteration of the optimization routine cannot reach the prescribed tolerance.
The purpose of this algorithm is to guide ReFEx from the EI the EoE which is reached

at around 6 km in altitude, at this point the main requirement of the mission has been fulfilled [65]. The execution time was also verified in [63].

B | Reference Frames

This appendix provides a description of all the reference frames that have been used in this study. Five reference frames have be considered during the work: Geocentric (GEOC), Geodetic (GEOD), North-East-Down (NED), Earth Centered Earth Fixed (ECEF) and Earth Centered Inertial (ECI).

Geocentric Reference Frame

It is a coordinate reference system centered at the Earth's center of mass, describes the position of the object using:

- h: altitude with respect to the Earth surface, by considering the Earth a perfect sphere. It is computed as $h = r - R_{\oplus}$ where r is the distance from the Earth's center and $R_{\oplus}$ is the equatorial radius of the Earth;
- ϕ: latitude, angle measured from the Earth's equatorial plane, positive northward;
- λ: longitude, angle measured eastward from the Greenwich meridian.

When using the GEOC reference frame, the velocity is represented using the following variables:

- v: norm of the velocity vector;
- γ: flight path angle, angle between the local horizon and the velocity vector. Describes if the vehicle is ascending or descending;
- χ: heading angle, angle between north and the component on the local horizon of the velocity vector. Describes in which direction the aircraft is moving with respect to the cardinal directions.

Geodetic Reference Frame

It is used to describe the position on or near the Earth's surface, accounting for the planet's shape as an oblate spheroid rather than a perfect sphere. When expressed in spherical coordinates, it includes:

- h_{GEOD}: geodetic altitude, the height above the reference ellipsoid (not from the Earth's center);
- ϕ_{GEOD}: geodetic latitude, the angle between the local normal to the ellipsoid and the equatorial plane. Unlike geocentric latitude, this considers Earth's flattening;
- λ_{GEOD}: longitude, the angle measured eastward from the Greenwich Meridian.

It differs from the GEOC reference frame because the latter considers the Earth as a perfect sphere. In general, the GEOD reference frame approximate much better the real Earth shape and the object position with respect to it than GEOC.

NED Reference Frame

Differently from the previous, the NED Reference frame is defined using a cartesian coordinate system. The three axis represent:

- *North* (x-axis): the axis pointing toward the true north, parallel to the local meridian;
- *East* (y-axis): the axis pointing toward the true east, perpendicular to the north axis and parallel to the local horizon;
- *Down* (z-axis): the axis pointing downward towards the center of the Earth, perpendicular to the horizontal plane formed by the North and East axes.

Within the simulator, it is mostly used to represent the velocity components on a specific position, it is therefore a local reference frame centered on the vehicle CoM.

Earth Centered Earth Fixed Reference Frame

The Earth-Centered Earth-Fixed ECEF Reference Frame is a Cartesian coordinate system used to describe positions relative to the Earth, where the origin is at the Earth's center of mass and the axes are fixed relative to the Earth's surface and they are defined as:

- *X-axis*: points toward the intersection of the Earth's equator and the Prime Meridian (0° longitude). This is the reference direction for longitude;
- *Y-axis*: points 90° east of the X-axis, tangent to the Earth's equator;
- *Z-axis*: points towards the North Pole, aligned with the Earth's rotational axis.

Since the coordinate system is Earth-fixed, the axes rotate along with the Earth while it spins.

Earth Centered Inertial Reference Frame

The ECI Reference Frame is an inertial frame meaning that it does not rotate with the Earth as the ECEF does. As the ECEF it is originated on the center of mass of the Earth and its axis are defined as:

- *X-axis*: points toward the intersection of the equatorial plane and the vernal equinox, essentially fixed in space and aligned with the distant stars. This is often used as the reference direction for the longitude of ascending nodes;
- *Y-axis*: completes the right-handed system by being perpendicular to the X-axis, in the plane of the equator. This axis also lies within the equatorial plane;
- *Z-axis*: points toward the north celestial pole, essentially aligned with the Earth's rotational axis (the Earth's spin axis), but fixed relative to the stars, not the Earth.

Transformation from cartesian to spherical reference frames

Since the dynamics is formulated in the GEOC reference frame, it is often useful to transform a vector expressed in a cartesian reference frame to its corresponding in the spherical one. This procedure must be defined similarly also for the derivative of a vector.
Lets define with $\mathbf{x}$ a vector expressed in the cartesian reference frame, with $\mathbf{dx}$ the derivative of a vector in a cartesian reference frame, with $\mathbf{s}$ a vector in the spherical reference frame and with $\mathbf{ds}$ its derivative.
The following transformations are valid:

$$\begin{cases} s_1 = ||\mathbf{x}|| \\ s_2 = atan\left(\dfrac{-x_3}{\sqrt{x_1^2 + x_2^2}}\right) \\ s_3 = atan\left(\dfrac{x_1}{x_2}\right) \end{cases} \tag{B.1}$$

Notice that in this case it is important to perform a further step to disambiguate the value result of the arc tangent.
The derivative of the vector can instead be computed as:

$$\begin{cases} ds_1 = \mathbf{dx} \cdot \dfrac{\mathbf{x}}{||\mathbf{x}||} \\ ds_2 = \dfrac{dx_1 x_1 + dx_2 x_2}{x_{hor}} \cdot \dfrac{x_3}{||\mathbf{x}||^2} - \dfrac{dx_3 x_{hor}}{||\mathbf{x}||^2} \\ ds_3 = \dfrac{dx_2 x_1}{x_{hor}^2} - \dfrac{dx_1 x_2}{x_{hor}^2} \end{cases} \tag{B.2}$$

where x_{hor} refers to the norm of the horizontal component of the $\mathbf{x}$ vector which is computed as $x_{hor} = \sqrt{x_1^2 + x_2^2}$.

Rotation from ECEF to NED

It is here reported the rotation matrix to convert a cartesian vector from the ECEF reference frame to the NED reference frame since it will be useful later.

$$R_{ECEF \to NED} = \begin{bmatrix} -sin\phi\ cos\lambda & -sin\phi\ sin\lambda & cos\phi \\ -sin\lambda & cos\lambda & 0 \\ -cos\phi\ cos\lambda & -cos\phi\ sin\lambda & -sin\phi \end{bmatrix} \tag{B.3}$$

C | Neural Network and Genetic Programming-law implementation details

This appendix details how to extract the guidance law from the training environment (i.e. Python-based) and export it in a format which is suitable for its evaluation in the testing environment (i.e. the 6DOF Simulink simulator). In second place, it reports the methodologies and results related to the tests done to validate the program implementation in the testing environment with respect to its behavior in the training one.

Export

Concerning the GP, the export phase is straightforward since the output of the training is a simple symbolic guidance law that can be expressed as a string and copied as line of code in the testing simulator. The only aspect to be considered in this phase is to define the primitive set in both the environments in the same way, but also in this case, this process consists in defining very simple functions which is done manually by coding their expression.

In contrast, the procedure for handling the RL output—namely, the trained NN—is significantly more complex. This increased complexity stems from two main factors:

- A NN is inherently more intricate than a GP-based guidance law. For proper evaluation, it is preferable to use the same software environment employed during training or an automatic export interface. This approach minimizes the risk of introducing discrepancies or errors during the export process.
- Because the goal is to use automated export tools, ensuring compatibility between the Python libraries used during training and their MATLAB counterparts played a critical role in the implementation of the complete pipeline.

To address these challenges, the network was implemented and trained using the PyTorch

framework within the pyrlprob and Ray RLlib environments. After training, the model is configured to remove the stochastic component making the policy deterministic. In this way the output of the network is no more the probability of taking an action from a distribution, but the expected value of the action sampled from that distribution. The model is represented using the Ray framework in a .pt file, which is a general PyTorch-compatible format independent of the training environment. This version is used to run the 3DOF simulations. To enable inference in Simulink, the network is additionally exported as an .onnx file using the Ray framework. This format can be imported into MATLAB using a dedicated toolbox available in version R2021b. At this point, the network can be seamlessly integrated into the 6DOF simulator for online evaluation.

Validation

To ensure that the Simulink implementation is coherent with the Python one, the intermediate model shown in Figure C.1 has been designed. It is a Simulink implementation of the standalone guidance law (either GP or RL based) which returns the correction command and the tracking errors defined as in Equation 5.1 taking as input the vehicle GEOC state vector. The validation process represented in Figure C.1 consists in five steps:

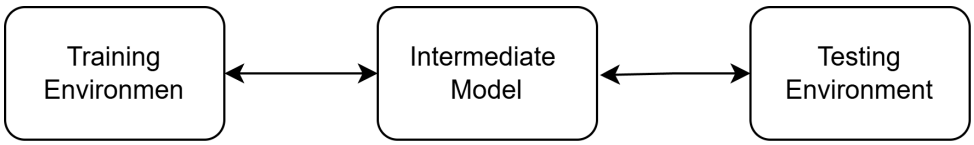
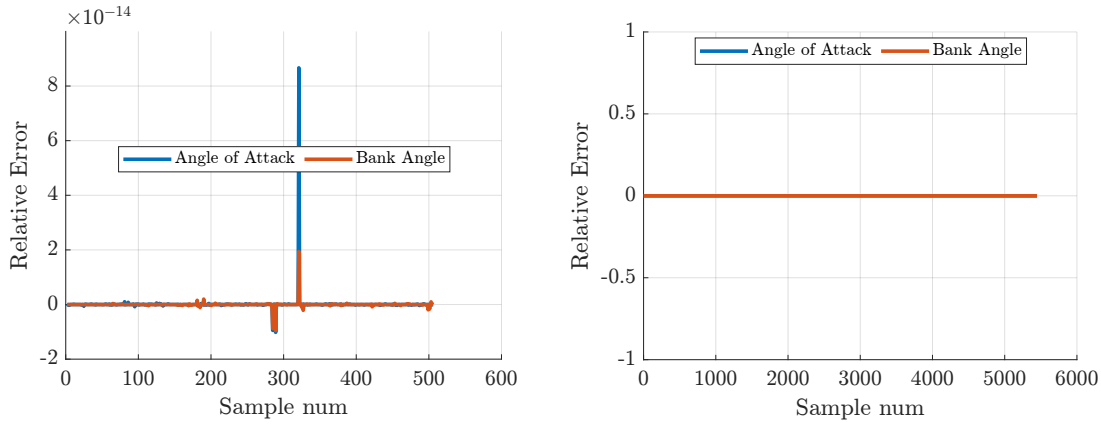


Figure C.1: Functional scheme of the validation architecture

1. Perform a test run in Python and log the GEOC state vector, tracking errors and correction command at each iteration step;
2. Perform a test run in Simulink and log the same variables as Step 1;
3. Use the intermediate model to compute the tracking errors and the correction command given the GEOC state vector computed in Step 1. Check if tracking errors and correction command returned by the intermediate model match those logged at Step 1;
4. Use the intermediate model to compute the tracking errors and the correction command given the GEOC state vector computed in Step 2. Check if tracking errors and correction command returned by the intermediate model match those logged at Step 2;

5. If both Step 3 and Step 4 are successfully verified, the implemented guidance law in the Simulink and Python environment are validated.

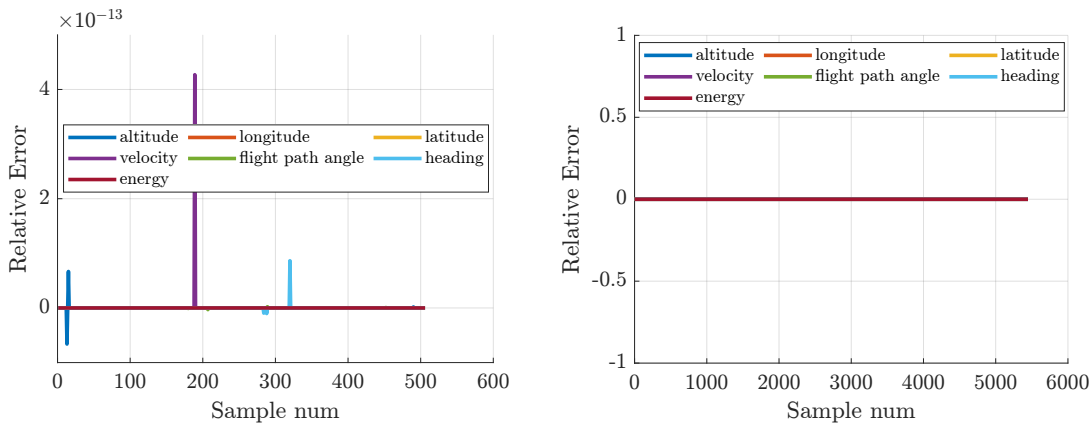
The intermediate model has been introduced to not modify the simulators and then simplify the full testing process The results of this validation are presented in the following figures. For both the GP and RL cases, the plots show the validation of both the preprocessing phase and the actual guidance command computation.



(a) Training environment wrt Intermediate Model

(b) Testing environment wrt Intermediate Model

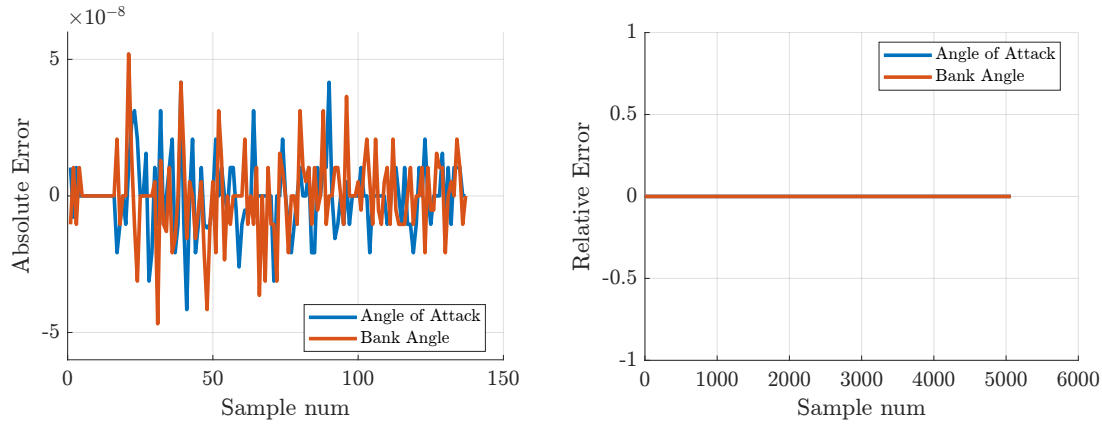
Figure C.2: GP-Law corrective command



(a) Training environment wrt Intermediate Model

(b) Testing environment wrt Intermediate Model

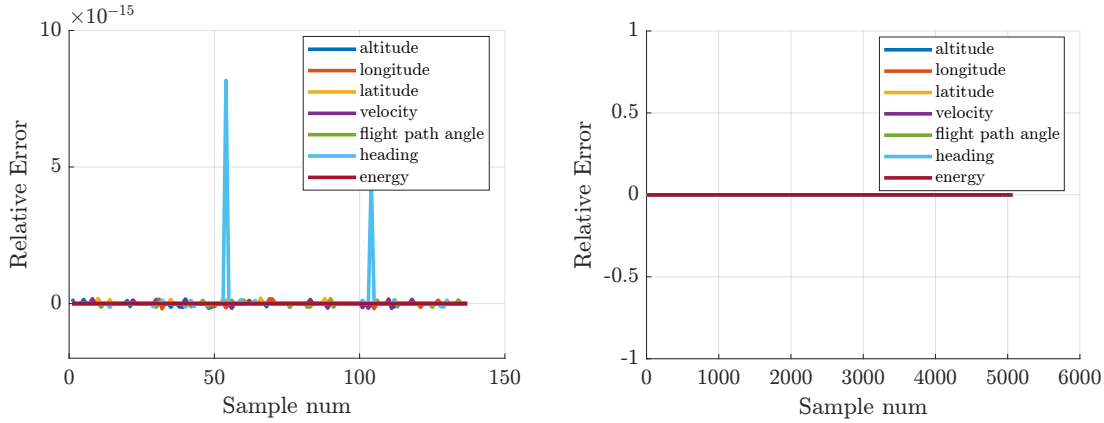
Figure C.3: GP tracking errors



(a) Training environment wrt Intermediate Model

(b) Testing environment wrt Intermediate Model

Figure C.4: RL NN corrective command



(a) Training environment wrt Intermediate Model

(b) Testing environment wrt Intermediate Model

Figure C.5: RL tracking errors

As can be seen, from Figure C.2b, C.3b, C.4b, C.5b, the intermediate model is exactly consistent with respect to the testing environment: this is reasonable since both of them are implemented on Simulink and they use the same functions, in any case this test is necessary to state the reliability of the intermediate model itself.

On the other hand in Figure C.2a, C.3a, C.4a, C.5a we can see the actual differences between the training and the testing environment, from the errors order of magnitude it is possible to state that the only differences are linked to numerical differences between the two implementations (i.e. Python and Simulink), in particular the relatively high error present in Figure C.4a is caused by a truncation error due to the output of the

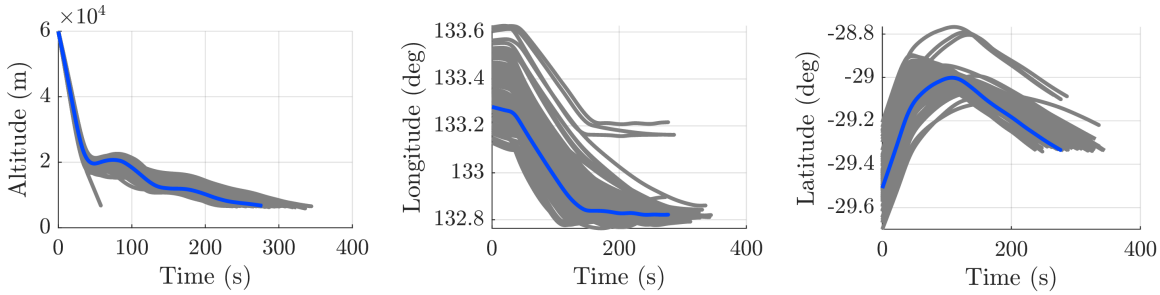
NN being represented in single precision in Python, and in double precision in Simulink. However, the errors are considered acceptable and the performances consistent.

D | Flight Simulations Results

The current appendix contains the complete 6DOF results (500 Montecarlo runs) of the most relevant guidance configurations, namely the baseline algorithm, the **60UC Smooth** RL configuration and the GP **EVOL 13**. For each configuration, the altitude, longitude and latitude in the GEOD reference frame, together with the norm of the velocity, the flight path angle, the heading angle and the guidance signals (α and μ) between the EI and EoE are here reported.

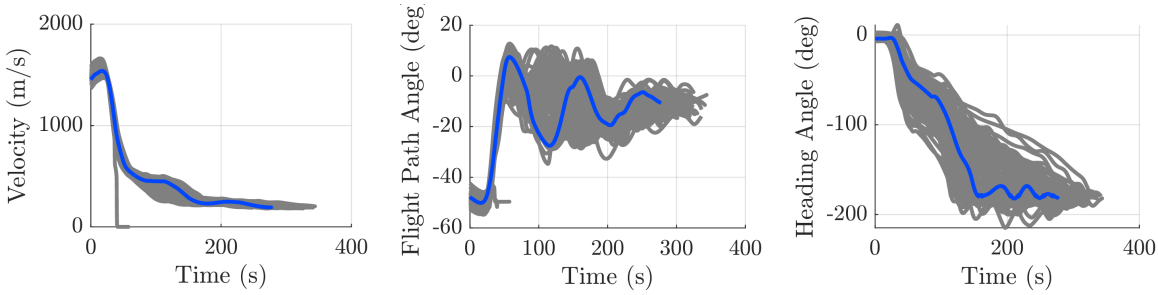
In the following plots, one example run has been highlighted in blue, while all the other 499 runs are depicted in gray. Among these, those that strongly deviate from the general behavior are unstable ones.

D.1. Baseline Results



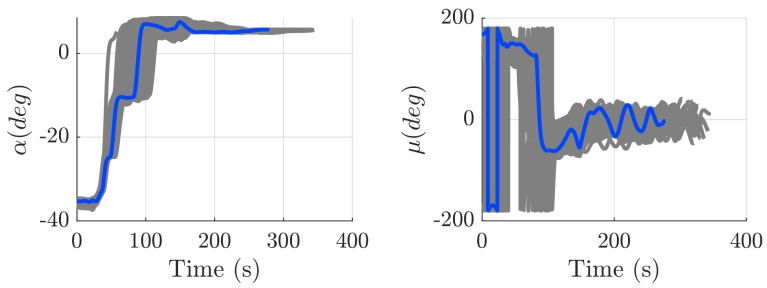
(a) Altitude (b) Longitude (c) Latitude

Figure D.1: Baseline guidance, positional states



(a) Velocity (b) Flight Path Angle (c) Heading Angle

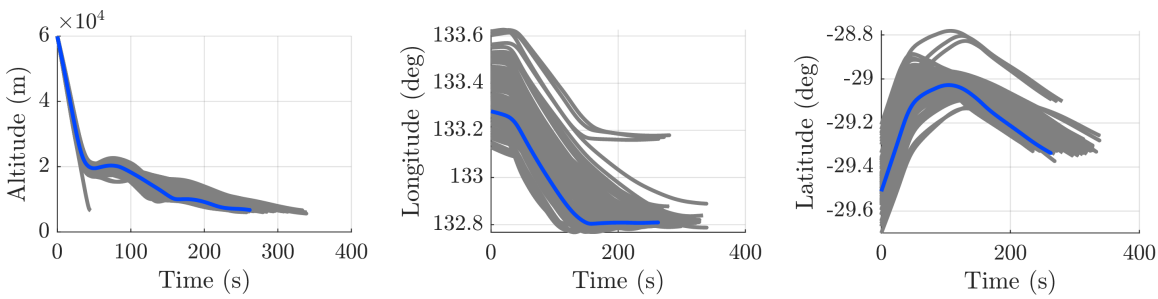
Figure D.2: Baseline guidance, velocity states



(a) Angle of Attack (b) Bank Angle

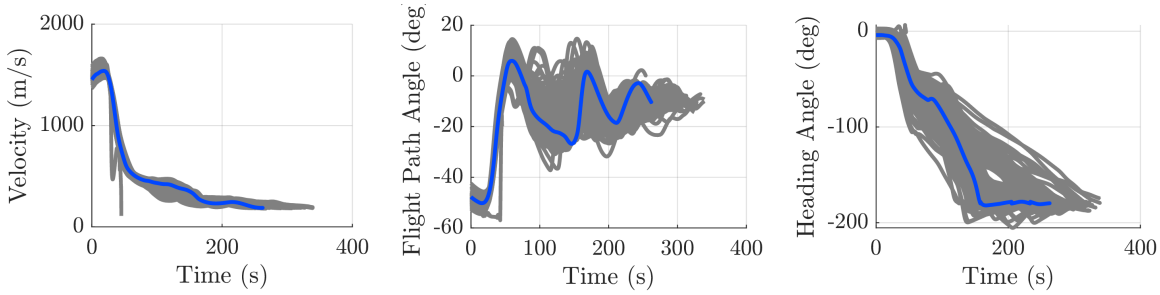
Figure D.3: Baseline guidance, guidance signals

D.2. RL 60UC Smooth Results



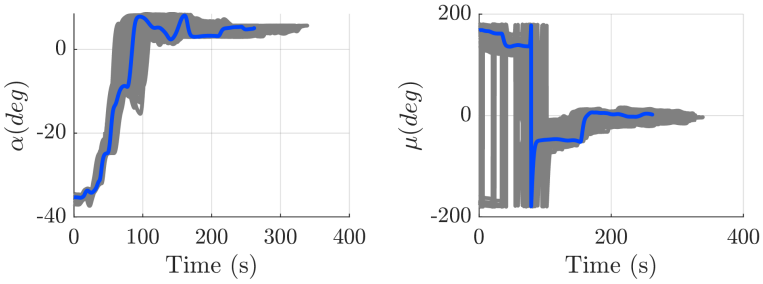
(a) Altitude (b) Longitude (c) Latitude

Figure D.4: RL **60UC Smooth** guidance, positional states



(a) Velocity (b) Flight Path Angle (c) Heading Angle

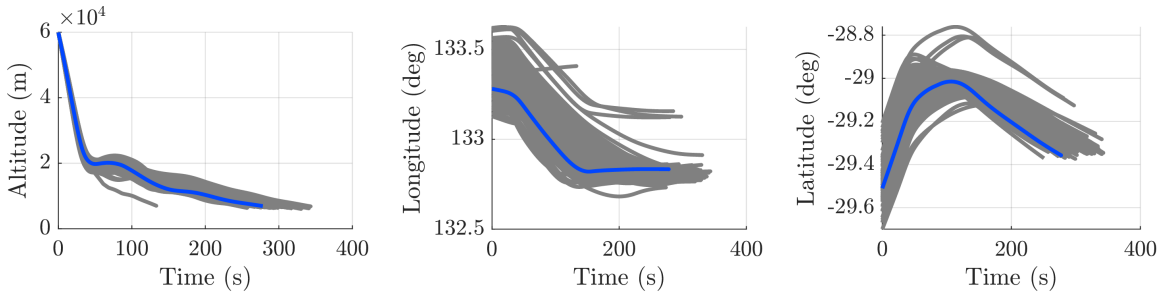
Figure D.5: RL **60UC Smooth** guidance, velocity states



(a) Angle of Attack (b) Bank Angle

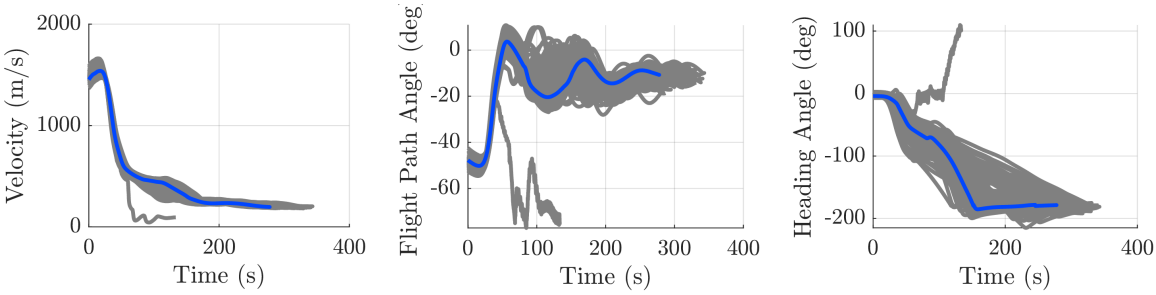
Figure D.6: RL **60UC Smooth** guidance, guidance signals

D.3. GP EVOL 13 Results



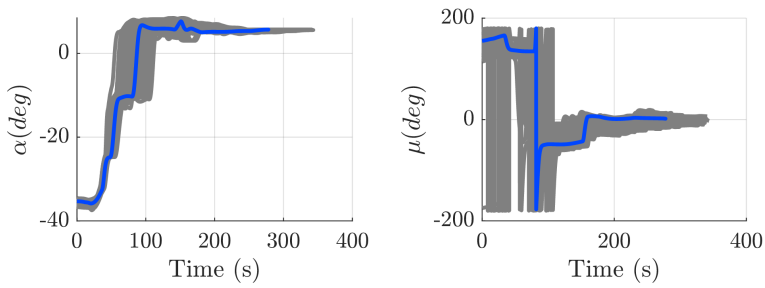
(a) Altitude (b) Longitude (c) Latitude

Figure D.7: GP **EVOL 13** guidance, positional states



(a) Velocity (b) Flight Path Angle (c) Heading Angle

Figure D.8: GP **EVOL 13** guidance, velocity states



(a) Angle of Attack (b) Bank Angle

Figure D.9: GP **EVOL 13** guidance, guidance signals

List of Figures

2.1 PPO clipped objective function [72] 22
2.2 Neural Network Model 23

3.1 Depiction of a GP individual corresponding to the function $8x - (y + 5)$. . 29
3.2 Graphical representation of mutation and crossover operations 33

4.1 ReFEx launch Configuration (left), section view of the Re-Entry segment ReFEx (right) [4] 39
4.2 Mission architecture and flight events [4] 41
4.3 Training Simulator Architecture 42
4.4 Atmosphere characterization - the blue line represents an illustrative instance 43
4.5 Wind components - in blue an illustrative instance 44
4.6 ReFEx mass distribution 44
4.7 Navigation model validation 49
4.8 Trimmability of ReFEx with respect to α and M 52
4.9 Simplified functional architecture 52
4.10 DKE conceptual architecture 54
4.11 GCC conceptual architecture: $\boldsymbol{\varphi}$ is the set of guidance commands α and μ, $\mathbf{x}$ is the GEOC state vector, $\boldsymbol{\omega}_{cmd}$ is the commanded angular rate vector and $\mathbf{M}_{cmd}$ is the commanded torque 55
4.12 High level GNC system structure [62] 56

5.1 ReFEx state at the beginning of the simulation 63
5.2 GP training dataset 64
5.3 RL training dataset 65
5.4 GP training saturation functions 72

6.1 Experiment pipeline 80
6.2 ReFEx states error at the EoE 82
6.3 6DOF trajectories with the baseline guidance strategy, 100 runs, GEOD reference frame 83

6.4 Fitness value of the best individual at each generation 86
6.5 **EVOL 4**, Corrective commands, 3DOF environment - in blue an example run 88
6.6 **EVOL 13**, Corrective commands, 3DOF environment - in blue an example run 88
6.7 Flyable region, 3DOF environment - in blue an example run 89
6.8 Longitude and Latitude relative errors at the EoE, 3DOF environment . . 89
6.9 Dispersion at the EoE, IGP, 6DOF environment 91
6.10 **EVOL 4**, 6DOF trajectories, 500 runs, GEOD reference frame 91
6.11 **EVOL 13**, 6DOF trajectories, 500 runs, GEOD reference frame 92
6.12 Flyable region, 6DOF trajectories, 500 runs 92
6.13 **EVOL 4**, Corrective commands, 6DOF environment - in blue an example run 93
6.14 **EVOL 13**, Corrective commands, 6DOF environment - in blue an example run 93
6.15 Individual size with respect to flight performances 94
6.16 Reward value during the training 97
6.17 **60UC Standard**, Corrective commands, 3DOF environment - in blue an example run 98
6.18 **60UC Standard**, Corrective commands, 6DOF environment - in blue an example run 98
6.19 **60UC Smooth**, Corrective commands, 3DOF environment - in blue an example run 99
6.20 **60UC Smooth**, Corrective commands, 6DOF environment - in blue an example run 99
6.21 RL experiments performances in the 6DOF environment 102
6.22 **60UC Smooth**, 6DOF environment, 500 runs 103
6.23 **60UC Smooth**, 6DOF environment, Guidance Signal - in blue an example run 103
6.24 Dispersion at the EoE, RL, 6DOF environment 104
6.25 State errors distribution at EoE comparison, 500 runs, 6DOF environment 105
6.26 Total Error, 500 runs, 6DOF environment 106
6.27 Experiments with the highest and lowest JS divergence value 111

A.1 Control parameters discretization [64] 135

C.1 Functional scheme of the validation architecture 144
C.2 GP-Law corrective command 145

C.3 GP tracking errors 145
C.4 RL NN corrective command 146
C.5 RL tracking errors 146

D.1 Baseline guidance, positional states 149
D.2 Baseline guidance, velocity states 150
D.3 Baseline guidance, guidance signals 150
D.4 RL **60UC Smooth** guidance, positional states 150
D.5 RL **60UC Smooth** guidance, velocity states 151
D.6 RL **60UC Smooth** guidance, guidance signals 151
D.7 GP **EVOL 13** guidance, positional states 151
D.8 GP **EVOL 13** guidance, velocity states 152
D.9 GP **EVOL 13** guidance, guidance signals 152

List of Tables

6.1 Target states and absolute tolerance 81
6.2 Baseline Guidance Performances 81
6.3 IGP parameters 84
6.4 GP training database generation settings 85
6.5 GP fitness and penalty functions parameters 85
6.7 IGP performances on the 3DOF training environment. (*): evolutions which terminate without solving each uncertainty case (**): selected for the 500 runs 6DOF Monte Carlo campaign 87
6.8 IGP performances on the testing environment. (*): evolutions which terminate without solving each uncertainty case (**): selected for the 500 runs 6DOF Monte Carlo campaign 90
6.9 Comparison between GP-law number of nodes and guidance performances 94
6.10 RL experiments variable parameters 95
6.11 RL training fixed parameters 96
6.12 RL reward signal parameters 96
6.13 RL training database generation settings 96
6.14 RL experiments performances recap 100
6.15 RL experiments performances on the 500 runs Monte Carlo campaign . . . 102
6.16 Guidance performances comparison, 500 runs, 6DOF environment 105
6.17 RL and GP performances using the same training dataset 107
6.18 Training computational resources 108
6.19 Order of magnitude of the expected average computational time to generate the guidance command 109
6.20 Jensen-Shannon divergence for RL experiments. Lower means better generalization 110
6.21 Jensen-Shannon divergence for GP experiments. Lower means better generalization 110

List of Symbols

Variable	Description	SI unit
x	GEOC state vector	m, m/s, rad
x^{rel}	tracking errors vector	m, m/s, rad
x^{ref}	reference trajectory GEOC state vector	m, m/s, rad
x_T	target GEOC state vector	m, m/s, rad
r	ECEF position vector	m
h	altitude	m
λ	longitude	rad
ϕ	latitude	rad
v	velocity	m/s
γ	flight path angle	rad
χ	heading angle	rad
E	specific energy	m^2/s^2
α	angle of attack	rad
μ	bank angle	rad
$\delta\alpha$	angle of attack correction	rad
$\delta\mu$	bank angle correction	rad
$e_{EoE,i}$	i component relative error at the EoE	-
e_{EoE}	norm of the relative error at the EoE	-
ϵ_i	i component absolute tolerance at the EoE	m, m/s, rad
R	Reward Signal	-
θ	general parameter, usually refers to α or μ	rad
L	Lift	N
D	Drag	N
Ma	Mach number	-
$(\hat{\cdot})$	Normalized component	-
JS	Jensen Shannon divergence	-

Acronyms

AI Artificial Intelligence

ML Machine Learning

DOF Degrees Of Freedom

HGV Hypersonic Gliding Vehicle

UAV Unmanned Aerial Vehicle

RMSE Root Mean Square Error

ODE Ordinary Differential Equation

CoM Center of Mass

RNG Random Number Generator

DLR Deutsches Zentrum für Luft- und Raumfahrt e.V.

IGRF International Geomagnetic Reference Field

RCS Reaction Control System

CFD Computational Fluid Dynamics

IMU Inertial Measurements Unit

FADS Flush Air Data System

PWM Pulse With Modulation

PDF Probability Density Function

PCC Pearson Correlation Coefficient

GNSS Global Navigation Satellite System

ESA European Space Agency

UC Uncertainty Case

RAM Random Access Memory

NLP Nonlinear Programming

SQP Sequential Convex Programming

RL Reinforcement Learning

DRL Deep Reinforcement Learning

MDP Markov Decision Process

POMDP Partially Observable Markov Decision Process

GNC Guidance Navigation and Control

DRL Deep Reinforcement Learning

NN Neural Network

DNN Deep Neural Network

PPO Proximal Policy Optimization

DQN Deep Q Network

SAC Soft Actor Critic

TRPO Trust Region Policy Optimization

DDPG Deep Deterministic Policy Gradient

TD3 Twin Delayed Deep Deterministic Policy Gradient

MSBE Mean-Squared Bellman Error

LSTM Long short-term memory

RNN Recurrent Neural Network

CNN Convolutional Neural Network

IID Independent and Identically Distributed

MLP Multilayer Perceptron

GP Genetic Programming

GA Genetic Algorithm

ADF Automatically Defined Functions

EA Evolutionary Algorithms

IGP Inclusive Genetic Programming

SGP Standard Genetic Programming

ES Evolutionary Strategy

GC Guidance and Control

GNC Guidance Navigation and Control

MIMO Multi Input Multi Oputput

ReFEx REusability Flight EXperiment

EoE End of Experiment

EI Entry Interface

RCS Reaction Control System

RLV Reusable Launch Vehicle

VTHL Vertical Take-off Horizontal Landing

VTVL Vertical Take-off Vertical Landing

ECI Earth Centered Inertial

ECEF Earth Centered Earth Fixed

NED North-East-Down

GEOC Geocentric

GEOD Geodetic

RF Reference Frame

HWM Horizontal Wind Model

GNC Guidance Navigation and Control

GCC Guidance and Control Computer

HNS Hybrid Navigation System

DKE Dynamics and Kinematics Environment

MVM Mission Vehicle Management

Acknowledgements

This thesis marks the conclusion of an intense yet deeply rewarding chapter. The months I spent in Bremen offered a meaningful opportunity for growth, both personally and professionally, I truly enjoyed every moment in this city. I want to thank Professor Francesco Topputo, whose classes sparked my interest in these topics and ultimately led me to apply for this position.

I am sincerely grateful to my supervisor here in DLR, Francesco Marchetti. Your insightful guidance, passion for the subject, and constant support, especially during the moments when I was doubting myself, played a crucial role in helping me navigate this journey. Working with you has truly been an honor and pleasure.

I would also like to thank the entire team of the GNC department at the Institute of Space Systems for welcoming me and creating such a comfortable and motivating work environment. It's not an exaggeration to say I was happy to come to work each day. I'm especially grateful to Jose Redondo, your technical advice and experience were essential to the successful development of this project.

Above all, I want to thank my family. Alessandra, Mamma and Papà: you have always been my strongest supporters, without your education, your constant love and encouragement, none of this would have been possible.

A special thank you goes to all my friends from Bremen, Milano, Treviso and Jesolo, your presence has been a constant source of joy and motivation. I feel incredibly lucky to have met you in my life; your kindness, help, and encouragement has truly made a difference.

I must also include Skyward and the people I met in the association, you taught me way more than wat I could even imagine when I first decided to join.

Lastly, I extend my thanks to all those who, even if not explicitly mentioned, have been part of this journey and helped me becoming the person I am now.

Riccardo Cadamuro

Bremen, June 2025