POLITECNICO DI TORINO

MECHANICAL AND AEROSPACE ENGINEERING DEPARTMENT

M.sc Degree in Aerospace Engineering

Master Thesis

Deep learning techniques for micro-launchers branching trajectories optimization

Supervisors
Prof. Nicole Viola
Dr. Jasmine Rimani

Candidate
Alessandro Princi
ID number: s262965

External Supervisor
Ing. Stephan Schuster
ESA'S TEC-MPA

Academic year 2020-2021

Abstract

In this thesis project, the problem of branching trajectories analysis for micro-launchers proposed by the ESA'S TEC-MPA section is developed. The growing interest of companies, agencies and training institutions in the small satellite market makes these launch systems, compared to other solutions, much more suitable and functional for this type of missions both in economic and functional terms. Since these are new technologies that are constantly evolving, it is incumbent upon us to study the reliability of these complex systems and to do so from both a component and mission analysis perspective. The interest is therefore to study optimal ascending trajectories in nominal and degraded situations, avoiding, in the second case, corridors that could lead to catastrophic damage both for the generation of debris and for urban protection. The solution developed is presented by Intelligent Rocket Ascent Trajectory Optimizer (IRATO), a tool that, compared to the state of the art of mission analysis software, is able to optimize point trajectories and simultaneously modify them in the presence of operational failures. With respect to the state of the art of mission analysis software, the tool proposes the possibility of inserting failures starting from the nominal trajectory in order to give the user the possibility of evaluating the possible risks deriving from the launch segment when planning the mission architecture. The result is obtained by combining solutions derived from the application of particle swarm optimization and deep learning solution. The first is an algorithm inspired by the movement of flocks of birds which, with respect to its function as an optimizer of subsystems design issues, is used in a dynamics problem. The solution related to deep learning involves a neural network that, instead of the propagation of the launcher after failure, will be able to predict the maximum height reached, thus saving computational cost. The dynamic model and the failures considered are influenced by the simplifications necessary to handle such a problem autonomously and without the support of data provided by private companies. Nevertheless, in order to guarantee to the work the highest possible reliability, the results produced will be compared and validated through the use of ASTOS software.

Acknowledgements

Un ringraziamento speciale va fatto alla Professoressa Nicole Viola per avermi dato la possibilità di intraprendere questo lavoro coinvolgendomi da subito nel suo gruppo di ricerca e per i suoi preziosi consigli durante questo periodo.
Grazie alla mia correlatrice Dr. Jasmine Rimani per avermi guidato con metodicità e costanza in questo lavoro di tesi. Mi auguro che la passione e la dedizione che applichi nel tuo lavoro possano sempre portarti grandi soddisfazioni.
Gre, Gas, Davido, Simo, Narcio e Fus. Sono stati cinque anni meravigliosi passati insieme, ritornassi indietro non cambierei niente di tutto ciò abbiamo fatto.
Mi auguro che possiate vivere delle vite meravigliose.
Grazie ai miei genitori e a mia sorella per il loro immancabile sostegno, per quanti mari mi auguri di vedere sarete sempre il mio porto preferito.
Grazie nonno per aver lottato tanto, questa tesi è dedicata a te.

Acronyms

TVC	Thrust Vector Control
CoM	Centre of Mass
LVLH	Local Vertical Local Horizontal
RLV	Reusable Launch Vehicle
LEO	Low Earth Orbit
FLPP	Future Launchers Preparatory Programme
Q@TS	Quick @ccess To Space
POST-II	Program to Optimize Simulated Trajectories II
RDS	Rocket Dynamic Simulator
ISA	International Standard Atmosphere
ECEF	Earth Centered Earth Fixed
PID	Proportional Integrative Derivative
PSO	Particle Swarm Optimization
SNN	Standard Neural Network
CNN	Convolutional Neural Network
RNN	Recurrent Neural Network
IRATO	Intelligent Rocket Ascent Optimizer
FF	Fuel Fraction
LB	Lower Bound
UB	Upper Bound

DoF	Degree of Freedom
FMECA	Failure Modes, Effects and Criticality Analysis
FTA	Failure Tree Analysis
AST	Office of Commercial Space Transportation
NASA	National Aeronautics and Space Administration
IEEE	Institute of Electrical and Electronics Engineers
RBD	Reliability block diagrams
PCA	Parts Count Analysis
ETA	Event Tree Analysis

Contents

List of Tables 7

List of Figures 8

I First Part **11**

1 Introduction 13
1.1 Rocket trajectory . . . 13
1.2 Reference frames . . . 17
1.2.1 Fixed planetocentric coordinates . . . 17
1.2.2 LVLH reference frame . . . 18

2 Focus on micro-launchers 19

II Second part **23**

3 Problem Statement 25

4 Rocket Dynamic Simulator 27
4.1 Electron Simulink model . . . 30

5 Gradient-based methods 35

6 Particle Swarm optimization 39
6.1 PSO Mathematical Model . . . 42
6.2 Algorithm Validation . . . 46

7 Artifical intelligence 49
7.1 Machine learning . . . 49
7.2 Deep learning . . . 53
7.2.1 Introduction to neural networks . . . 53

8 Problem resolution 59
8.1 Ascent Optimization V1 and V2 . . . 59
8.2 Implementation of branching trajectories . . . 63
8.3 PSO application . . . 65
8.4 Intelligent Rocket Ascent Trajectory Optimizer . . . 67

9 ASTOS Validation 83
9.1 Electron model creation . . . 83
9.1.1 Environment definition . . . 83
9.1.2 Actuators definition . . . 83
9.1.3 Aerodynamics definition . . . 84
9.1.4 Component definition . . . 84
9.1.5 Phase definition . . . 86
9.2 IRATO Validation . . . 87

A Reliability Analysis 91
A.1 Sistem Safety . . . 91
A.2 Reliability analysis methods . . . 94
A.2.1 Reliability Block Diagrams . . . 94
A.2.2 Parts Count Analysis . . . 95
A.2.3 Failure Modes, Effects, and Criticality Analysis . . . 96
A.2.4 Fault tree analysis . . . 98
A.2.5 Event Tree Analysis . . . 100

Bibliography 103

List of Tables

4.1 Electron data 29
7.1 Machine learning algorithms 50
8.1 Logic flow of the second optimization attempt 64
8.2 Electron data for trajectory optimization 68
8.3 Database layout 70
9.1 Variation of resistance coefficient as a function of Mach number . . 84
9.2 1_{st} Stage dimensions 85
9.3 2_{nd} Stage dimensions 85
A.1 RBD basic layouts 94

List of Figures

1.1 ΔV as a function of payload ratio and structural efficiency 17
1.2 Local Vertical Local Horizontal (LVLH) reference frame 18
4.1 Body fixed reference frame 28
4.2 *Electron rocket* block 30
4.3 Height, Velocity and mass consumption as a function of the time . . 32
4.4 3-D trajectory visualization 32
6.1 Graphical representation of a Particle Swarm Optimization (PSO) step executed by an agent 43
6.2 PSO applied to Funcion 6.14 46
6.3 PSO applied to Funcion 6.15 47
7.1 Different representations of the number 3 53
7.2 Sigmoid function 55
7.3 Neural network scheme 56
8.1 Example of fuel consumption optimized trajectory of Ariane V launcher 60
8.2 Optimized trajectory computed through *ascent optimization* V_2 . . 63
8.3 Failure scenario in which at $t = 180$ s there is a 30% deterioration of the maximum η value 65
8.4 Neural network configuration 70
8.5 Neural network scheme 71
8.6 Neural network performance 71
8.7 Validation checks and gradient descent 72
8.8 Regressions 72
8.9 Neural network performance 73
8.10 Validation checks and gradient descent 73
8.11 Regressions 74
8.12 Schematic representation of r_{obst} quantities 75
8.13 Schematic representation of branched problem identification 76
8.14 Heigth, velocity and mass consumption in branched case 77
8.15 Throttle and angle of attack as a function of the number of segments in branched case 78
8.16 Absolute and relative horizontal distance results in branched case . 78
8.17 Heigth, velocity and mass consumption in abort case 79

8.18 Throttle and angle of attack as a function of the number of segments in abort case . . . 79
8.19 Absolute and relative horizontal distance results in abort case . . . 80
9.1 Electron vehicle preview . . . 85
9.2 Nominal trajectory scenario validation . . . 87
9.3 Degraded trajectory scenario validation ($t_{failure} = 100s$ and 30% of throttle upper bound reduction) . . . 88
A.1 System safety process . . . 92
A.2 System safety process . . . 95
A.3 FMECA breakdown structure . . . 97
A.4 Example of FMECA worksheet . . . 97
A.5 Example of FTA scheme . . . 99
A.6 Example of ETA scheme . . . 100

Part I

First Part

Chapter 1

Introduction

In this first part, the elements that make up the branching trajectories problem performed by a launcher are considered. It is in fact regarded in first analysis the trajectory of a launcher and the main types of optimization. This type of analysis is then declined in the class of microlaunchers, among which Electron is considered as an example.

1.1 Rocket trajectory

[1] A rocket launcher is a system capable of carrying a given mass called a payload to a certain orbital altitude thanks to the elementary use of the third law of dynamics . This mass is typically composed of one or more satellites that, following the release, will perform their functions.
Imagining therefore both ascent and re-entry, it is not difficult to figure out how the external conditions to which our launcher is subjected during its flight envelope are different and how every aspect that characterizes must take into account this factor.
In fact, in addition to covering speed ranges from subsonic to hypersonic, it is also necessary to consider the different characteristics of the atmosphere: from the terrestrial one of the launch base to the more rarefied space environment. The assumption of a flat and non-rotating Earth is taken into account to simplify the model that will lead to the derivation of the equations of motion.
Understanding how a rocket works is the first step in being able to derive, or simply understand, the equations that affect it. We know in fact how these systems need a quantity of propellant, that once accelerated will produce a thrust directed towards the opposite direction to the motion that we want to obtain. The quantities required by orbital mechanics to leave the Earth's sphere of influence or to orbit around it are such that they require large quantities of fuel (and consequently large structural masses to contain it). During the ascent phase it is understandable

how, by decreasing the fuel, this structural mass becomes more of an encumbrance than functional to the accomplishment of the mission. Therefore, once the mass of fuel contained in a part of our vehicle (which we will call *stage* from now on) is consumed, it is separated using what is technically called *staging*.

This study start by simply considering our body as point-like mass and associating to it a vector indicating its position $\vec{r}(t)$, then its derivative $\vec{v}(t) = d\vec{r}(t)/dt$. From now on, every discussion will refer to the center of mass of the main body with respect to which the motion occurs, i.e. the Earth.
It's also known how even the mass itself is a function of time due to fuel consumption for this reason, neglecting the external forces considered lower order than the thrust, the second law of dynamics can be written as:

$$m\frac{d\vec{v}}{dt} + \frac{dm}{dt}\vec{v} = \vec{f} \tag{1.1}$$

Where the right term represents the external forces acting on the launcher.

$$\vec{f} = \vec{f_A} + \vec{f_T} + m\vec{g} \tag{1.2}$$

These respectively are:

- Aerodynamic force
- Thrust force
- Weight

Orbit insertion is a process that requires a large amount of energy aimed at increasing both the potential energy (desired height) and kinetic energy (velocity competing with the orbit) of the launcher. The state of the art envisions only this type of aircraft for insertion into orbit of a given payload. The process is relatively simple: an oxidizer and a fuel are placed in their relative tanks. Through a feed system they will come into contact releasing energy that will be channeled in the opposite direction to that towards which it wants to move. In addition to the position it is also necessary to take into account the attitude during the mission because it is responsible for the aerodynamic loads that weigh on my structure, in this regard the angle of attack should be kept as small as possible. While in the past passive stabilization methods were used, nowadays Thrust Vector Control (TVC) allows attitude control through three main methods:

- Nozzle deflection
- Fluid injection
- Flow Deflection

Nozzle deflection is a technique that allows the flow produced by the reaction between oxidant and fuel to be directed in the desired direction, special gimbals are used. The fluid injection is used inside the nozzle to actively modulate the thrust levels according to the situation in which you are while the flow deflection allows to modify the geometry of the flow thanks to special tools such as spoilers. These tools are based on thrust reduction and not all of them allow to manage the complete attitude of the vehicle. As mentioned earlier, optimization of the combustion process is accomplished by dividing the launcher structure into stages. This allows at the moment of the launch to exploit the amount of thrust produced by a very large tank and, approaching towards the orbit, to lose this kind of weight to fit inside it and circularize. The disadvantage of staging is to have an engine for each stage and all the technology related to it. Considering a system on which no forces act, in an environment completely rarefied enough to resemble space ($p_a = 0$), conservation of momentum says that:

$$(m + \Delta m_f)v = m(v + \Delta v) - \Delta m_f(v_e - v) \rightarrow m\Delta v = \Delta m_f v_e \tag{1.3}$$

Where:

- Δm_f is the mass of the fuel burnt.
- v_e is the velocity at which the flow exits.

Moving to infinitesimal magnitudes, Equation 1.3 becomes:

$$m\frac{dv}{dt} = \dot{m}_f v_e + p_e A_e \tag{1.4}$$

Instead, the total thrust is composed of the following contributions:

$$T = p_e A_e + \dot{m}_f v_e = \dot{m}_f c = \dot{m}_f g_0 I_{sp} \tag{1.5}$$

where c is the effective exhaust velocity, g_0 is the gravity acceleration of the considered planert and I_{sp} is named specific impulse . The last one is one of the most important performance parameters of a rocket engine. The steps that will follow in this discussion are designed to derive the Tsiolkowski equation or fundamental equation of the rocket, considering what has been written before is made explicit the acceleration acting on the body.

$$\frac{dv}{dt} = -\dot{m}\frac{c}{m} = -\frac{c}{m}\frac{dm}{dt} \tag{1.6}$$

Integrating Equation 1.6 over time intervals:

$$\Delta v = c\log(\frac{m_0}{m}) = c\log(\frac{m_0}{m_0 - m_f}) \tag{1.7}$$

The specific impulse is then introduced:

$$\Delta v = g_0 I_{sp} \log(\frac{m_0}{m_0 - m_f}) \rightarrow m_f = m_0[1 - \exp(-\frac{\Delta v}{g_0 I_{sp}})] \tag{1.8}$$

Where m_f is the total useful propellant mass:

$$m_f = m_0 - m_e \tag{1.9}$$

And can be expressed in a parametric way by using Λ:

$$\Lambda = \frac{m_0}{m_e} > 1 \tag{1.10}$$

The study led throught parameters is useful to discern of the reference launcher dimensions and to generalize the problem with which the reader had to deal with. In this part, starting from the fundamental masses which compose a general launcher, a parametric study will be argumented using a schematic procedure. The reader may consider the launcher as consisting of certain masses essential to its architecture and proper operation:

$$m_0 = m_f + m_s + m_p \tag{1.11}$$

Equation 1.11 then considers a launcher as a gathering of the fuel m_f which will lead it to the target, a structure m_s which contains the fuel and other systems and the mission payload m_p. Dimensionless parameters useful for the study of a generic launcher are introduced, respectively. These are respectively the payload ratio (Equation 1.12), the structural efficiency (Equation 1.13) and the propellant ratio (Equation 1.14).

$$\lambda = \frac{m_p}{m_0} \tag{1.12}$$

$$\epsilon = \frac{m_s}{m_s + m_f} \tag{1.13}$$

$$\phi = \frac{m_f}{m_0} \tag{1.14}$$

This kind of study is useful to obtain results which depends on the parameters introduced above. Preliminary design analysis can be conducted by plotting the trend of the interested magnitude in function of another parameter, an example is reported in Figure 1.1.

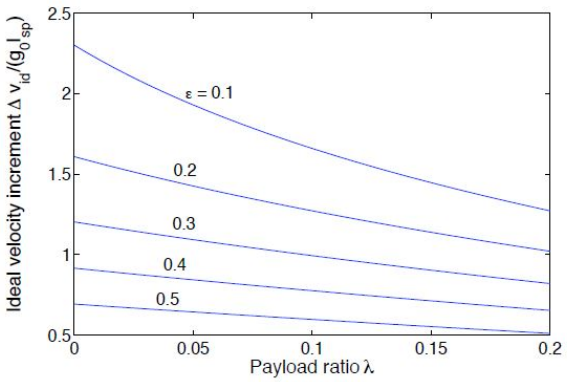

Figure 1.1. ΔV as a function of payload ratio and structural efficiency

1.2 Reference frames

[1] Mission analysis and related trajectory study involves the collection of data that may change with respect to the reference system in which it is measured. It is sufficient to know that three fundamental elements are needed to define a right-handed oriented one of theme:

- Origin of the reference system (which is the main body Centre of Mass (CoM)).
- Fundamental plane (which is the plane where x and y lies).
- Positive direction of the z axis.

Then, It's important to go to specify the most used in the study of the rocket motion to know how to better interpret the equations that concern its dynamics and kinematics.

1.2.1 Fixed planetocentric coordinates

The first reference frame $F_P := \{O, x_P, y_P, z_P\}$ is *fixed*, that means that the direction of the axis respect to the fixed stars doesen't ever change. It's defined by an origin O which coincides with the centre of the Earth, the fundamental plane is the equatorial one and the third axis, being perpendicular, is directed positively towards the Earth spin axis.
Gathering the informations mentioned until now we are able to write generic position, velocity and angular velocity of an object in this reference frame:

$$\vec{r}_P = (x_P, y_P, z_P)^T;\ \vec{v}_P = (v_{xP}, v_{yP}, v_{zP})^T;\ \vec{\omega}_P = (0, 0, \omega)^T \tag{1.15}$$

The strength of the planetocentric fixed system is that it can be approximated as almost inertial, in this way, not considering the rotation movement of the earth and the relative atmosphere, the study of the dynamics of our launcher can be conducted in a simplified way.

1.2.2 LVLH reference frame

The LVLH reference frame is, as the name suggests, a local reference frame which give us mostly information about the attitude of the mass in which it is centered. Then, it's main features are:

- Origin in spacecraft CoM.
- The fundamental plane is formed by the z-axis which is direct towards the centre of the Earth and by the x-axis which is directed towards the direction of the velocity.
- The positive direction of y-axis derived from the right-hand rule.

A clearer example is provided by the following scheme 1.2.

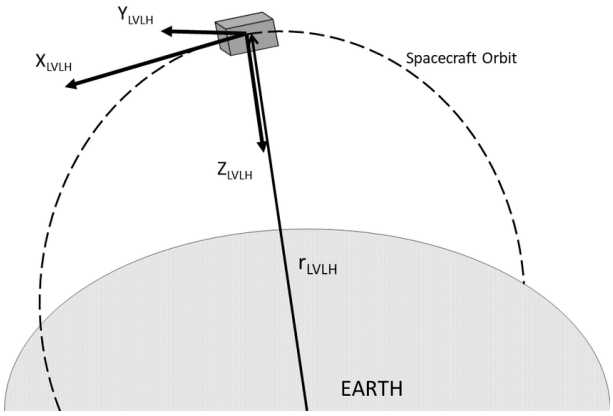

Figure 1.2. LVLH reference frame

Chapter 2

Focus on micro-launchers

[3] Part of the tradition associated with putting satellites into orbit is the use of a launcher that can carry the payload to the target orbit. Generally the launch foresees the arrangement of the satellites one next to the other in order of importance and priority according to the mission profile. In this period the access to satellites, especially those in the small category (below 500 kg) is becoming more and more extensive not only by companies but also in university projects and start-ups. The micro-launcher offers a practical and satisfactory solution to this type of request. There are generally two types of micro-launchers, the first, more traditional ones are launched from fixed platforms on the ground. The alternative involves launching from an aircraft. This type of solution is as practical as functional to have in orbit a satellite or constellations as distributed and organized as possible. Between 2011 and 2015, demand for small satellite development from universities and minority groups made up about 60% of launches with this payload. The main purpose is the development of solutions to observe the earth, technology demonstration of new devices and telecommunications. With fewer limitations than heavier launchers, micro-launchers are an opportunity to make space within everyone's reach. Over the years, private companies have been sponsored by space agencies to develop this type of technology. A case in point was in 2015 when NASA awarded RocketLab for their innovative solutions introduced by Electron within this market. Given the evolution of this technology, there could be three possible scenarios:

- Demonstration of a sustainable economic model: Despite the funding received, the lack of a business model could cause this ecosystem to collapse. Potential obstacles can be identified by technological burdens and different demands from the satellite market.

- Status Quo: The market may not change further by continuing to be fueled by customers such as small satellite operators for Low Earth Orbit (LEO) missions.

- Economic Boom: Thanks to technological innovation, the right economic support and a satisfactory demand from the operators of small satellites this type of market could grow more and more bringing both government and private interest in the development of this solution.

One of the goals with which we seek to improve the state of the art in micro-launchers is the demonstration of increasingly autonomous guidance, navigation and control related also to processes involving both attitude and position dynamics. For now it would seem that the third scenario is taking over leading space agencies to support feasibility studies about the scope of micro launchers creating new business opportunities. What ESA wants to do is to strengthen the European industry by promoting the development of these programs in order to create a technologically advanced and competitive environment. An application example of how this type of context is looking for concrete solutions is given by Future Launchers Preparatory Programme (FLPP). This program enabled five companies to sponsor an economically feasible and commercially viable micro-launcher, thus not relying on public funding. The goal, as mentioned earlier, is to place small category satellites in orbit in an efficient and economically viable manner from the ground or through an airdrop. The results were presented at a workshop in Paris where 150 participants actively exchanged ideas. From this meeting key points emerged on which future microlancer technology will be developed including low-cost avionics, composite engines and turbopumps. Presented projects are as following:

- Miura presented by PDL space.
- The Azμl program to allow an efficient launch system from the azore islands thanks to the Orbex Prime launcher developed in collaboration with Deimos.
- Avio presented the remodelling of VEGA and VEGA-C.
- MT Aerospace provided plans and ideas regarding this area.
- ArianeGroup also presented within the Quick @ccess To Space (Q@TS) program an idea regarding a prototype micro-launcher.

Starting in 2019 also, ESA has begun to fund this type of projects trying to make then space transportation and technologies applicable in orbit increasingly affordable for everyone bringing benefits in a variety of fields (from agriculture to climate change monitoring).
Since these are new technologies that are constantly evolving, it is incumbent upon us to study the reliability of these complex systems and to do so from both a component and mission analysis perspective. As far as the analysis of reliability that must be made about these systems, the information present in the bibliography are not many because commonly the data that concern this type of study remain in confidential form inside the companies. The first analysis related to reliability is

described in theoretical terms in the Chapter A.2, while the mission analysis and the state of the art of the softwares that allow to realize it will be discussed below. The problem proposed by ESA (Chapter 3) engineers that led to the realization of this thesis requires the study of branching trajectories in order to evaluate the ascent of a micro-launcher. As normal, it was then continued by searching the bibliography and sitography for software that might be able to evaluate branching trajectories in order to optimize them. Among these, not many have the possibility to alter the characteristics of the dynamics of the body of which we want to know the propagation by producing a multiple-shooting optimization. Produced by NASA and very similar to this type of task is Program to Optimize Simulated Trajectories II (POST-II)[19]. This is a discretely parameterized target point dynamics software that allows for optimization. It is capable of studying the dynamics of bodies orbiting planets both nominal and non nominal conditions due to the ability to keep certain subsystems active or not within the simulation. The all happens in precise way according to the demands of the customer that will be able to add costraints either equality or inequality. POST-II would have been perfect for the evaluation of the problem for which this thesis arises were it not for the fact that access to the code is via a request to be made to NASA. In order to avoid bureaucracy and, at the same time, to give more customization to the work performed it was decided to start from scratch with the design of a similar program that will be explained in Chapter 8.4. Another alternative is provided by ASTOS by ASTOS Solution, professional software used for mission analysis that allows a complete study of the systems and dynamics of the object concerned. This software is as complete as it is complex and its use is subject to the possession of an appropriate license that allows its use. Although the data provided by this tool is very reliable, branching trajectories are not allowed. The only alternative considerable is that one to simulate the optimization of the same object altering of the nominal characteristics and estimating the hypothetical ramifications that it could assume. For these reasons, the results produced by Intelligent Rocket Ascent Optimizer (IRATO), the solution developed by the author, will be validated in the Chapter 9 by this software. These software, together with their more commercially known counterparts, make up the state-of-the-art solutions currently available for conducting this type of study. The purpose of this thesis is to increase the range of available solutions by generating a tool that ad hoc solves a problem of optimization of branched trajectory identifying figures of merit that can allow to continue to the target orbit despite the presence of an operational failure or, if this is limiting the continuation of the trip, to create a corridor of abortion for the protection of population centers. For this reason in the next chapter a discussion around the problem faced starting from its statement will be introduced.

Part II

Second part

Chapter 3

Problem Statement

In this section we get into the heart of the thesis by evaluating the proposed problem and the various attempts that were tried to arrive at the final results. The execution was not immediate and the path to the solution was as articulate as it was instructive from a didactic point of view. Now, let's consider together the problem presented by ESA:

> *"We need a study that considers the branching trajectories of a micro-launcher/Reusable Launch Vehicle (RLV) during its ascent phase. In particular, the problem needs to consider a study of how to handle abort trajectories to avoid catastrophic situations such as impacting population centers."*

Subsequent thoughts formulated by the author, and related results, are given below. It was known that the dismay from covering such a broad and general topic stemmed from multiple perplexities, including:

- What branching trajectories are ?
- What is the difference between a micro-launcher and a RLV ? How can I manage the relationship between them ?
- What kink of failure can cause an abort trajectory ? How the system is comprimised by that ?

It was realized that this doubts could only be managed and dealt with by dividing the main problem into subproblems and understanding the interrelationship between them. What was clear from the beginning was that there was a need a tool capable of simulating the trajectory of a rocket (let's leave RLV aside for now) in both nominal and non nominal conditions. In this regard it was immediate to refer to a numerical calculation software for the generation of this kind of trajectory. The initial choice foresaw among the various available:

- ASTOS by *ASTOS Solutions*
- Python
- MatLAB/Simulink by *MathWorks*

The first provisional choice fell on the software by *ASTOS Solutions* as it is specialized in mission analysis and certainly more accurate with regard to the consideration of real models that would involve both the system of interest and the surrounding environment. This, through an already configured Electron model, would have allowed a reliable and precise study of the nominal situation of the flight, however not allowing too many degrees of freedom on the type of study that we wanted to carry out. It was therefore decided to use this program not as a test bench but as a confirmation of the results obtained through more manual codes but no less useful for the didactic purpose of this work. Python has been considered as a programming language given its great adaptability in any situation. Since the work would and has involved parts of artificial intelligence a pro in favor of this language would definitely be the community that involves and manages these packages. However, given the large amount of information and problems to manage provided by the mission statement the choice of this program was excluded to avoid problems related to the deadline of the work. The final results are in any case accessible through this type of language given the conceptual simplicity with which they were obtained. Last but not least, *MathWorks'* solution was considered as the only suitable candidate. This choice would have resulted in a more elastic handling of the problem given the presence of both MatLAB and Simulink. While the former allows for a strong tool for numerical computation and a broad spectrum for data visualization and management including AI libraries, Simulink would have granted a solution based on simulation and management of a fully scalable model since it is built by the user.
The fact of managing a new work, for which we didn't have enough knowledge (neither personal nor bibliographical) made us choose these two softwares in order to be as careful as possible and to guarantee a wide range of viable solutions.

Chapter 4

Rocket Dynamic Simulator

[12],[13] The initial reasoning in testing the trajectories of a launcher was to reconstruct a mathematical model of it. The motivation that led to this choice was the basic idea of having an approximate model that would allow the evaluation of the trajectory and the data that composed it instant by instant. Going to build a precise model it would have been also easy to vary the nominal characteristics of the launcher to appreciate not only the nominal results but also those resulting from a possible failure thus developing the argument of branched trajectories.
In addition to the bibliography, the creation of this thesis was also made possible by media materials found on the internet (video tutorials, online courses). What follows is an explanation of how the Electron model was built in Simulink thanks to the explanation of the webinair: *Simulink: Rocket Launch Simulation* by the MatLAB ambassadors Marco Lombardo and Andrea Togni. Although the tutorial leads to an inaccurate result the code has been corrected and you can evaluate reliable results when compared to the imposed approximations.
Electron model is structured as follows: it is a multi-stage rocket in that its functional operation is to release an operational payload into orbit. The goal of the multi-stage is to maximize the amount of payload relative to the ΔV provided. During launch, the stages, once the fuel is exhausted, are undocked, this is an advantage because we are going to eliminate the structural masses that contain the fuel now exhausted. To better realize how these quantities are related, the Tsiolkowski's equation is introduced:

$$\Delta V = v_e \; ln(\frac{m_0}{m_f}) = g_0 \; I_{sp} \; ln(\frac{m_0}{m_f}) \tag{4.1}$$

Where:

- ΔV is the amount of speed required to reach a given orbit
- v_e is the effective exhaust velocity

- m_0 is the initial mass
- m_f is the final mass
- g_0 is the gravitational acceleration on the planet surface
- I_{sp} is the specific impulse

To write the equations that instead govern the dynamics of my rocket we refer to a body fixed system with the x-axis agreeing with the longitudinal axis of the rocket, the y-axis exiting the plane and the z-axis completing the triad. An x_{ned} is also introduced that points by definition to the north. A clearer raffiguration is provided by the scheme 4.1. In this way we are able to write the following equations:

$$T = -D - Mg\cos\theta \tag{4.2}$$

$$\dot{m}v_e + (p_e - p_a)A_e = -D - M_{inst}g\cos\theta \tag{4.3}$$

$$\frac{dm}{dv}v_e + (p_e - p_a)A_e = -\frac{1}{2}\rho V^2 C_D A - M_{inst}g\cos\theta \tag{4.4}$$

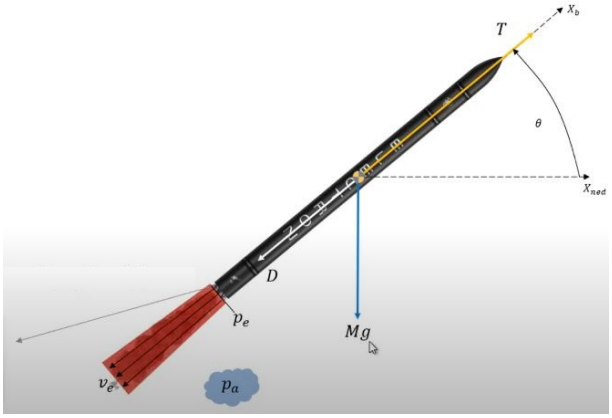

Figure 4.1. Body fixed reference frame

A first approximation involves the fact of considering within the resistance coefficient only the area related to the fairing and not the effects of resistance that are formed on the lateral part of the system. However, the body fixed reference system does not take into account aspects related to the rotational dynamics of the rocket. Therefore, Euler equations referred to the center of gravity of the rocket are introduced:

$$I_1\dot{\omega}_1 + (I_3 - I_2)\omega_2\omega_3 = M_1 \; with \; \omega_1 = \frac{d\theta}{dt} \tag{4.5}$$

$$I_2\dot{\omega}_2 + (I_1 - I_3)\omega_3\omega_1 = M_2 \; with \; \omega_2 = \frac{d\phi}{dt} \tag{4.6}$$

$$I_3\dot{\omega}_3 + (I_2 - I_1)\omega_1\omega_2 = M_3 \; with \; \omega_3 = \frac{d\psi}{dt} \tag{4.7}$$

In the drafting of the equations must be taken into account that some quantities such as the mass year non-zero derivative in time. In fact we know from the basic working mechanism of a rocket that this quantity varies during time and is also directed towards a certain type of orientation, for these reasons it must be assumed as a vector and not as a scalar. Given the above equations, a keen eye might realize the complexity of the problem that needs to be addressed, there is indeed a system of six concatenated differential equations. As far as the state of the art of today's calculators can allow us to deal with this problem, we have used the Aerospace Blockset of Simulink that solves this problem automatically, calculating the trajectory (both in terms of pose and attitude) given certain parameters as input.
The data set used for the construction of the Electron model concerning its nominal characteristics is then reviewed:

Parameter	Value
C_D	0.25
I	*eye(3)*
m_{s1}	9250 + 250 Kg
m_{s2}	2050 + 250 Kg
m_p	150 Kg
$\dot{m}_{se}$	8 Kg/s
v_e	3050 m/s
p_e	100000 Pa
n_{es1}	9
n_{es2}	1

Table 4.1. Electron data

Compared to the approximation made on the aerodynamic drag (0.25 is a typical value for a cone of that size) we notice how the inertia matrix also does not faithfully represent the physical-geometric reality of our launcher. To obtain an even more reliable simulation it would be necessary to perform a more in-depth structural study. The following subchapter will consider step by step the realization of the model used for this first phase of study.

4.1 Electron Simulink model

Since the dynamics of a variable-mass 6 Degree of Freedom (DoF) launcher can be easily computed by the package made available by Simulink, the part on which the study of the code focuses is the modeling of everything that is excluded outside of the dynamics equations. It was in fact created a block called "Electron rocket" in which this type of integration takes place. Taking a look at the figure 4.2 it can be seen that the outer ring of the code consists of all the external forces (resistance and control) acting on our system.

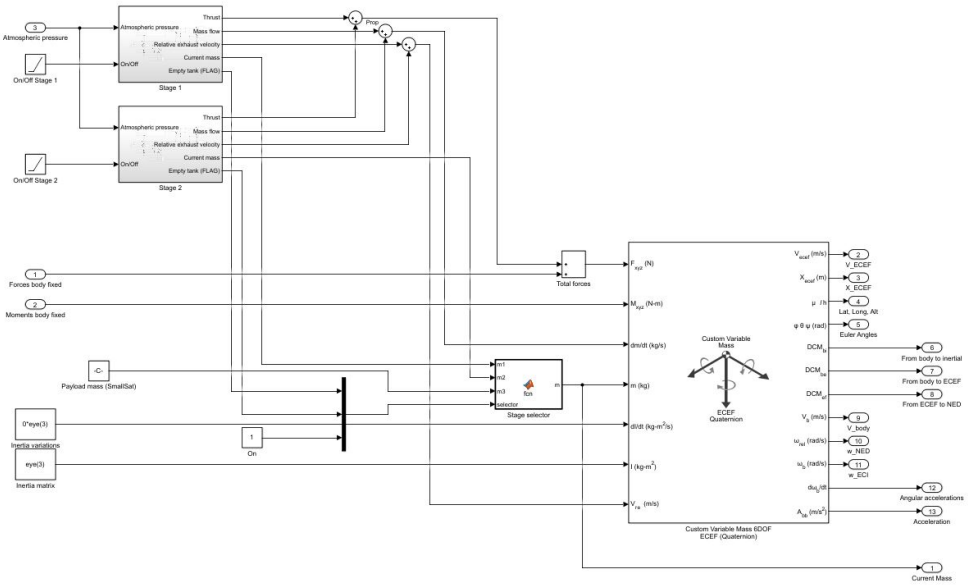

Figure 4.2. *Electron rocket* block

Based on the block described above we note that the most relevant part of the code was to build inputs that would allow us to simulate the launcher as realistically as possible. Starting from the forces we know that the most notable is the one given by the thrust. For this purpose it was built a set for each stage that would emulate the behavior of thrust generation through the nozzle, always updating the value of ambient pressure and evaluating reliable values. The behavior of the motors is also indicative of the value that the mass and its variation assume in time. Last but not least, the effective exhaust velocity also depends on the construction of this block. In order for the stages to assume the correct sequence, a system of logic gates was used that would take into account the mass of fuel inside each of them. Once finished in fact, the system updates itself by unhooking the exhausted stage and then responding coherently with respect to the physics of the problem. The first stage, despite being composed of 9 engines, has been treated for the total

thrust it can provide since a single block was made for all of them. Obviously in case you want to obtain a greater accuracy, both in the results and in the control, what you could do is to divide and manage individually each of them going so to improve the usefulness of the code and its results. Contributions of drag, weight force, and an attitude control force were also added to the thrust. The first one, always keeping in mind the assumption made on the fairing, was calculated using the aerospace blockset with the simplification that center of gravity and center of pressure coincided. Using the International Standard Atmosphere (ISA) atmosphere model the blockset evaluates perturbative forces and moments that must be taken into account in the resulting total forces and moments. As far as the weight force is concerned, the evaluation of this one was carried out taking into account the dependence of this value with respect to the altitude at which one is located. It is inevitable to think how in the calculation of these forces it was necessary a constant work of transformation using the appropriate rotation matrices that have allowed the transition from fixed and centered reference systems (Earth Centered Earth Fixed (ECEF)) to local systems (body). A final control force was added. Although the systems that regulate attitude are far more complex it was just right to impose a pitch angle that would bring my launcher from 90° to horizontal once I got to altitude. Real phenomena such as gravity turn are not handled by the simulation. This type of control was handled by using a Proportional Integrative Derivative (PID) in its proportional-derivative form and tuning the parameters competing with its coefficients so that the setup would be pitch consistent. The management and appreciation of the results has been made possible thanks to the use of the Mapping Toolbox of MatLAB, the data are then reported about the 3-D and 2-D nature of the problem, the latter with focus on height, speed and amount of mass consumed as a function of time.

The final results of the simulation can be appreciated in the figures 4.3 and 4.4. Electron's reference mission is to release the payload into a polar orbit at an altitude of 500 km. As you can see this happens in about 1600 seconds. We will see how in reality and for the following results are much smaller. Since there is no propagator of the two bodies problem, once the nominal altitude is reached, the launcher starts to lose altitude, which would not happen if the radial velocity conditions were respected. A further validation is given by the results that will be discussed in Section 8.4.

The purpose with which the simulation was built is to have a program that is able to act on the characteristics that influence the thrust to evaluate how the trajectory is modified following a failure. Having expressed a dependence in an explicit form on what happens inside the nozzle, it would therefore not have been difficult to go from a nominal trajectory to a degraded one by including information about branched trajectories. However, although this method was a key part of starting to approach the work, this turned out to be after some reasoning the not best path on which to continue. The weakness of this method lies in the fact that there is no

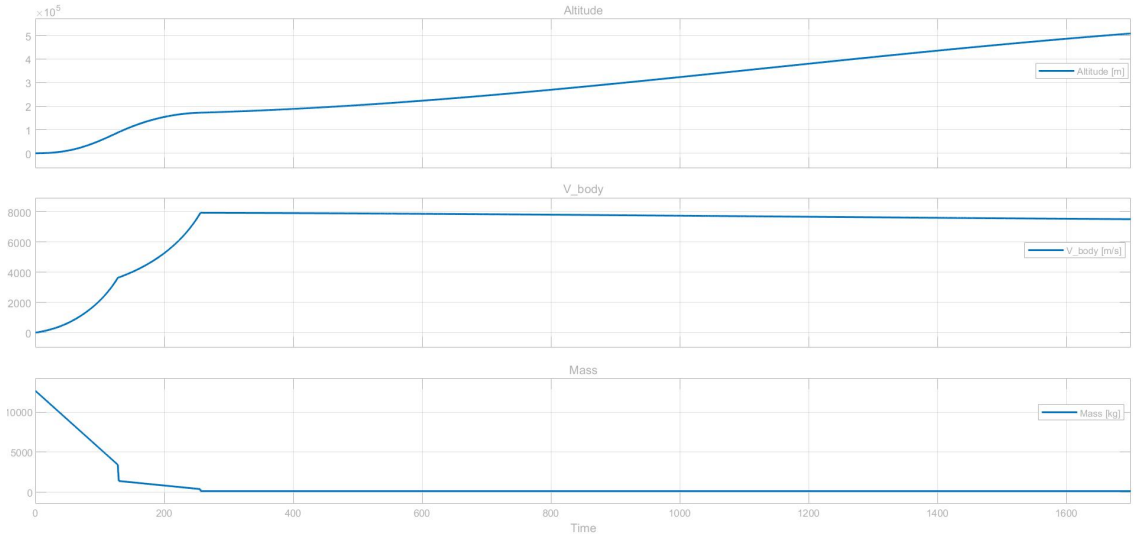

Figure 4.3. Height, Velocity and mass consumption as a function of the time

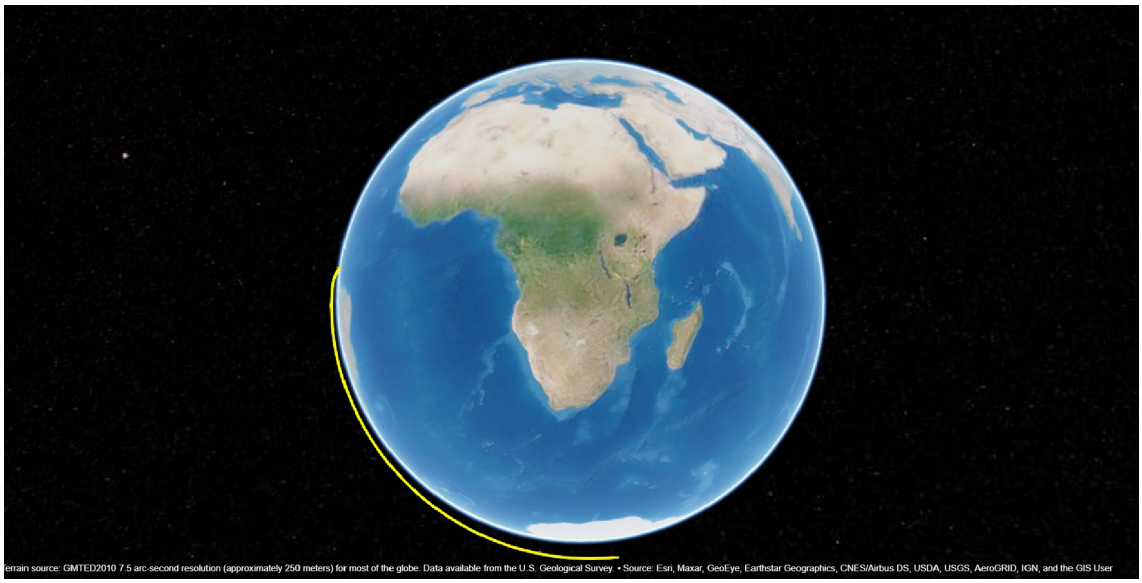

Figure 4.4. 3-D trajectory visualization

control law that goes by controlling some parameter of my launcher in such a way that a certain magnitude (such as fuel economy or mission time) is minimized. The launcher does nothing but produce thrust as long as there is fuel inside the relevant tank. So this reasoning leads us to appreciate a "free" behavior of my system and

not calibrated towards what could be a real need. The trajectory produced in output is not an optimized trajectory and is also affected by the simplifications that have been made to the model so that it can be treated with relative simplicity. The understanding of this concept has been a fundamental step in the continuation of the work because it is understood that a simple propagation is not enough, the art of optimization of the trajectory is something that, compared to the ability of a system to perform a certain task, goes to scratch the superfluous going to generate results that in addition to having physical and geometric consistency also have a practical implication related to the saving of a certain amount. The awareness about the fact that this kind of solution was not the best has taken even more way since I realized that the goodness of the results obtained also dependeds on how well the model was built. So, although this initial step was an end in itself, it definitely defined better how optimizing was a fundamental prerogative to give more reality and reliability to the data presented. This change of program would certainly have meant a move from Simulink to a numerical calculation program. Fortunately, as mentioned earlier, the other side of the coin, MatLAB, fit like a glove for such problems, confirming that the versatility of MathWorks' product was the right choice. The next chapter is an introduction to the methods used to perform the optimization not found through this method, as well as a logical continuation of the work performed.

Chapter 5

Gradient-based methods

[4] Consequently, the logical workflow shifted towards finding different algorithms that would allow the optimization of a trajectory. But what does it mean to optimize ? If we were to think of this action as a recipe, we would need to identify the elements that make it up. Basically, there are two:

- The variable you want to maximize/minimize
- The variables on which you want to act with a control

Once you have identified the pivotal figures to generally obtain as output a trajectory that respects the imposed constraints, you must keep in mind that the methods to make this happen are multiple and all have pros and cons. In this section we will focus on the so-called gradient-based methods, i.e., methods that exploit gradient descent.
To understand how this algorithm works we can introduce a simple example: let's imagine we are on a mountain and we want to reach the plain (which will be conceptually understood as the minimum of the function). In all this what we possess of the environment is not a global information but, due to the presence of many trees near us, we are able to evaluate a local information about the path in which we are venturing. Surely the most intuitive solution would be to look around and take a step towards the direction in which the slope seems to be the steepest because following that we will arrive quickly to our destination. Iterating in this way we will arrive, step by step, in an area where the lowest steepness is the one linked to a flat territory, in this way we will have reached our goal.

This is conceptually the working principle of a gradient-based algorithm. The fundamental steps for the realization of the same are therefore:

- Search direction
- Step size
- Convergence check

The first step might find its way into our example by considering the moment in which we choose the direction in which to descend. In mathematical terms the derivative is the tool that gives us global information about the growth/decrease of the example function. In the simplest cases (one-dimensional functions) this is geometrically associated with the slope of the function itself, while for functions the operator will take the form of a gradient. The direction of maximum punctual steepness will be identified thanks to this tool by defining the search direction. Once the desired direction has been identified, we proceed wondering for how long to continue towards that direction. Our function, or mountain, could change its topological characteristics in a non-uniform manner and it is therefore not certain that once the direction of maximum slope has been identified, this will remain constant throughout our journey. Therefore it will be necessary to individuate a step size suitable to avoid overshooting problems. The name that is used by the solver to indicate this feature of my algorithm is precisely the step size. After defining the first two elements, we have set a code that is able to identify a direction of maximum speed towards which to descend and also a step size. These two ingredients are necessary to arrive at an optimum, but being sure that the solution I've found is the global one is the necessary condition to know that you get a global convergence. A mathematical discussion of the above is given below.
Necessary condition sufficient for this method to be applied is, given a function $F(x_1, ..., x_n)$ is that this is continuous and differentiable around a point $\mathbf{a}$. We know that the gradient of the function calculated in this form:

$$-\nabla F(\mathbf{a}) \tag{5.1}$$

Indicates the direction in which my function descends most steeply from the moment in which, by definition, the gradient of a function indicates the direction and value of the steepest increase. In this way, starting from an attempt value $\mathbf{x}$, we will proceed iteratively updating the value according to the procedure:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n) \tag{5.2}$$

The term γ is a real, positive and number chosen in such a way that if it is enough small, the quantity:

$$\mathbf{a}_{n+1} = \mathbf{a}_n - \gamma \nabla F(\mathbf{a}_n) \tag{5.3}$$

will make the function assume a smaller and smaller value than the previous one. The speed of convergence and the reliability of a result typically relate to the morphology of the function created, whether for example the function is convex or not. The pseudo code for this reasoning might look as follows:

Algorithm 1: Gradient descent pseudo-code

```
n=0; initialization;
while F(x(n)) != 0 do
    instructions;
    p(n) = -∇ F(x(n));
    α(n) = ...;
    x(n+1) = x(n) + α(n)p(n);
    n=n+1;
end
```

The declination of this technique changes depending on the type of function being handled. In our case, dealing with the nonlinear dynamics of a rocket, the approach of a nonlinear problem is presented.
Given my initial system, I can write the associated function $G(\mathbf{x})$. In this way, the objective function will become:

$$F(\mathbf{x}) = \frac{1}{2}G(\mathbf{x})^T G(\mathbf{x}) \tag{5.4}$$

Which we will be able to minimize. The loop can then begin to iterate starting from an attempt value $\mathbf{x}_n$ and thus updating the next solution as:

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma \nabla F(\mathbf{x}_n) = \mathbf{x}_n - \gamma J_G(\mathbf{x}_n)^T G(\mathbf{x}_n) \tag{5.5}$$

Where $J_G(\mathbf{x}_0)$ is the Jacobian matrix.
The gradient descent method is therefore a mathematical tool as elementary as it is necessary in understanding how to reach the local or global minimum of a function. As much as this can be used directly within the nature of the problem to solve it, its applications sometimes involve collateral aspects. In this thesis we will appreciate a few examples. Although the optimization tools of commercial software provide some sets of functions that exploit this type of technique (in its declination both constrained and unconstrained) we will delve into techniques also called non-heuristic and in the comparison between the two solutions. The mathematical nature of this concept is something that will also be taken up in the later application of the neural network. The explanation, simple, but at the base of which lies the operation of this technique uses precisely mathematical laws such as gradient descent to work and produce results as reliable as possible. Anticipating the treatment of the problem, it is not difficult for us to understand how this type of methodology did not prevail over other options. The rocket trajectory was optimized by dividing itself into N segments. To compose each segment a vector of control variables (throttle, thrust angle) would be varied in order to minimize the fuel fraction used, thus saving fuel. A gradient based solution would be fine in this perspective because the presented problem has both control and state variables. However, in the practical view of contemplating even abrupt variations related to

the generation of an instantaneous failure, the algorithm would not be able to switch function and identify and update the optimization conditions. A valid alternative remains for this type of strategy the multiple shooting optimization. Since the mathematics of the problem is complex to deal with, in the next chapter what is presented is an alternative, as simple as effective through which we were able to obtain an optimization by varying in real time the characteristics of a launcher and thus obviating the multiple shooting.

Chapter 6

Particle Swarm optimization

[2],[5],[6],[7],[15] This chapter introduces a relatively recent algorithm that is as simple as it is effective in optimizing a wide range of funions. We speak in fact of particle swarm optimization. This algorithm is mentioned for the first time in by Kennedy, J. and Eberhart, R. in 1995. The algorithm was born from the observation of some social phenomena present both in nature (such as the coordinated movement of schools of fish or flocks) and observations related to their collective behavior. Moreover, a further look is directed to the category of genetic algorithms to which this type of strategy refers. The strength of this technique lies precisely in its simplicity: in fact the mathematical model described uses fundamental mathematical operators limiting itself to the derivative as the most complex. Conceptually it is very easy to understand and to implement in a few lines of code if the reader is interested in declining it in his work. A further pro is that, computationally speaking, the code is not heavy and can provide good results even in the case of very complex functions with many variables. Through the use of computer simulations that recreated the behavior of the flocks, it was possible to understand some of the logic of the mass dynamics inherent in the flocks. Greater focus has been highlighted in the mechanisms of movement of large numbers of particles in forming particular choreographies. The interest was just in understanding how the coordination was managed in moving some parts in a coordinated way of the groups but especially because that type of disposition represented an excellent for the logic of the flock. historical times we are living in, is that sharing information between elements is the only way the species has to find a solution. In fact, quoting sociobiologist E. O. Wilson, *"In theory at least, individual members of the school can profit from the discoveries and previous experience of all other members of the school during the search for food. This advantage can become decisive, outweighing the disadvantages of competition for food items, whenever the resource is unpedictably distributed in patches"*. Sharing information is therefore, in evolutionary terms, the only way that a group has to survive in nature. This is the cornerstone on which the particle swarm optimization is based. The initial movement of the single agent was realized

taking into account two aspects: the velocity of the nearest neighbor and craziness. In fact, by initializing a species in a scattered form within a region of space, each agent was associated with a portion of the speed of its neighbor, in order to recreate the same natural effect of harmony. However, the effect produced was that of a fairly monotonous motion since, as they all followed the direction of the nearest neighbor, they lacked inventiveness about changing direction. Therefore, the second element, craziness, was added so that a one-time agent would respond randomly instead of following its neighbor. In this case it was possible to build a more real mathematical model. The turning point during Happner's simulation development was when craziness was replaced by a goal imposed on the flock. Happner knew that one of the many times when the flock "rests" and stops dancing is when food is found. It may be that no member has ever been in that region before, but thanks to an efficient exchange of information, only one agent needs to be aware of this information to deviate the collective logic of the group in that specific region of space. The next step of his study was then the implementation of a cornfield vector that allegorically could represent in our case the topological optimum of any region of space. The mutual distance between the single agent position and the cornifield, located in (100,100), has been defined as it follows:

$$Eval = \sqrt{(present_x - 100)^2} + \sqrt{(present_y - 100)^2} \tag{6.1}$$

Where $present_x$ and $present_y$ are the agent coordinates. With the inclusion of a global optimum to be found, the individual agent will weight its movement (pose and velocity) as a function of distance relative to the local optimum it travels. For each agent there will be this component named p_{best}. As well as value, coordinates of the best will be preserved in such a way that each agent will be defined by a class[1] composed by:

- Its current position $present_x$ and $present_y$
- The best it has never met p_{best}
- Best coordinates p_{bestx} and p_{besty}
- Its current velocity v_x and v_y
- The next velocity increment $p_{increment}$

In this way, the four configurations (above, below, right and left) that the agent will assume with respect to the global best are the following according to the reciprocal position between the two:

[1] In this specific example values will be expressed as vectors

```
if presentx[] > pbest[gbest] then vx[]=vx[]-rand()*g_increment
if presentx[] < pbest[gbest] then vx[]=vx[]+rand()*g_increment
if presenty[] > pbest[gbest] then vy[]=vy[]-rand()*g_increment
if presenty[] < pbest[gbest] then vy[]=vy[]+rand()*g_increment
```

This algorithm is repeated by an arbitrary amount of time and every step is called "epoch". Having defined the initial set of parameters, the development of particle swarm optimization has been refined by removing both the random variable called craziness and the information inherent in the nearest neighbour.

"The increments on p_{best} and g_{best} are necessary. Conceptually p_{best} resembles autobiographical memory, as each individual remembers its own experience (though only one fact about it), and the velocity adjustment associarted with p_{best} has been called "simple nostalgia" in that the individual tends to return to the place thiat most satisfied it in the past. On the other hand, g_{best} is conceptually similar to publicized knowledge, or a group norm or standard, which individuals seek to attain. In the simulations, a high value of princrement relative to g-increment results in excessive wandering of isolated individuals through the problem space, while the reverse (relatively high-increment) results in the flock rushing prematurely toward local minima. Approximately equal values of the two increments Seem to result in the most effective search of the problem domain.[2]"

We will now see how this type of algorithm has been declined in practical form within the MatLAB environment to go and use it in what will be the problem of optimizing the ascent trajectory of a launcher.

6.1 PSO Mathematical Model

Now that we understand the conceptual workings of the algorithm, we will get into the nitty-gritty of the pseudo-code to understand the relationships that affect the motion of the particles in search of an optimum. Each particle defined within our N-variable space will be a candidate solution to a problem identified with a cost function where N is the number of variable on which we want to get an optimum. Each particle will be identified by a position $\vec{\mathbf{x}}_i(t)$ where $\mathbf{x}$ belongs to the set of real numbers and t is a discrete-time variable. The particle will also have a velocity denoted by $\vec{\mathbf{v}}_i(t)$ which tell us the magnitude and the direction of how the particle is moving. In addition the "simple nostalgia", or, better to say, the memory of the personal best reached and kept by the particle will be identified under the variable "personal best" that, in notation, becomes $\vec{\mathbf{p}}_i(t)$. Since the information of the single individual is useless if it is not communicated to others, it will be necessary to introduce a collective memory $g(t)$ in which the best exeperience of the whole group is stored, so that this will function as a reference. At each epoch, the value of position and velocity will be updated. The next scheme shows in a schematic way how these values modify the velocity of the particle after a single step.
Then, we can write:

$$\vec{\mathbf{x}}_i(t+1) = \vec{\mathbf{x}}_i(t) + \vec{\mathbf{v}}_i(t+1) \tag{6.2}$$

$$\vec{\mathbf{v}}_i(t+1) = w\vec{\mathbf{v}}_i(t) + c_1(\vec{p}_i(t) - \mathbf{x}_i(t)) + c_2(g(t) - \vec{\mathbf{x}}_i(t)) \tag{6.3}$$

[2] [5]

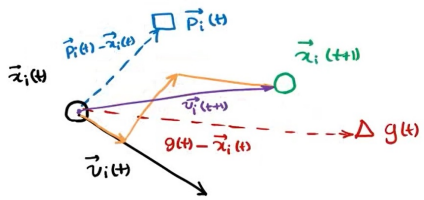

Figure 6.1. Graphical representation of a PSO step executed by an agent

The previous equations fully represent the entire mathematic law which describes agent's motion. Instead, the speed value at the next step will be updated as follows:

$$\vec{\mathbf{v}}_{ij}(t+1) = w\vec{\mathbf{v}}_{ij}(t) + r_1 c_1(\vec{\mathbf{p}}_{ij}(t) - \vec{\mathbf{x}}_{ij}(t)) + r_2 c_2(\vec{\mathbf{g}}_j(t) - \vec{\mathbf{x}}_{ij}(t)) \tag{6.4}$$

The first member to the right of the equation is called *inertia term*. The second one is called *cognitive component* and the latter is called *social component*. Where:

- j is a scalar value which denotes the j-th component of $\vec{\mathbf{v}}$
- c_1 and c_2 are the *acceleration coefficients*
- r_1 and r_2 are two evenly distributed numbers between 0 and 1
- $\vec{\mathbf{g}}_j(t)$ is the j-th component of the global best vector

Meanwhile the equation to update the position is similarly:

$$\vec{\mathbf{x}_{ij}}(t+1) = \vec{\mathbf{x}_{ij}}(t) + \vec{\mathbf{v}}_{ij}(t+1) \tag{6.5}$$

Going to translate this kind of reasoning in the form of code the main parts that we should go to identify are basically five:

- **Problem definition:** In this section to be defined is a cost function, i.e., a non/linear relationship that binds the state and control variables of our problem. The goal of the algorithm will be to minimize/maximize the function, such that a desired set of variables is obtained. Furthermore it is advisable to start creating a vector which takes into account the number of variables on which the problem depends by defining their domain.
- **Parameter of PSO:** The parameters of the algorithm involve the number of iterations to be performed by the swarm, the number of agents composing it, and the value of the inertia coefficients, personal and social. The choice of these parameters is of course arbitrary and depends on the type of problem you need to solve. Intuitively the higher the number of iterations, the greater the accuracy of the optimum found, however this type of procedure does not ensure

that the minimum found is global. In this regard, the more agents available, the greater the region of space inspected, raising the probability of finding a global optimum. The best choice is obtained by making a compromise between these two parameters in order to obtain reliable results without increasing the computational cost of my code.
Talking about the coefficient c_1, c_2 and w, they were calibrated following Clerck and Kennedy constriction in this way:

$$k = 1 \tag{6.6}$$

$$\phi_1 = 2.05 \tag{6.7}$$

$$\phi_2 = 2.05 \tag{6.8}$$

$$\phi = \phi_1 + \phi_2 \tag{6.9}$$

$$\chi = \frac{2k}{\| \, 2 - \phi - \sqrt{\phi^2 - 4\phi} \, \|} \tag{6.10}$$

In this way ...

$$w = \chi \tag{6.11}$$

$$c_1 = \chi\phi_1 \tag{6.12}$$

$$c_2 = \chi\phi_2 \tag{6.13}$$

- **Initialization:** In the initialization part what needs to be done is basically to define an "agent" class consisting of: position, speed, cost, best position and best cost. In this way all the elements we have analyzed in the theoretical part will be evaluated. An iterative loop will repeat this type of action as many times as there are particles that we have decided to initialize.

- **Main loop PSO:** In this step each particle performs what is the substantive reasoning of PSO in which what is evaluated is the merit achieved by the particle in its current position. Following an evaluation between the personal optimum and the one identified by the rest of the swarm, the particles will move to the next step according to the laws described so far.

- **Results:** Once the cycle is finished for the number of epochs decreed by us, the combination of variables that lowers the cost of the merit function as much as possible will be exposed as a solution. Of course, this does not ensure the fact of having obtained a global minimum of our function, only a careful trade off between the number of particles used and the number of epochs can ensure the goodness of the results we have obtained. Usually results are plot since the merit function is not indicative about the nature of the primary problem.

As general final considerations, one could say that increasing the number of particles often has a very positive effect in avoiding premature convergence to a local solution. Of course, if one is very confident that one has found the desired optimal solution (or at least a satisfactory solution), increasing the number of particles is not helpful. In principle, for a new application, it is preferable to employ a large number of particles if you see poor/satisfactory convergence on at least 5 processes employing a low number of particles. However, evaluating whether a solution is satisfactory or not can be complicated, since the optimal value of the objective function is not known a priori, unless all necessary optimal conditions are employed (and this is the approach of the indirect heuristic method).

As much as the proposed coefficients are derived from the literature, they are generally fine for a preliminary study. One could further improve the efficiency of the algorithm by selecting some of them after many tests on benchmark functions.

The goodness of a heuristic algorithm for nonlinear programming is generally tested on some benchmark problems, usually functions that admit multiple minima. Thus, ultimately, the goodness depends on the results. The parameters by which such an algorithm is evaluated are:

1. Convergence ability (which consists of avoiding "stagnation", i.e., the identification of a local minimum or a non-minimizing solution)

2. Efficiency, i.e., the number of function evaluations in a single iteration and in the entire process

3. Numerical accuracy, i.e., the ability to satisfy constraints that may be present in the problem formulation.

It is evident that parameters 1 and 3 are closely related to the solution produced. A further qualitative parameter is simplicity, which is certainly a strong point of PSO, which in fact is simple to implement.

6.2 Algorithm Validation

As anticipated, we will see how PSO will be used in solving the problem of optimizing a branching trajectory of a launcher ascent in Section 8.4. Since the function related to this problem turns out to be very complex (containing from 2 to 5 variables to be optimized) it was decided to validate the algorithm with simpler functions and with a known topology. The validation of the PSO is a fundamental step to give as much reliability as possible to the algorithm and the results it produces for two reasons. The first problem is that of identifying a minimum of the associated function allowing the whole population to converge to a precise region. The latter involves a problem called "stagnation". This type of issue can be explained by imagining the population pointing to an unreliable solution once it has found a local and not a global minimum. This drawback is solved by generating a necessarily large population relative to the topology of the function considered. There are two functions selected to test the goodness of the algorithm and their equations are given below.

$$f(x, y) = x^2 + y^2 \tag{6.14}$$

$$f(x, y) = xe^{(-x^2-y^2)} \tag{6.15}$$

These functions were chosen because they are relatively simple. The function 6.14 is a paraboloid with vertex in the origin of the reference system generated by the right-hand triad (x,y,z). For this reason it is easy to imagine that the function has only one point of minimum in the origin of the same reference system. Applying PSO with the same parameters used in the optimization of the rocket ascent trajectory, the results are reported. It can be seen that in the Figure 6.2 the population is positioned in the region close to the vertex of the paraboloid, thus validating the model used.

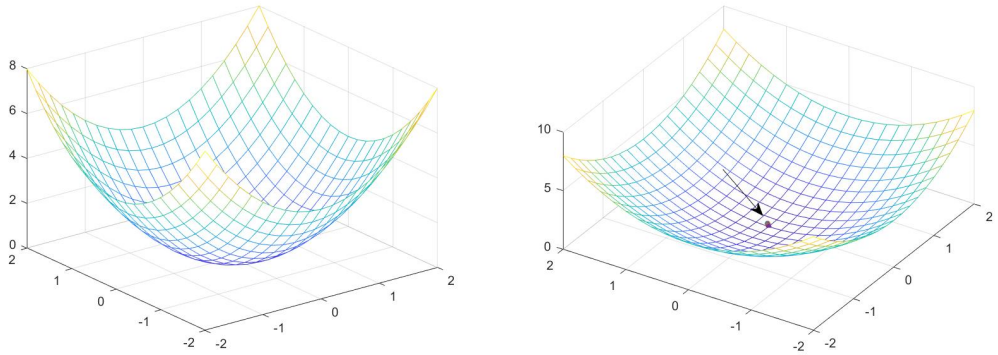

Figure 6.2. PSO applied to Funcion 6.14

The validation continued considering the second Function 6.15. Also in this case

the algorithm allowed to investigate what was the minimum global and not local of the function going to place a flock composed of 100 particles, after 100 iterations, in the region of interest. Also for this case the results are shown in the following Figure 6.2.

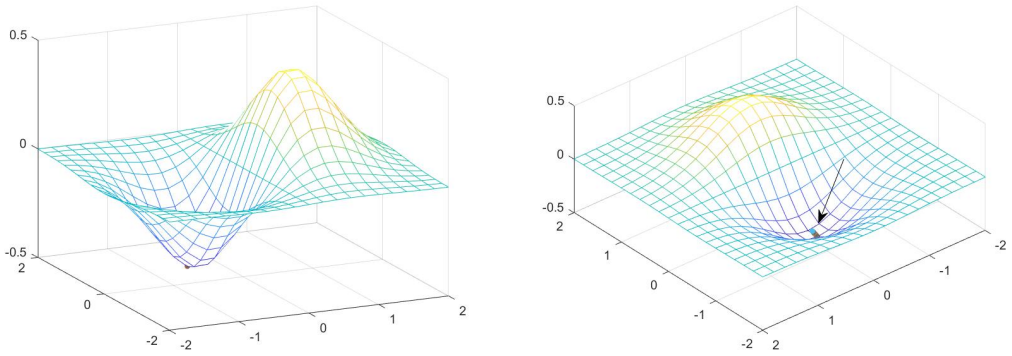

Figure 6.3. PSO applied to Funcion 6.15

At this preliminary stage, the PSO parameters that will be used to solve the main problem, addressed from Chapter 8, have been validated for use on more complex functions. This type of expedient must necessarily be employed when dealing with algorithms of this type (indirect heuristics). In the next chapter, a survey of artificial intelligence will be conducted starting with a breakdown of the various methods that make up this relatively recent technology. The goal of this thesis is to solve the problem of optimizing branching trajectories for a micro-launchers.PSO provides a very innovative tool in regards to the optimization part (the most common applications in which this algorithm is usually applied involve multidisciplinary design optimization). Artificial intelligence will come in handy in independently understanding how, following an operational failure, our trajectory should evolve.

Chapter 7

Artifical intelligence

[16],[17] Artificial intelligence is a concept that has now become very common in everyday language. It adds fascination to the instrumentation it supports, and some of its applications are now essential to some commercial algorithms. However, the use that this can have covers a wide range of areas since it should be made a subdivision of the various categories that make up these two big words. The following sections describe the theory I gathered through the development of this thesis to find a suitable technique for optimizing a trajectory. More focus is given to the neural network part which proved to be the best tool to achieve the expected results shown in the next chapter. In addition to the bibliographic learning, I am pleased to note that Andrew Ng's guides and other multimedia content have been equally helpful in discovering this field.

7.1 Machine learning

Machine learning is a rib of artificial intelligence that exploits the ability to produce results from an algorithm that is not necessarily programmed to produce them. This type of process is possible thanks to the use, storage and management of data that are provided as input in a more or less organized way. After a certain time our algorithm, provided a different data set from the training one, will be able to predict and reconstruct logical patterns that will lead to the association of a certain output. This association will be as reliable as the better the algorithm has been trained. It's easy to see how the state of the art in computers today allows for enough cumulative and computational capacity to handle and train quantities of data that, through machine learning, manage to be very commercially useful. Every mechanism at the base of anti-spam systems for email or the associations that are made in suggesting products similar to those we use in fact use large amounts of data and through the use of machine learning produce very useful results for large and small companies. This type of mechanism will only get better given the

ability of modern systems to handle and accumulate ever larger amounts of data. To implement a machine learning algorithm, typically there are always four steps to follow and they are described below:

- **The training data set:** The preparation of a data set is very important because based on the goodness of the internal organization of the database will depend on the reliability and goodness of our algorithm. This phase, for the reason described above, is therefore expensive in terms of time and attention. It typically involves collaboration between the programmer and the data collector. These last ones can in fact derive from market analysis or simulations of the phenomenon considered in various conditions. The data can be managed either as a "label", i.e. identified within a category in order to make its classification even more streamlined or, in some cases, unlabeled. A machine learning algorithm is able to manage both these resources in a more or less efficient way, reconstructing, for the latter, a logical path at the end of which it is able to categorize even the unclassified data.
- **Algorithm choice:** Based on the type of problem, e.g. labeled or unlabeled data/quantity of data, the best algorithm strategy with which to approach the problem can be identified. Based on the type of problem, e.g. labeled or unlabeled data/quantity of data, the best algorithm strategy with which to approach the problem can be identified. The most classic scheme to follow is summarized in the table 7.1

Labeled data	Unlabeled data
Regression algorithms	Clustering algorithms
Decision trees	Association algorithms
Instance-based algorithms	Neural netowrks

Table 7.1. Machine learning algorithms

  - **Regression algorithms:** Regression is a linear or non-linear relationship that links a certain input to a certain output. According to the number of elements this can be linear (an input and an output) or not. Given therefore a certain amount of data it will be possible to reconstruct with accuracy a function that is in a position to binding the information deriving from these and, in case a new input came supplied, it will come supplied a value of output that follows the best possible the constructed model. An example would be an algorithm that is able to predict the cost of a house based on certain characteristics of the house. Let's consider the simple case in which the only variable is the square footage of the

house. In this case we find ourselves in a situation in which I will have to construct a linear relationship between the cost of the house (independent variable) and price (dependent variable). Using as much data as possible already collected on this pair, we will be able to build a function that, when we insert a square footage never seen before, will estimate the most reliable price value possible. This is the case where the regression is linear because we have a single independent variable that grows linearly with the expected output. It would be enough to add another variable (such as the crime rate of the neighboring area) to realize that, as this variable increases, intuitively the price could go down. In this case one would have to deal with a multi-variable non-linear regression. Both cases are handled in an almost equal way and there are already function packages (both MatLAB and Python) able to reconstruct a regression given a set of initial data.

- **Clustering algorithms:** Clustering is a technique that rather than producing a result, catalogs an existing piece of data within a category. For this reason it is very convenient when dealing with unlabeled data. By identifying the common "characteristics" it will be possible to perform this type of subdivision using ad hoc algorithms such as K-means[1].
- **Decision trees:** decision trees are algorithms that exploit labeled data in such a way that a particular type of executive decision can be determined on it. The algorithm will identify some characteristics of the relative data, each of them will give a relative weight to a function of merit that, weighed against the others, will be able to estimate a relative decision about the data itself. An example can be given by an algorithm that decides whether to bet or not on a soccer team based on parameters such as goals conceded in previous games, injured players, etc.
- **Association algorithms:** Association algorithms rely on connections of unlabeled elements by exploiting the *"If-Then"* nexus. For example, within a supermarket it can be useful to identify the type of recipe that the consumer will want to prepare based on part of the ingredients already bought to suggest other products to complete it. If, for example, it turns out that the binomial "flour" and "sauce" (*"If"*) is associated with the preparation of "pizza" (*"Then"*), the supermarket can place another ingredient similar to the recipe next to them, such as, for example, the most expensive yeast in stock.
- **Instance-based algorithms:** This type of algorithm is able to produce

[1] At a glance certain attributes are associated to the data in terms of vectors that will position it within a class defined as a vector space.

a classification of the data considered within a category based on its characteristics and how similar these are to those of other data.

- **Neural netowrks:** Neural networks, compared to the above types, have different behavior and utility. It is fair to classify them within machine learning when they are single layer or simple neural networks. For more elaborate types it is more correct to refer to the *Deep Learning* section where the mathematical theory behind this tool will be treated more openly. Their structure allows, through the activation of neurons, to evaluate the classification of a certain data within a class or to make a certain decision based on the fate provided as input.

- **The training:** The training part is fundamental because on the basis of this process is based the real intelligence of our algorithm. Generally speaking, in fact, on the basis of the data presented, the algorithm will associate to each item a certain weight or bias in such a way as to produce a function as reliable as possible in the case where a data never seen before is used. Of course the accuracy with which weights and biases are chosen depends on the quantity and quality of data provided, for this reason it is very important to ensure an effective collection of the latter. Of course, depending on the type of algorithm used it will be possible to evaluate the goodness with which this works using certain evaluation criteria that we will see in the case of multi-layer neural networks.

- **Improve the model:** As soon as you begin to see that the algorithm converges to a sensible solution, you can reinforce it by adding data during the training phase and evaluate relative improvements. It doesn't exist in fact a solution that can be globally evaluated as the best one but it will be up to the skill and the experience of the user to modify the parts of code in order to obtain a predictive/classificative model as accurate as possible.

In addition, we can say that typically the methodology with which a machine learning algorithm is trained can be of two types: supervised and unsupervised. The first category includes the use of the data set not only for training, but also as a confirmation that the output generated by the computer is correct. This type is very convenient when you have a database that is not too rich in data, but each of them is defined by precise labels. On the other hand, the unsupervised method produces the same result with the difference that the data fed to the algorithm are not cataloged with so much information. All these declinations of machine learning were investigated in order to find an application that could be functional to the basic problem, i.e., optimization of a trajectory that could include ramifications arising from failure generation. As we will see in the following chapters, the optimization has been possible thanks to PSO of which the theoretical functioning is widely described in the homonymous chapter. At this point it was thought that the utility

of AI could be employed in the generation and prediction of launcher behaviors from certain non-nominal throttle and thrust angle parameters. We will see how to accomplish this task, the use of a multi-layer neural network was the best choice. In the next chapter, the substantial difference between machine and deep learning is explained with subsequent theoretical discussion of this type of network.

7.2 Deep learning

Deep learning is a subgroup of machine learning that involves the use of algorithms inspired by the mechanism that the human brain uses in learning and predicting more or less elaborate concepts. This imitation mechanism takes the name of neural network and basically can be divided into three categories:

- Standard Neural Network (SNN)
- Convolutional Neural Network (CNN)
- Recurrent Neural Network (RNN)

The first category is the most generic and can potentially solve most problems since there is no application limitation. CNNs are a type of network that exploits the superposition of matrices, very useful in solutions that involve the recognition of images since the three matrices can work according to RGB coding. RNNs are functional for the recognition of digital signals such as voice recognition. Starting from this simple subdivision it will be easy to understand how our choice has fallen in the first group (Standard neural network) of NNs, however the theoretical description of their operation in the following subchapters will take as an initial example a CNN to understand its operation and then extend it to a global treatment.

7.2.1 Introduction to neural networks

Let's start understanding how neural networks works starting from an example:

Figure 7.1. Different representations of the number 3

Although it is handwritten and each handwriting is different from the previous and following ones, we have no problem in stating with certainty that the number represented in the figure 7.1 is the number 3. This type of calculation is performed by our brain in a fraction of a second and this is possible because some neurons collaborate in defining the black pixels as the number to be identified while associating to the white ones an empty space not to be considered in a space that will be composed of NxN pixels. So by identifying each pixel according to a coding such that if the pixel is black its value is 1 while if the pixel is white this value is 0 we are able to mathematize the problem and make it more "digestible" even to a computer. At this point we can associate to each neuron a pixel of my grid and insert within it the value of the gray scale that we have defined above. If then our image will be composed of 28x28 pixels, we will have 784 neurons each with a number between 0 (white) and 1 (black) composing the *first layer* of our neural network. We also know that the single digits that can be identified by our eye are the numbers from 0 to 9, so the last layer will be composed of 10 neurons, each always containing a number between 0 and 1, the closer to 1 the number considered, the greater the probability that the written digit corresponds to that number. In the middle of these two extremes we could define some structures known as *hidden layers*, their number and the neurons that compose them are arbitrary although there are some rules in literature based on the type of problem considered. To sum up, the first layer identifies the areas of space that define my number, these in turn will activate the next layer to get to have a solution that identifies one, of the ten possible, as the best candidate for the written number identified in the last layer of the network. The function of hidden layers is functional to the extent that a number is a combination of several symbols that can repeat each other. Take for example the number "6" and the number "8", while the first is composed of a loop and a line, the second is given by the superposition of two loops. Also the "9" in the same way as the "6" is composed of the same elements with a different positioning in space of them. The hidden layer deals with this, considering the largest problem (the entire number), it selects the most detailed details of it in order to make a more precise subdivision and categorization. Imagining that the neurons of the previous layer are all connected to each of the next layer, it might be natural to ask what kind of information travels between these elements in order to exchange information with each other. This type of connection is achieved by associating to each neuron of the next layer a weight from each of the previous ones. The weight is a number that is then multiplied by the activation of the neuron. Referring to the activation of the j-th neuron of the second layer[2] we could write:

$$a_j^2 = a_1^1 w_{1,j} + a_{2,j}^1 w_2 + ... + a_n^1 w_{n,j} \tag{7.1}$$

[2]In notation the superscript represents the layer referred to while the subscript refers to the neuron under consideration.

We are looking at a weighted average of real numbers. To restrict this type of solution to the domain of real numbers between 0 and 1, the result of this operation will be calculated within the *sigmoid function* described in figure 7.2.

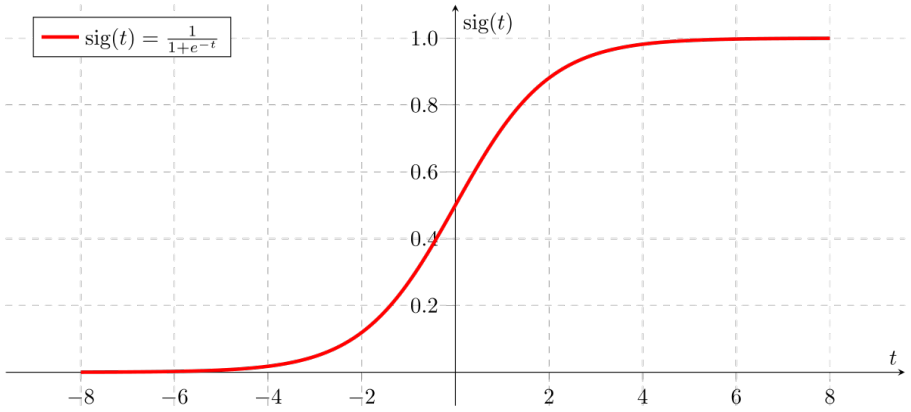

Figure 7.2. Sigmoid function

In this way, equation 7.1, can be rewritten as:

$$a_j^2 = \sigma(a_1^1 w_{1,j} + a_{2,j}^1 w_2 + ... + a_n^1 w_{n,j}) \tag{7.2}$$

In addition, to prevent our neuron from reaching its activation threshold beyond a certain value, what can be done is to add a term known as *bias* to 7.2. In this way we can write:

$$a_j^2 = \sigma(a_1^1 w_{1,j} + a_{2,j}^1 w_2 + ... + a_n^1 w_{n,j} - b_n) \tag{7.3}$$

So the weigths tell you what pixels pattern this neuron in the second layer is picking up and the bias tells you how high the weighted sum needs to be before the neuron starts getting meaningfully active.

So writing our problem in a compact notation we will have for the second layer:

$$a^{(1)} = \sigma(\mathbf{W}a^{(0)} + \mathbf{b}) \tag{7.4}$$

With:

- $\mathbf{W}$ is a $k \times n$ matrix where k and n are respectevly the number of neurons of the previous and the considered layer.
- $\mathbf{b}$ is a n-dimensional vector

Thus, we were able to circumscribe a slightly complex concept within a very simple equation. This, in addition to facilitating learning towards this solution strategy, makes us understand how in a programming logic is easy to implement and does

not require the use of external libraries. In the next subchapter we will try to understand how the descent of the grandient is also essential for this type of solution and why.

Gradient descent applied to neural networks

The weights and biases, once initialized, will have to change their value in order to produce, given an initial layer (image of the number) a correct output (value of the number itself). But then, how do you find the best combination of these two elements so that most of the data is read and processed correctly? Because randomly initialized in the first attempts weights and biases will give naturally unreliable results. To train the net to identify the connection between input and correct output what we do is to introduce a function of cost to make it understand the times in which, at least initially, it guesses the number or not. Using the

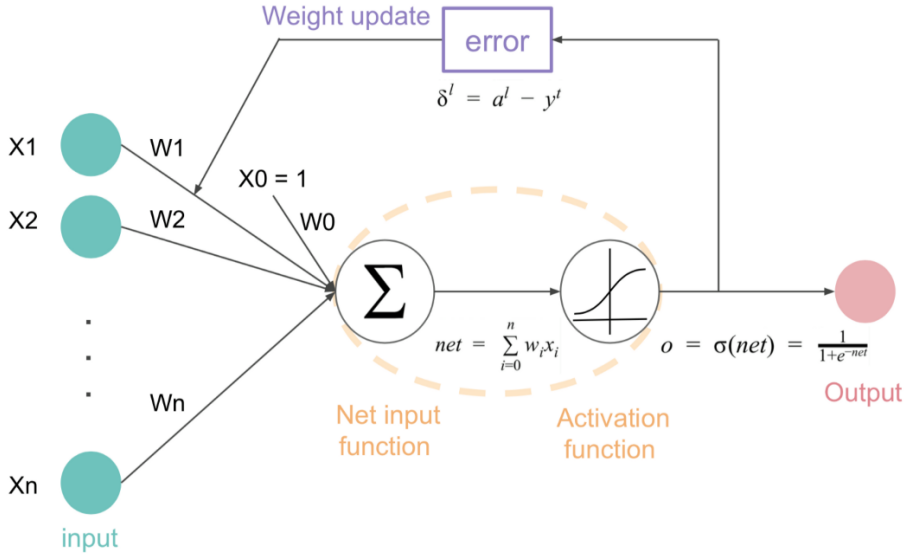

Figure 7.3. Neural network scheme

diagram in the figure 7.3, our goal is to go minimize **w** and **b** in order to minimizes the *error* block. By considering our cost function as:

$$C(w) \tag{7.5}$$

where w is the entire set of weights and biases, we can use the notion learned in Chapter 5 "**Gradient-based methods**" to understand how to find the minimum of 7.5. In fact if we move, starting from a point and imagining the function in a variable, towards the direction with greater slope and with a step proportional

to it, after a certain number of iterations we will satisfy this type of requirement related to the minimization of $C(w)$. A problem with more variables will have to consider instead the direction of maximum descent, given by the operator:

$$-\nabla C(w) \tag{7.6}$$

Despite its derivative. Equation 7.6 will tell us if the corresponding element should increase or decrease its value depending on the sign while the value will tell us how high the relative increase will be.

Backpropagation

The backpropagation is a technique that allows to use methods like the descent of the gradient to obtain in this way a set of weights and bias such to minimize the function of cost. The term "back" derives from the fact that the calculation of the descent of the gradient is performed from the last layer of the net, until the first one, in order to avoid redundancies. Let's consider the concrete case in which our newly initialized network is given the image of a "2". Naturally, due to the nature of the network itself, weights and biases will be initialized randomly, so we should not be afraid to find an output that does not necessarily converge on the neuron associated with the same number of the image. With respect to the process that will be calculated, what will have to be done is a modification on the weights of the final layer in such a way that they "understand" that when the image is similar to the one proposed, a priority is given to the desired number. The weights and biases will then be raised/lowered according to their value. If the activation relative to the desired neuron is low because the output is in fact another the choices we have are basically three, in fact we can:

- Increase $\mathbf{b}$
- Increase $\mathbf{w}_i$
- Change a_i

A useful solution could be given by modifying, wanting to concentrate only on the weights, those values that have more influence on the previous layer, consequently a solution could be to act directly on those that, associated to the number of interest, weigh more than the others. In this way it would be proceeded therefore to backward going to modify the weights of the previous layer "adjusting them" in the measure in which the produced result is that one expected. In this way it will be easier to understand which direction every single weight and bias will have to go, if therefore this will have to increase or decrease suggesting to the function of cost a direction towards which to move using therefore algorithms based on gradient descent.

With this small parenthesis it is concluded the second part of the thesis in which to be described is the theory that is at the base of the instruments used in the realization of the tool IRATO. We will see, in the successive part as some of these techniques have been functional to the realization of the same one while others have been revealed not adapted to the resolution of the problem but not for this didactically useless. I would like to underline how the contact with this kind of tools would never have been so effective if, in addition to the bibliography, I had not also consulted the excellent multimedia material present on the platforms and made even better thanks to the concessions that some sites have deliberated because of the health emergency.

Chapter 8

Problem resolution

Starting from this chapter all that we have seen previously will be resumed and explained in the declination of the resolution of the problem. We review the formulation of this in the form of a mission statement:

"To develop a tool which exploits branching trajectories in cases of failures, trying to obtain the best possible corridors by minimizing the failure impact."

Starting from this we tried to develop through MatLAB a solution that could be as effective as possible. In the following chapter will be described the types of algorithms used and their results.
Regardless of the optimization algorithm employed, as a first step it is fair to explain and show the basic structure of the problem and the mathematical framework behind it. This reasoning would never have been possible if it were not for the advice of my supervisor Jasmine Rimani who provided me with a tool from which to start, what follows is the the chronological evolution of the progress made up to the creation of IRATO.

8.1 Ascent Optimization V1 and V2

[11] Without external advice, what I thought was that having a simulator that could reconstruct the behavior of a launcher could be useful for two reasons: the first related to versatility, working in fact with degraded trajectories the fact of having something that allowed a study not nominal would be as useful as the original trajectory. The second is related to the scalability of the model, to which it would have been sufficient to add elements to improve reliability. In this regard the first solution proposed was the Rocket Dynamic Simulator (RDS), which is discussed in more detail in Chapter 4. Regardless of the conceptual complexity of the program this was quickly debunked. The main reason was the fact that a simulator, as accurate as it can be, is not born with the purpose of "optimizing" a trajectory,

but more to simulate it. Only after having understand the binomial "optimization-useful" I have married this type of objection trying to reimpose therefore a strategy for the elaboration of a job that, instead of analyzing the "free" behavior of my system, it controlled it to the aim to produce the profit of the mission statement. The starting point after gaining this kind of awareness was a set of tools produced by my supervisor called *ascent optimization* $V1$ and $V2$. Both codes produce an optimized trajectory, respectively the first one returns in output the maximum attainable height given the nominal characteristics of a launcher while the second, with the addition of a target height, tries to produce a trajectory minimizing the amount of fuel used.

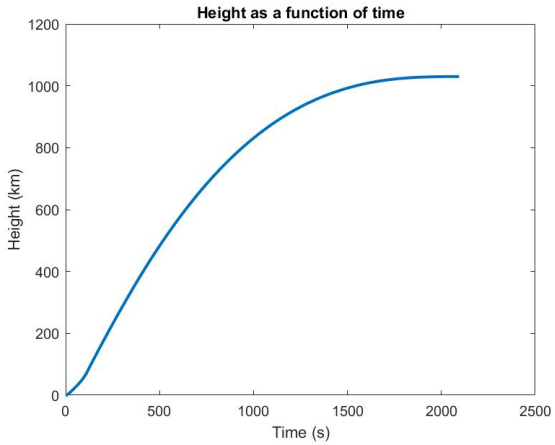

Figure 8.1. Example of fuel consumption optimized trajectory of Ariane V launcher

Although the function of these two tools is slightly different, the way in which the trajectory is reconstructed is the same and is described below. A number of segments N_{seg} are initially defined in which our trajectory will be decomposed. To increase N_{seg} means to have a higher reliability since there is a more precise discretization but it also means to increase a lot the computational cost of the tool itself. Each segment is defined by three main parameters:

- η which is the throttle.
- α which is the thrust angle.
- FF which is the amount of fuel fraction[1] used in that segment.

[1] In ascent optimization V1 there are just η and α because fuel fraction is not a control variable.

Each segment is given as input to a function named *trajectory* that will calculate its resulting trajectory and the amount of mass consumed to make sure that subsequent segments have physical and geometric consistency with each other in order to recreate a continuous result. There is, in fact, the continuous integration of these parameters within characteristic equations of motion that will return in output a state vector p that contains within it the information of:

- r is the radius vector connecting the launcher taken as a point to the center of the Earth.
- θ is the pitch angle.
- V_r is the radial component of rocket velocity.
- V_t is the tangential component of rocket velocity.
- m is the rocket mass.

Equations of motion used are the following:

$$dr = V_r \tag{8.1}$$

$$d\theta = \frac{V_t}{r} \tag{8.2}$$

$$dV_r = \frac{V_t^2}{r} + \frac{\eta T}{m \sin\alpha} - \frac{\mu}{r^2} - \frac{1}{2}\rho_0 \frac{e^{\frac{R_{earth}-r}{h}}}{m} \sqrt{V_t^2 + V_r^2} C_D A V_r \tag{8.3}$$

$$dV_t = -\frac{V_r V_t}{r} + \frac{\eta T}{m \cos\alpha} - \frac{1}{2}\rho_0 \frac{e^{\frac{R_{earth}-r}{h}}}{m} \sqrt{V_t^2 + V_r^2} C_D A V_t \tag{8.4}$$

$$dm = -\frac{\eta T}{g_0 I_{sp}} \tag{8.5}$$

At this point we have therefore present how using a set of values of η, α and FF, it is possible to reconstruct the stretch of trajectory that competes with the relevant segment. The task of selecting suitable values to achieve maximum height in the case of the first version and fuel fraction savings for the second, will be the burden of the optimization technique employed. For both versions of the tool the technique used is the gradient descent, accessible and declined in an optimization problem thanks to the set of functions of the *optimization toolbox* of MatLAB. Among the various functions provided by this toolbox has been used $fmincon$, function used to find minimum of constrained nonlinear multivariable function. The general problem solved by fmincon is the following:

$$min f(x) = \begin{cases} c(x) \leq 0 \\ c_{eq}(x) = 0 \\ A \cdot x \leq b \\ A_{eq} \cdot x = b_{eq} \\ lb \leq x \leq ub \end{cases} \tag{8.6}$$

This kind of problem is easily solved and described by this line of code [18]:

```
[x,fval,exitflag,output,lambda,grad,hessian] = fmincon(objective,x0,[],...
...[],[],[],lb,ub,@(x) mycon(x,data),options);
```

Speaking as a human, one might say that to produce an optimal set of two $N_{segment}$ $\times$ N_{stage} matrices and possibly a fuel fraction value (all contained within the x vector), what is needed is an initial attempt, an objective, i.e., the value you intend to minimize[2], and a limitation to the control parameters in such a way as to preserve a geometric and physical consistency with the reality of the problem I am addressing. To also communicate to the code the fact that we impose some equality constraints on the minimization process in order to make sure that the problem is done within a mission logic we use the following function, also belonging to the input of fmincon. Lingering on the function:

```
mycon(x,data);
```

We can see how this, received as input the set of control parameters, makes sure that this leads to a trajectory on a circular orbit at a more or less desired distance. These type of condition is imposed by three equality constraints that are represented by the following equations:

$$\frac{V_r}{V_{scale}} = 0 \tag{8.7}$$

$$\frac{V_t - \sqrt{\frac{\mu}{r}}}{V_{scale}} = 0 \tag{8.8}$$

$$\frac{r - r_{target}}{r_{earth}} = 0 \tag{8.9}$$

Focusing then on the second version of the program we can analyze in Figure 8.2 the results that the program offers us taking as a reference model a mission in which Ariane-V aims at an altitude of 1030 km. The execution time is 196.0 s.

[2]In Ascent Optimization V2 this will be the fuel fraction

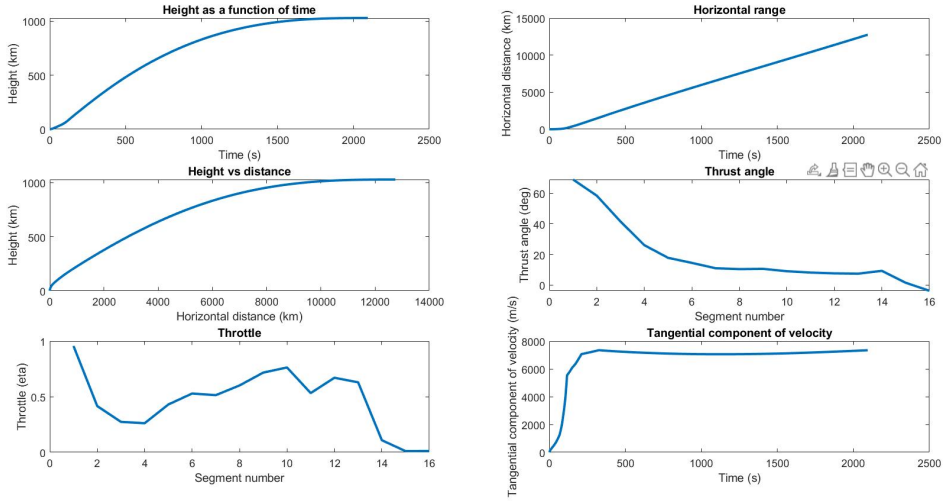

Figure 8.2. Optimized trajectory computed through *ascent optimization* V_2

8.2 Implementation of branching trajectories

What we have seen so far regarding the two tools we have just discussed is useful in understanding how an optimization problem is addressed within a mission analysis discussion. We need a data set to start with, state variables and control variables. However, our reasoning must take into account the fact that at a certain time a failure occurs that alters the initial characteristics of my starting problem. It is obvious therefore like there is the necessity to intervene in the middle of the process of optimization in order to communicate to my code the fact that some condition must be modified.
The logical process through which it has been applied this type of reasoning has carried first of all to ask itself like a failure could be represented and interpreted from the code. A solution that turned out to be quite simple was to reduce the field of my state variables (η and α) that could derive from an operational failure, respectively two example cases could be a problem related to the propulsion feed system for the first and a problem related to the gimbal lock of the thruster for the second. With this reasoning the problem was carried forward by adding to the optimization seen in the section 8.1 another additional optimization step. Compared to the previous code the use of $fmincon$ was used not once, but twice so that the second optimization could take as initial conditions not those of the launcher at the moment of lift off, but those at a specific time that will coincide with our failure. Consistently with what has been said before, not only the initial values of the state vector have been modified assuming a certain percentage of deterioration resulting from the operational failure. Therefore to appreciate this type of modification the outgoing

trajectory will be organized according to the nominal optimization until the time of failure, after which the data of the second optimization will be used. This type of solution unfortunately did not seem to be consistent with the constraints imposed by using *fmincon*. A further problem, more of a conceptual nature, arose since it would have been impossible to predict and therefore tell the algorithm what the maximum altitude reachable after failure would be, thus leading to non-convergence of the code. So a new type of solution was immediately considered that would take into account both giving a physically and geometrically consistent connection to the true branches of my trajectory, and directing my launcher to a new type of maximum attainable altitude. The second strategy took the utility of the two codes in section 8.1 to set up the problem as it is described in table 8.2:

Code	Usage
Ascent Optimization V2	Generates the nominal trajectory with possible fuel savings.
FAILURE	
Ascent Optimization V1	After the failure, Lower Bound (LB) and Upper Bound (UB) change. The maximum height that can be reached is calculated and it will be compare to the previous target.
Ascent Optimization V2	Generates the branched trajectory with possible fuel savings towards the new target.

Table 8.1. Logic flow of the second optimization attempt

Following the scheme of the table 8.2 it is easy to understand how the problem has been set up by playing with the results provided by the single codes. Without changing the anatomy of the code, a first use of ascent optimization V2 will calculate the nominal trajectory, then, after a failure will be modified lower and upper bound of my state variables and ascent optimization V1 will update the value of maximum height reachable to intervene with a last use of the first code and then update, always optimizing, the trajectory with the new values. This type of strategy has turned out not suitable for two reasons: the first tied up to the fact that, even if degraded, the maximum attainable height in some cases exceeded that target going therefore to make to lose to the code a coherence from the conceptual point of view. The second deriving from the double use of *fmincon* that, for connection problems, has not produced reliable results (as you can see in Figure 8.3) from the physical point of view.

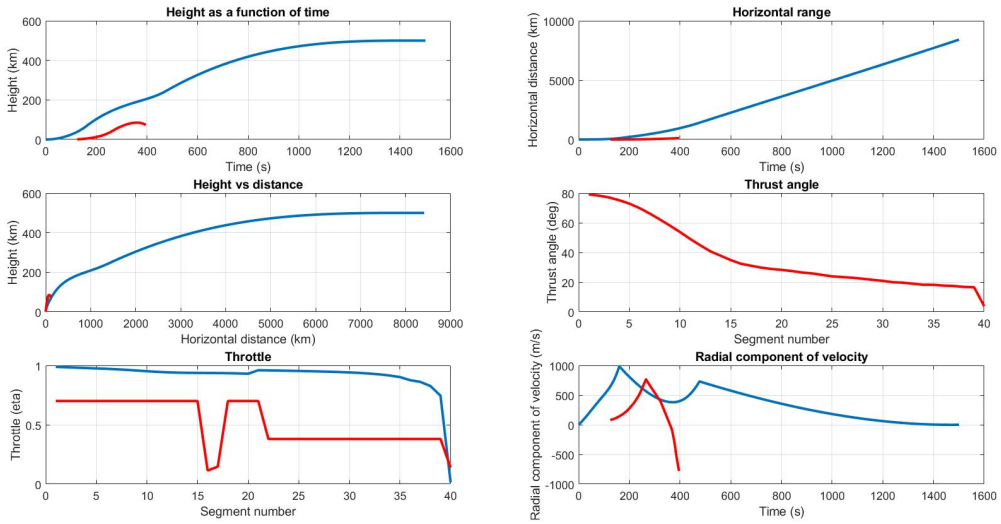

Figure 8.3. Failure scenario in which at $t = 180$ s there is a 30% deterioration of the maximum η value

8.3 PSO application

[5],[6],[7],[15] Given the previous results, it became clear that applying a gradient-based algorithm was not the best way to proceed with producing reliable results. We have seen like the first attempt has used the employment of ascent optimization V2 for two times in concatenated way while in the second case products from both versions of the tool have been used . However this has not seemed to be the best way. Taking a step back was the only plausible solution, investigating even deeper the nature of the optimization algorithms, considering this time also those not necessarily based on gradient descent. This type of process first involved research into commercial software that can solve so-called "multiphase" problems. Although these are present, the cost to make them accessible is too high, so I preferred to continue on my way producing a result probably less accurate but certainly more open to the community. The strength and difficulty in tackling this thesis topic was just that: although the reader can realize the simplicity of the results and how they were derived, it was difficult to investigate what was the best method for solving the problem. This, in addition to being challenging, was at the same time very important to the didactic learning that took place for the most part independently. The PSO algorithm carries with it a very nice moral: collaboration between individuals is the only way to achieve an optimum. A swarm paradoxically composed of one element does not have the same effectiveness as one composed of a hundred, not because the intrinsic ability of the unit increases, but because by exchanging

information between them a group is better oriented towards the solution of a problem. This feeling of sharing was the same sense that brought me to this algorithm. In addition to a great step forward for the thesis results, it was a wonderful human lesson and I hope that I learned a lot from this experience for what will be my future path.
For what concerns the theory behind the code we refer to section 6. What will be described in this section is how this algorithm was declined in the optimization of the branched trajectories of a launcher and the related discussion of the results.
Let's briefly reconstruct the logic of the problem to understand how this tool was applied: throttle η, thrust angle α and fuel fraction were chosen as state variables. The trajectory is managed in a discrete way by dividing each section, defined by its stage, into N segments. Considering for example $N_{segment} = 4$, our state variables will be:

Stage 1	Stage 2
$\eta_{1,1}$	$\eta_{1,2}$
$\eta_{2,1}$	$\eta_{2,2}$
$\eta_{3,1}$	$\eta_{3,2}$
$\eta_{4,1}$	$\eta_{4,2}$

Stage 1	Stage 2
$\alpha_{1,1}$	$\alpha_{1,2}$
$\alpha_{2,1}$	$\alpha_{2,2}$
$\alpha_{3,1}$	$\alpha_{3,2}$
$\alpha_{4,1}$	$\alpha_{4,2}$

Another scalar value indicates the amount of fuel saved (varying therefore between 0 and 1) that we will indicate in notation as FF. In our case, taking Electron as a reference, a set consisting of $N_{stage} = 2$ and $N_{segment} = 8$ was chosen.we are going to define the variables contained within each particle and their relative maximum and minimum values. From the previous reasoning, it is not difficult to see how these are:

$$2(N_{segment} \times N_{stage}) + 1$$

Then in our specific case, each particle will be composed by 33 variables (16 η values, 16 α values and one related to the fuel fraction saved).
Variables will be respectively constrained within a field of existence so that the assumed values are consistent with the physics of the problem, these are:

- $0.01 \leq \eta \leq 1$
- $-\frac{\pi}{18} \leq \alpha \leq \frac{\pi}{2}$

- $0.95 \leq FF \leq 0.99$

Since each particle contains a potential trajectory, a cost function will be defined within this section that, given the trajectory, will weight both its physical consistency with the nature of the problem and its actual savings in terms of fuel fraction. The figures of merit that make up the cost function are normalized quantities that will compare the final velocity and position values of my launcher against the reference imposed by the mission with the addition of the Fuel Fraction (FF) used. In this way for the nominal trajectory the following equation was chosen as the cost function:

$$C = 25\ FF + 2\ \|V_{r,norm}\| + 2\ \|V_{t,norm}\| + 40\ \|r_{norm}\| \tag{8.10}$$

By associating weights with each figure of merit, we are able to establish a hierarchy of importance of the conditions that must be met depending on the situation in which we are dealing with (nominal or degraded trajectory). We note from the equation 8.10 how achieving the desired height is the priority with eventual savings of a certain amount of fuel. Radial and tangential velocity conditions do not have much weight as long as they are respected without any particular imposition. This strategy produced results that are proposed as the final solution to the problem, consequently the logical procedure will be described in detail in the next section.

8.4 Intelligent Rocket Ascent Trajectory Optimizer

IRATO is the acronym of the tool that allowed to produce the results that will be proposed to the reader built on the theoretical concepts of which we have spoken so far and their relative collaboration. Let's move on to the scheme of our algorithm which will be divided as follows:

1) Data definition

To build a model that was as accurate as possible, reference was made to the data that make up Electron and its standard missions. 500 Km have been considered as target altitude since the Electron reference mission sees it at this altitude in sun-synchronous orbit. The launcher data are described in Table 8.4. For consistency issues in this section, the number of segments into which the trajectory (both nominal and degraded) will be divided is defined so that the relative accuracy does not change from one to the other.

2) Nominal problem definition

This section defines my problem from the elementary unit of PSO: the particle.

Environment data		
g_0	$9.80665 \frac{m}{s^2}$	Standard gravity
μ	$3.98600441810 \times 10^{14} \frac{km^3}{s^2}$	Standard gravitational parameter of the Earth
R_{earth}	6371 km	Radius of Earth
$\rho_{air,0}$	$1.2754 \frac{kg}{m^3}$	Air density at sea level
ASH	8.5 km	Atmosphere scale height
Launcher data		
T_1	224190.37 N	1st stage thrust at sea level
T_2	25799.69 N	2nd stage thrust in vacuum
$I_{sp,1}$	311 s	1st stage specific impulse
$I_{sp,2}$	343 s	2nd stage specific impulse
$m_{1,fuel}$	9250 kg	1st stage fuel mass
$m_{1,structure}$	950 kg	1st stage structure mass
$m_{2,fuel}$	2050 kg	2nd stage fuel mass
$m_{2,structure}$	250 kg	2nd stage structure mass
$m_{payload}$	150 kg	Payload mass
$C_{D,1}$	0.5	1st stage aerodynamic drag coefficient
$C_{D,2}$	0.25	2nd stage aerodynamic drag coefficient
A	1.13 m^2	Cross section area

Table 8.2. Electron data for trajectory optimization

It will then be defined how many variables it should contain and what are their boundaries. The best combination of parameters, which will then have minimized my cost function[3], will be the solution to our problem when the rocket is in nominal phase.

3) Nominal trajectory optimization

Before iterating we need to define how many particles our swarm will be composed of and how many iterations to do. To define whether an attempt is a reliable choice we use benchmark functions whose global minima are known. To increase the number of particles means to obtain a wider coverage of the topology of the studied function but also to increase the computational time of the code. In our case a set of 100 particles and 100 iterations was chosen, this combination provides reliable results when compared to those produced by *fmincon* and an acceptable computational time (about two minutes). The choice of parameters related to particle dynamics were selected from the results of Clerck and Kennedy showed in 6.11, 6.12 and 6.13. Then the algorithm starts running by initializing the values

[3]For the nominal trajectory the equation is 8.10

contained in the particles in a uniformly random pattern in the space in which they are confined. After each iteration, the i-th particle will compare the score derived from the cost function with two values: the first relative to its personal best and the second relative to the global best identified by the group. On the basis of this type of correspondence will be decreed the area towards which to move and the relative speed with which to do so. At the end of the iteration process the lowest score that the cost function has reached and its coordinate within the space defined by the state variables is saved. A function named *trajectory* will receive in input the final values of α, η and FF in order to reconstruct the trajectory providing in output the relative quantities:

- r is the radius vector connecting the launcher taken as a point to the center of the Earth.
- θ is the pitch angle.
- V_r is the radial component of rocket velocity.
- V_t is the tangential component of rocket velocity.
- m is the rocket mass.

4) Failure generation

The first step toward conceptualizing a failure was to find a link between an operational error related to the mission phase and a numerical value of performance degradation. This type of step was covered theoretically in later chapters and for now the explanation will bypass this process. The idea is to find a class of accidents that will limit the range of existence of the control variables, especially η and α in order to observe how the trajectory changes. These two limitations can respectively result from a feed system related problem in the case of η while for α the problem can be traced back to the thruster gimbal. It is important for the algorithm to know how to estimate, given the time istant at which the failure occurs and the percentage lost of the two control variables, what the maximum attainable height is. In this way it will be possible to have a prediction of what will happen and consequently decide on the future of the mission. Since the optimization process would have required a third use of the PSO thus increasing the computational time of the final product, it was decided to adopt a neural network for the prediction of this value. Based on what was said in the section 7.2.1 to train such a network we start with a data set that will be used to train weights and biases that, once optimized, will weigh a new input in order to provide a solution that is as accurate as possible. The code, in order to carry out this study, was therefore momentarily repurposed as follows. Fixed a nominal trajectory was varied the range of $t_{failure}$, η_{max} and α_{max} in a random way. Then the final height value was saved within a text file in such a way as to create a database as shown in Table 8.3.

$t_{failure}$(s)	η_{ub} percentage decrease	α_{ub} percentage decrease	Final height (km)
263	51	82	499.99
451	44	25	483
27	54	80	351.47
...	...	...	...

Table 8.3. Database layout

Considering the scenario in the first row as an example, we can see that at time $t = 263s$ the upper bound of the angle of attack goes from the value $\alpha = \pi/2$ to approximately $\alpha = \pi/4$ while the throttle is limited to a maximum value of $\eta = 0.82$. The dataset used consists of 987 events calculated over a three-day computation period. Data processing through AI techniques is made easier by tools and libraries available for programming languages and numerical computing software. For coherence reasons we have always chosen the MatLAB environment which, thanks to nntool, allows the setup and training of a neural network. Setting up the neural network is done according to Figure 8.4.

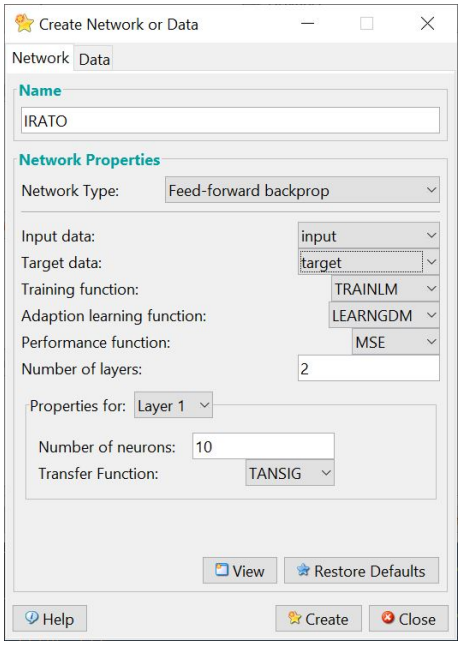

Figure 8.4. Neural network configuration

The input is given by the tern $t_{failure}$, α and η while the target will be the final height. The type of network follows the model of feed backpropagation and is composed of two layers, the first consisting of 10 neurons with sigmoid activation,

while the second consists of 2 and linear activation. A graphic concept of the structure is shown in Figure 8.5. In this case, as in the nominal, the cost function

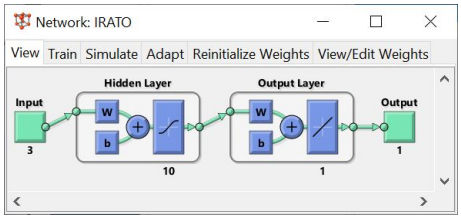

Figure 8.5. Neural network scheme

used is always the 8.10. In both cases (nominal and branched) what is prioritized is the fuel fraction savings, which occurs on average with a 5% fuel saving relative to total mass. Neural network performance from training with the dataset named earlier was not great. This may be justified by the fact that the combinations of $t_{failure}$, and upper bound reduction of α and η are too many to be evaluated with only 987 different scenarios. This can also be demonstrated by the network performance shown below.

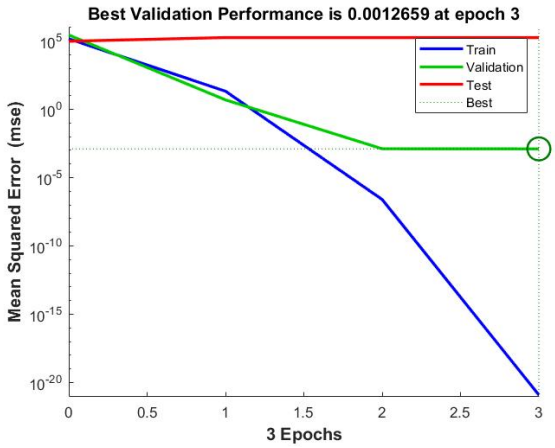

Figure 8.6. Neural network performance

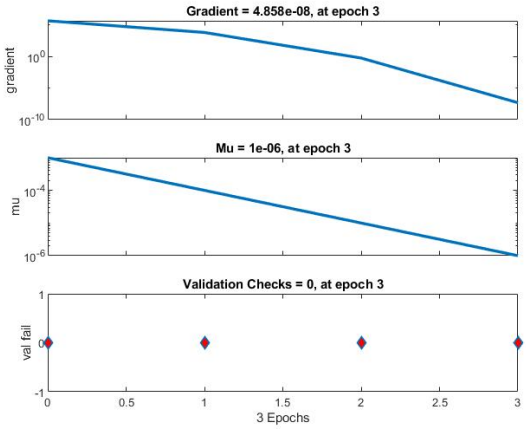

Figure 8.7. Validation checks and gradient descent

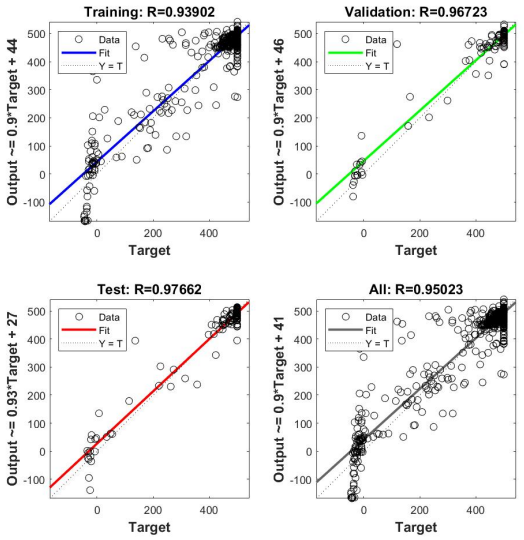

Figure 8.8. Regressions

The alternatives in order to improve this type of result are two: the first one increase the number of simulations while the second one is to reduce the field of the considered variables. Since the computational power available has resulted insufficient to continue with the first way it has been decided to discretize the quantities in the following way:

- $t_{failure}$ from 20 s to 500 s with step 10 s
- η_{ub} and α_{ub} percentage decrease from 0.2 to 1 with step 0.1

In this way, more circumscribed and better catalogued data were obtained. The simulated scenarios numbered more than 1000 and took a computational time of four days to compute. The results for the performance of the second version of the network are shown below.

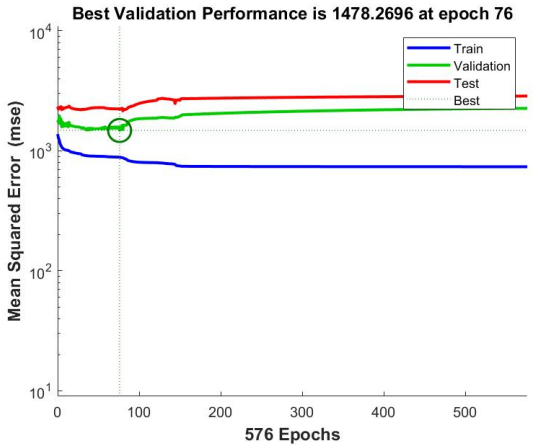

Figure 8.9. Neural network performance

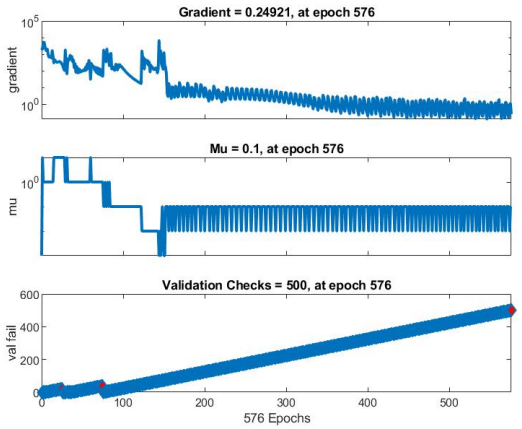

Figure 8.10. Validation checks and gradient descent

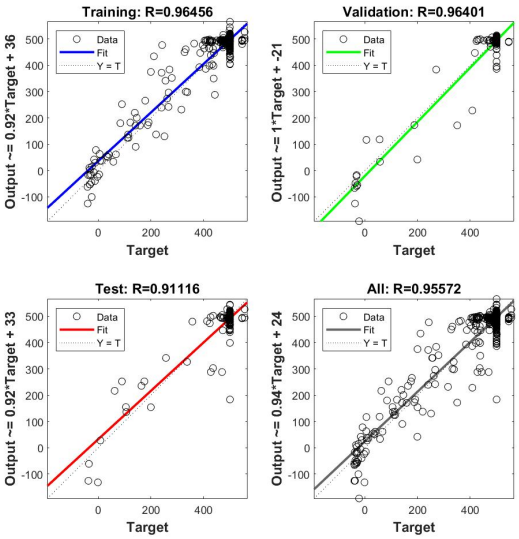

Figure 8.11. Regressions

5) Branched problem definition

The definition of the branching problem follows almost the definition of the nominal problem with some modifications. The number of segments is kept equal to that of the starting problem to avoid problems arising from inconsistency between the discretization of the two trajectories, therefore, according to what has been said, it has been chosen a $N_{segment} = 8$. The definition and limitation of the control variables has remained constant while the cost function has undergone modifications, which, following the type of phase in which we find ourselves, will allow the execution of the relative trajectory. The fact that the optimization takes place by modifying only the coefficients linked to the figures of merit of the cost function is a strong point of the program, in fact there will be no need to change other characteristics of the code, thus lightening the programmer's task. After receiving from the neural network the updated information about the target height after failure the user can decide whether to continue the mission or abort, to make this process happen automatically was set the condition:

$$\| r - r_{target} \| < 100 \tag{8.11}$$

If respected, it means that the height at which my launcher can arrive after failure is between 400 km and 600 km considering a target of 500 km, in this case it will continue to orbit excluding the condition on fuel saving. The merit function associated to this situation is the following:

$$C_{branched} = 2\ FF + 10 \parallel V_{r,norm} \parallel + 10 \parallel V_{t,norm} \parallel + 30 \parallel r_{norm} \parallel \tag{8.12}$$

As can be seen the structure is similar to 8.10 but the figures of merit are weighted differently, in fact more importance is given to the achievement of the final orbit close to the fuel consumption used. This kind of result will almost completely zero out the importance given to fuel economy focusing the mission objective on reaching orbit. In an abort trajectory scenario modifications more relevant have been executed altering the structure of the functions of cost seen until now. A prerogative of the problem statement is that the branched trajectory went, in the case of abortion, to limit the damages caused by the launcher. An example provided was the impact with population centers, consequently the study focused on this aspect with the introduction of a new figure of merit.

$$r_{obst} = 1 - \frac{\|d_{FC} - d_{FS}\|}{d_{FS}} \tag{8.13}$$

Where:

- d_{FC} is the horizontal distance between the launch site facility and the interested city.
- d_{FS} is the horizontal distance travelled by the launcher starting from the launch site.

The information to avoid the obstacle is not passed as quantitative but qualitative, in fact it will be the launcher, evaluating the instant in which the failure occurs and what kind of limitations this implies, to decide autonomously and to maximize as much as possible the distance to that ground obstacle. A schematization of the geometric problem is shown below in figure 8.12.

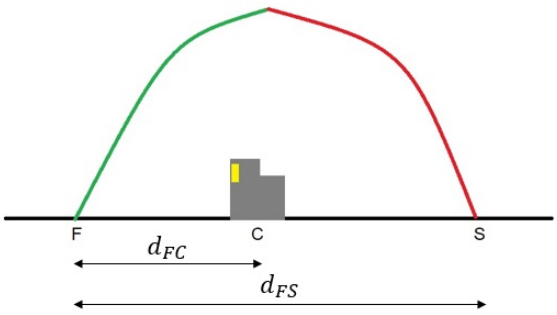

Figure 8.12. Schematic representation of r_{obst} quantities

The figure of merit that has been associated with this type of situation is the equation 8.14 :

$$C_{abort} = 30 \parallel r_{norm} \parallel + 30 \parallel r_{obst} \parallel \tag{8.14}$$

The geometry of the new problem is also defined at this stage. Once the user has entered the time at which the failure occurs, a function ($findsegment$) will associate the input to the relative segment. The previous values of α and η will be saved as they are constituents of the trajectory that until that moment is carried out in nominal phase. The number of variables will be given instead by the total number of segments to which the nominal part is subtracted. There is also an update of the upper bound values resulting from the failure.

6) Branched trajectory optimization

At this step the PSO is applied for the second time. Each particle, compared to the previous case, will have a smaller number of variables since all throttle and thrust angle values prior to failure will have already been calculated by the first iteration of the algorithm. Although my problem is discretized in segments it is evident how this algorithm is useful in multiphase optimization of a problem. The schematic layout of this process is showed in figure 8.13.

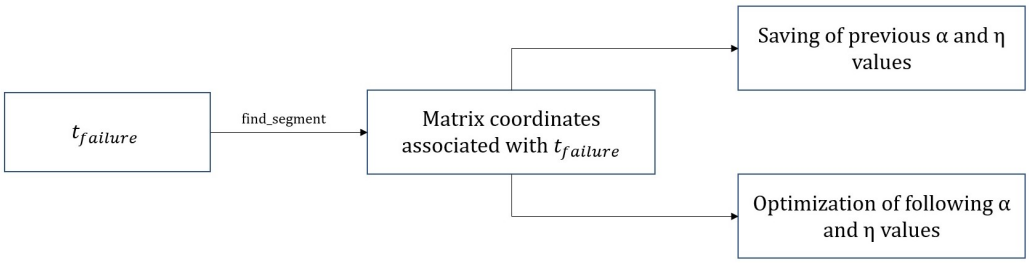

Figure 8.13. Schematic representation of branched problem identification

7) Results

Once the nominal and branched process has been optimized, the results of the analysis can be appreciated. The trajectory study was conducted by considering the launcher dynamics as point-like. Performing a 6-DoF analysis would take a lot of time and data resources involving the geometry of the considered launcher. Launchers, unlike payloads and other spacecraft, require less accuracy in terms of attitude precision due to their inherent nature. However, it might think about combining the results from IRATO combined with RDS (Chapter 4) to set the continuation of the work toward this direction. The work produced is a non-ballistic multiphase trajectory analysis (thus implementing staging) . Results as a function of two main quantities are shown below:

- Time

- Number of segment

The results as a function of time show how the position (and its derivatives) varies . In this way it is possible to appreciate the time evolution of the 3-D trajectory decomposed into its horizontal and vertical components. The number of segments was chosen to instead appreciate how the state variables change along the segment considered. Of course, the link between time and number of segments is present since each segment consists of a certain amount of seconds that vary depending on the optimization. A scenario is shown below as an example case, failure data are $t_{failure} = 120s$, η_{ub} and α_{ub} percentage decrease are respectively 0.3 and 0.5. The new target predicted by the neural networks is 487 km thus the mission will continue toward its completion.

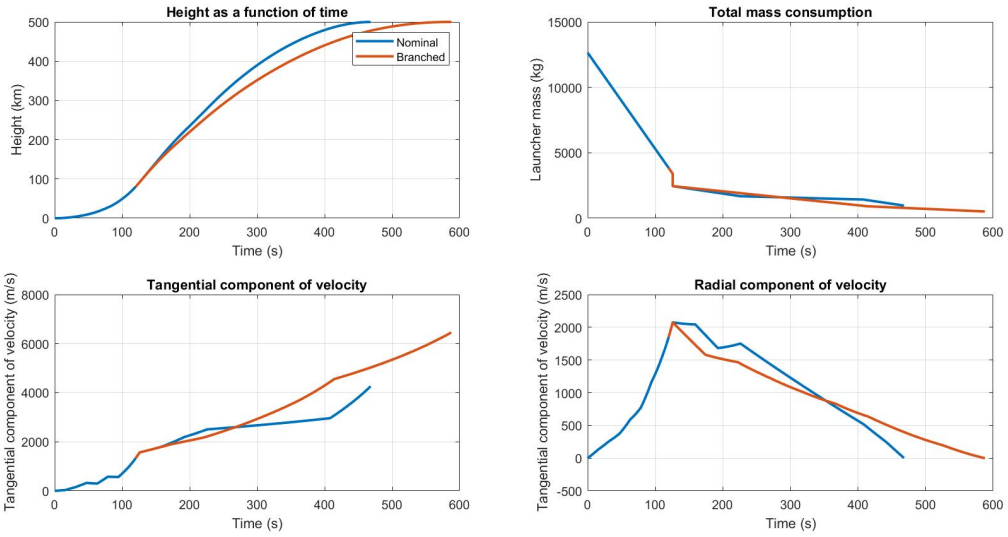

Figure 8.14. Heigth, velocity and mass consumption in branched case

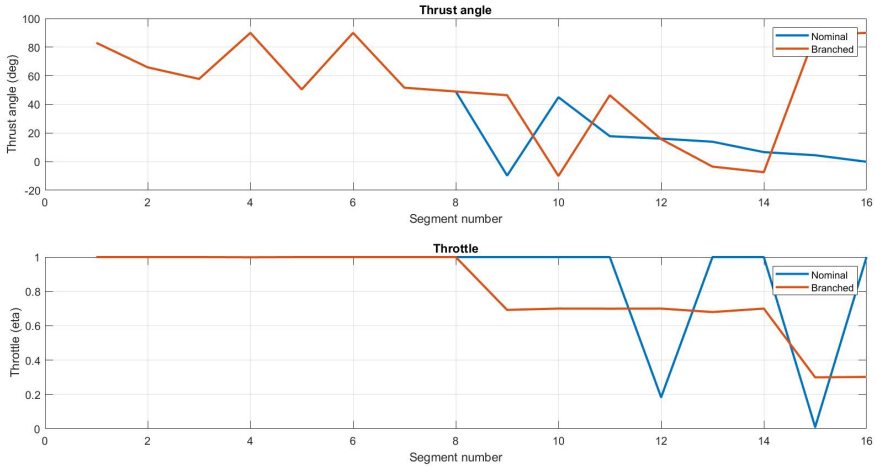

Figure 8.15. Throttle and angle of attack as a function of the number of segments in branched case

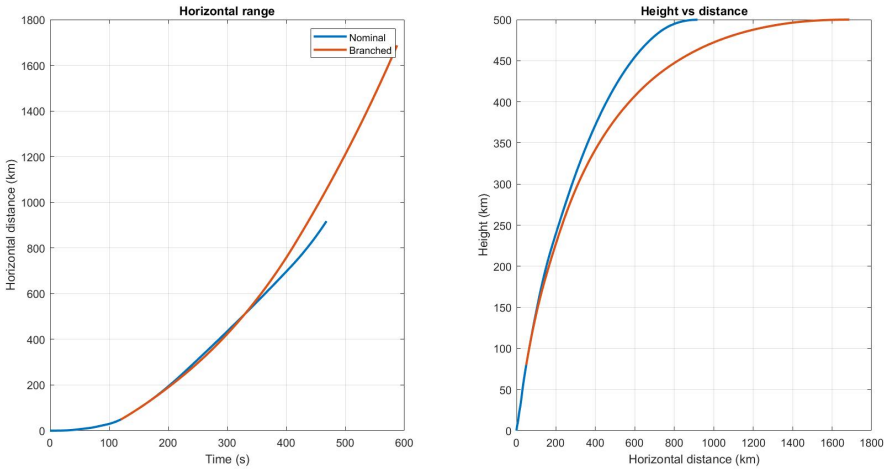

Figure 8.16. Absolute and relative horizontal distance results in branched case

For completeness it is brought back the same result considering a failure with different data. $t_{failure} = 120s$, η_{ub} and α_{ub} percentage decrease are respectively 0.3 and 0.5. The new target predicted by the neural networks is -35.0.2 km thus the mission will abort avoiding population center 250 km away from the launch site.

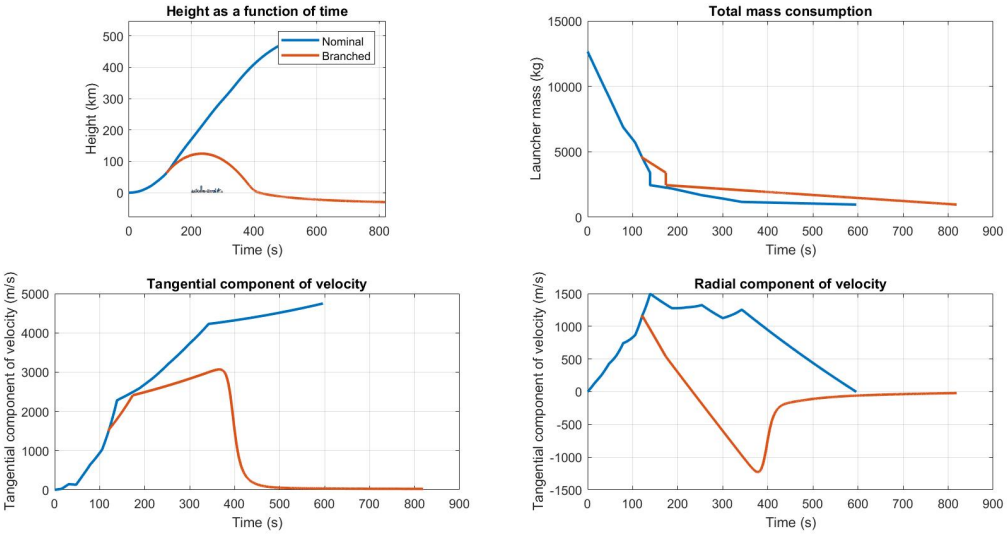

Figure 8.17. Heigth, velocity and mass consumption in abort case

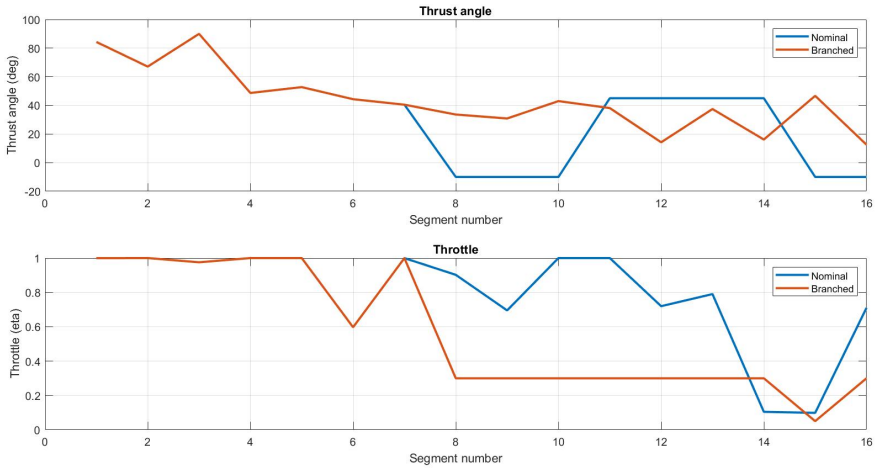

Figure 8.18. Throttle and angle of attack as a function of the number of segments in abort case

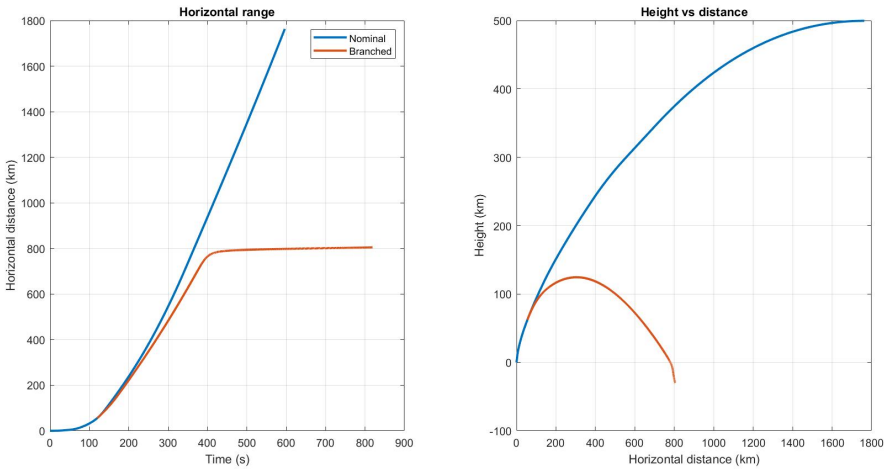

Figure 8.19. Absolute and relative horizontal distance results in abort case

This part concludes after showing the theory behind the tools used in producing the final results and the application, with related theoretical insight, of the best strategy. In the next chapter, the methods that provide most reliability to this product are shown.

Chapter 9

ASTOS Validation

[12] In this chapter difference between ASTOS scenarios and IRATO results will be presented. ASTOS is software used for a complete mission analysis overview for the most of vehicles. Through the creation of an environment, either related to the vehicle or the external environment allows an overall study with results concerning dynamics, aerothermodynamics, attitude dynamics, and control theory. For this reason, it has been used as a confirmation software to validate the results produced by IRATO. The following chapter is arranged to present to the reader a step by step process of the model building, then the comparison between the two models will be introduced.

9.1 Electron model creation

9.1.1 Environment definition

To be modeled for first it is the atmosphere, that is the universe in which my simulation comes tested. Within this section the celestial bodies present within my study and their atmospheric characterization can be defined. Since the problem related to the ascent of a launcher involves the Earth up to its LEO orbit, this is the only planet that is defined. The geometry connected to it is the spheroid going to define an equatorial radius of 6378.137 Km and a polar radius of 6356.752 Km. The modeling is done with zonal coefficients making non-zero as the only coefficient $J2 = 0.0010827$. The atmosphere inserted is the default US Standard 76.

9.1.2 Actuators definition

In this section to be defined are all the elements of the system that act on it in such a way as to generate a control. The simplicity of Electron is in its architectural simplicity since the motor used is always the same, that is the Rutherford. A system

of 9 motors will power the first stage while the second stage will consist of only one. The engine 1st stage input specifications are as follows :

- Nozzle area = 0.0345 m^2
- Vacuum thrust = 21.11 kN
- Vacuum I_{sp} = 303 s

Meanwhile the engine 2nd stage input specifications are as follows:

- Nozzle area = 0.0345 m^2
- Vacuum thrust = 25.8 kN
- Vacuum I_{sp} = 343 s

9.1.3 Aerodynamics definition

The aerodynamic force was entered simply by referring to a 1.2 m diameter profile. Next, a list of force coefficients was introduced as the Mach varies along the x_{body} axis of the rocket as follows:

Name	Mach Number	Data
Unit	None	None
1	0.1	0.3224
2	0.4	0.3108
3	0.8	0.2864
4	1.0	0.7373
5	1.2	0.7994
6	2.0	0.6086
7	3.0	0.4622
8	4.0	0.3767
9	6.0	0.2904
10	10.0	0.2904

Table 9.1. Variation of resistance coefficient as a function of Mach number

9.1.4 Component definition

Components are the active and non-active parts that make up the system under consideration. For the sake of simplicity, only the two stages and the payload will be defined without taking into account additional components such as the fairing

and the jettisonable batteries. The data entered for the first and second stage are respectively:

Dimension	Value	Unit
X	12.0	m
Y	1.2	m
Z	1.2	m
Structural Mass	950	kg
Propellant Mass	9250	kg

Table 9.2. 1_{st} Stage dimensions

Dimension	Value	Unit
X	2.4	m
Y	1.2	m
Z	1.2	m
Structural Mass	200	kg
Propellant Mass	2300	kg

Table 9.3. 2_{nd} Stage dimensions

Next, after defining components and engines in section 9.1.2, these will be assembled both geometrically and physically in the **Vehicles and POI definition** section. Once this segment is finished it is available to appreciate through a GUI the product of what has been built as shown in Figure 9.1.

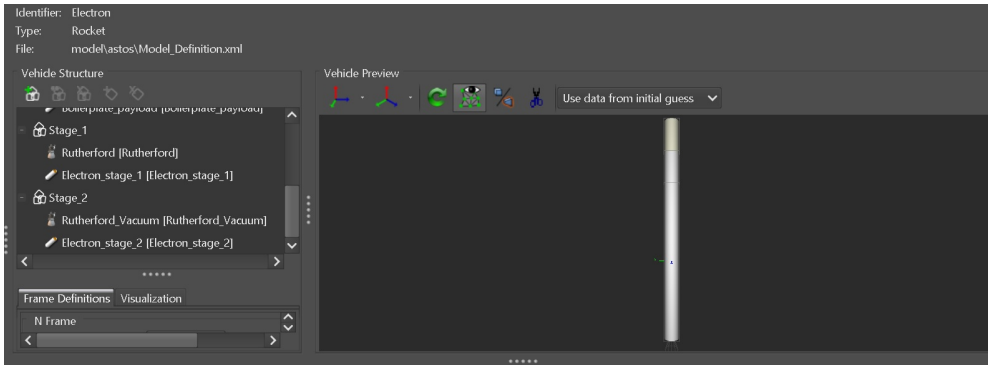

Figure 9.1. Electron vehicle preview

9.1.5 Phase definition

At this step it is necessary to define the various phases that characterize the mission. Regarding the study of the dynamics of a launcher this process is very complicated because, regardless of the nominal phases that make up its trajectory, estimating a range within which these dynamics occur is never easy. Moreover, to characterize the various phases there are discontinuities due to events typical of this rocket launcher such as:

- Change in attitude control laws due to different maneuvers.
- Change of forces and aerodynamic laws acting on the launcher.
- Change in boundary conditions
- Mass change due to detachment of launcher parts

The steps that typically involve the trajectory of a launcher will now be examined. This investigation is useful on a theoretical level and serves to better understand how this section was set up in the Electron case study. Typical phases are as follows [9]:

- Launcher locked at the pad during the ignition
- Vertical ascent with constant yaw pointing in the desired direction (90° orbit inclination) until a certain altitude is reached to avoid the collision with the launch pad.
- The pitch over maneuver is used to give the rocket a direction. It uses a linear pitch law and a constant yaw law. The pitch rate is characteristic for specific launchers: heavy launchers (Ariane 5) around -1°/sec, small launchers up to -3°/sec or more.
- An intermediate constant pitch phase is required to meet the gravity turn conditions: flight path angle equal to pitch angle or angle of attack equal to zero. The yaw angle is constant.
- The gravity turn phase follows until the aerodynamic forces are small enough for high angles of attack. In the beginning the yaw angle remains constant. It depends on the launcher, how many phases fly with gravity turn and at which time point yaw control is allowed.
- For sure after jettisoning the fairing (low dynamic pressure) the attitude control is free. During the initial guess, required-velocity guidance can be used. An optimizable configuration normally uses optimizable controls for yaw and pitch.
- Coast arcs should be modeled using a linear law.

9.2 IRATO Validation

The validation is a very important process because it allows to judge the data obtained in an amateur way and to compare them with a simulation produced by an advanced software, in this case ASTOS. The model described in Section 9.1.1 if optimized produces reliable results given the more than accurate description of the micro launcher, the environment and the systems that compose it. As an example, three examples are reported that show the comparison between scenarios produced through the use of IRATO and ASTOS in order to validate the results of the first tool.
The first case shown in Figure 9.2 compares an ascent scenario with a target height of 200 km during which no failure occurs in a way that validates the nominal situation.

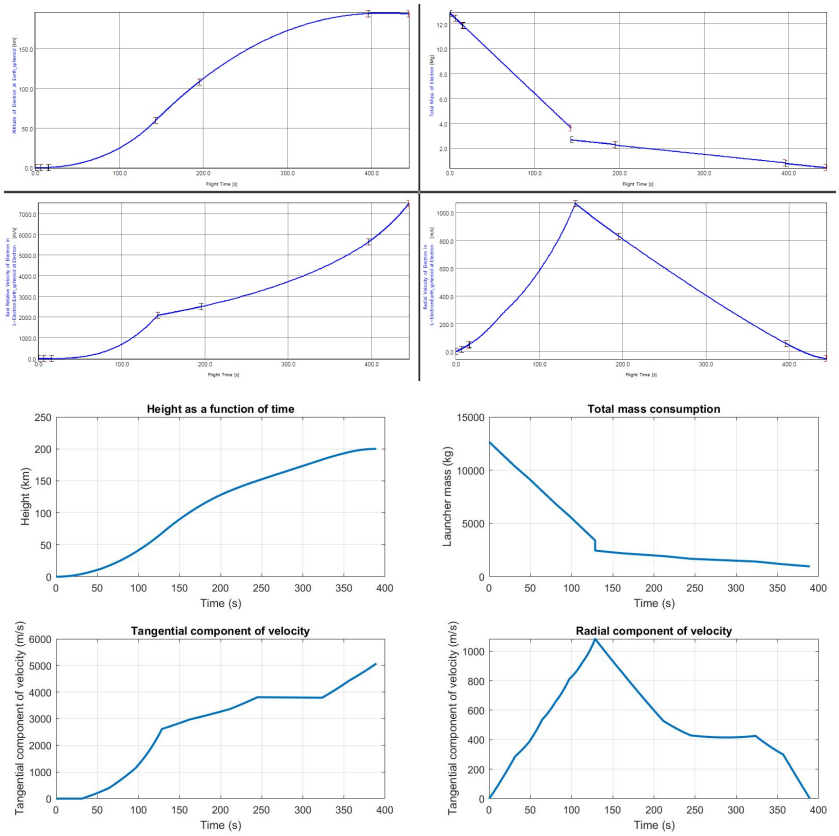

Figure 9.2. Nominal trajectory scenario validation

The second case shows the same mission but with a failure at $t = 100s$ that will limit its maximum value in terms of throttle percentage to 30%. Considering the

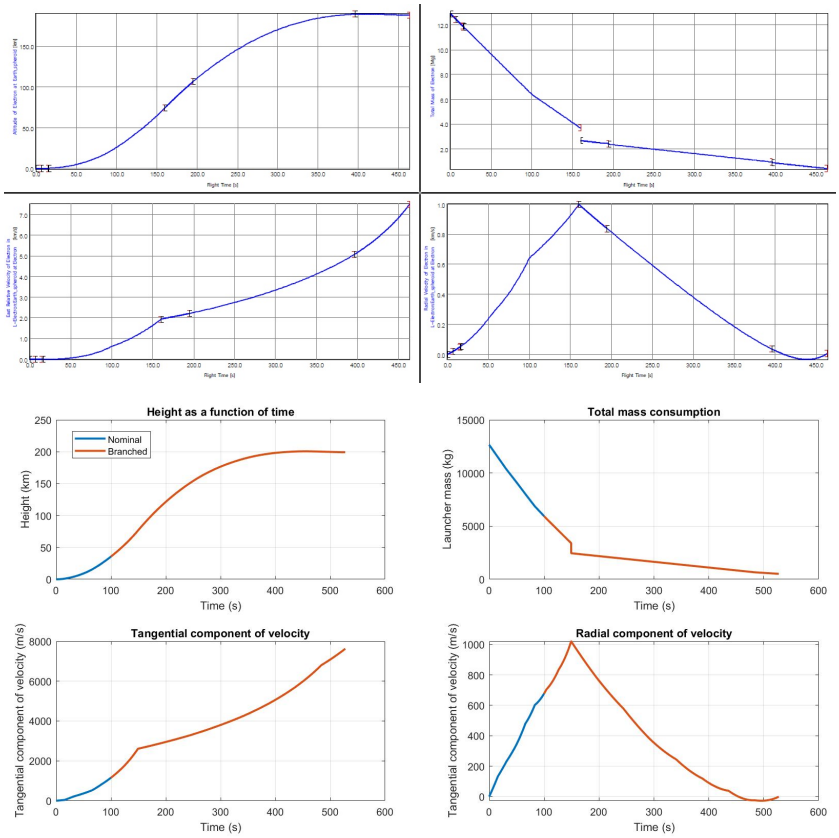

Figure 9.3. Degraded trajectory scenario validation ($t_{failure} = 100s$ and 30% of throttle upper bound reduction)

two examples cited in Figure 9.2 and 9.3 the reader has demonstration of how the results were validated using professional mission analysis software.

Conlusions

This thesis presents a summary of the work performed over seven months working independently following a request from ESA's TEC-MPA department with the engineers Stephan Schuster and Guillermo Ortega. The solution presents a methodology to study the branching ascent trajectories of a micro-launcher in nominal and degraded phases. There were several benefits to this study, the first coming from the nominal trajectory study.
Taking Electron as an example, it has been demonstrated how an efficient optimization can save, relatively to the simplified model considered in the analysis, up to 5% of fuel compared to the datasheet. At the same time it has been possible to study the behavior of the launcher when an operational failure is generated, the optimization will take into account from that moment the new altered characteristics evaluating if the launcher can still reach the target or abort.
In the first case the fuel saving information will be left out in order to fulfill only the orbit achievement while in the second case the distance to a population center will be requested as input and the algorithm will avoid it guaranteeing a mission profile in respect of safety and reliability.
The realization of this process was possible first of all by investigating the various optimization methods and selecting among them the particle swarm optimization. The algorithm provides, compared to the more classic gradient descent based, an effective method to alter the conditions of the problem in view of an operational failure and at the same time its use is exclusive since it is usually used in problems of multiple subsystem design optimization.
The background of the tools used is expanded through the use of a neural network that will aim to predict, given an operational failure, whether the studied launcher will be able to continue to orbit or abort. Also this solution gives the tool a computational cost efficiency since, through deep learning, the part dedicated to the propagation of the launcher will be bypassed.
The final result is a tool that, given the nominal characteristics of a micro-launcher, allows to study, with respect to a nominal mission in which a target orbit is reached, the branches that this could undergo depending on operational failures that limit the range of the thrust angle or the throttle. This first strength that, if compared to the state of the art of commercial software, is a novelty goes to add up with

a computational cost reduced to a few minutes (about 2) for the calculation of a trajectory compared to methods based on gradient descent (about 8 minutes). The conclusion of this work led to the comparison of the results produced by IRATO with those of ASTOS to give the work more reliability. The work set up in this way is scalable to any class of launcher, from micro to super-heavy with the caveat of training the neural network each time a new launcher is inserted. This procedure requires a database that the more consistent it will be, the more reliable it will be, for this reason a production of about 1000 scenarios (24 hours) can be considered sufficient. The work produced opens avenues for future implementations to the work presented here that will be shown below:

- The implementation of a rigid body dynamics could bring to the dynamic analysis performed on the launcher relevant information also the attitude of the considered vehicle adding new constraints to the considered problem.
- A study conducted through in-company access could provide more reliable information about the failures characterizing the class of micro-launchers by identifying the most likely failures and the organs involved. This type of study and the methods that characterize the study of the reliability of a system are discussed in theoretical terms in Appendix A.
- The tool could be rewritten as an add-on for commercial software using a python interface. Software such as STK allows this type of implementation.

Appendix A

Reliability Analysis

[10] This part will deal with the connections between the assumptions made for the creation of IRATO and the actual effects that affect the trajectory of an Electron-like launcher. The aspects on which this type of analysis will be based are basically two: the first will be to find a link between the operational failures that may occur during the launch and the values with which it was decided to define it. The second one will see a comparison between the trajectories computed by the tool and those computed by the ASTOS program that, being naturally more precise, will show us if what has been produced can be assumed to be correct. Following chapters are dedicated to the theoretical discussion of everything related to safety analysis and the methodologies used for risk prediction. This will be followed by more emphasis on risk analysis methods and then move on to application in our study.

A.1 Sistem Safety

In this part the study will be focused on the phase of the mission when, from an ascent trajectory, due to abortion issues or because a RLV is being analyzed, it is pointing towards the Earth's surface. This type of study is part of systems engineering and involves methods, techniques, and analysis that involve the object of study throughout its life cycle. A system safety process involve the following items:

- Identification of safety-critical events.
- Subsystem and system analysis to identify failure modes.
- Validation and verification throught pratical methods.

The system safety process provides inputs which are useful to expect failure modes, choose among different conceptual design with safety-driven methodology and define operational test. After this step, the analyst can go on with a study based on

subsystem hazard analysis by using the probability data on the interested failure. This technique is useful to involve by the combinations of failures deriving from environment, software and human error. Some practical methods for obtaining

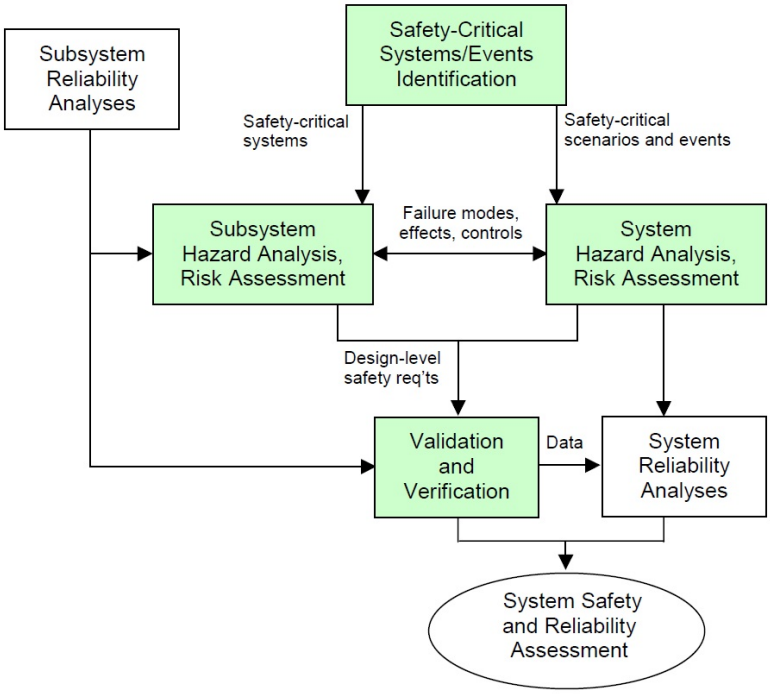

Figure A.1. System safety process

useful data for risk analysis and safety assessment are provided by methods such as Failure Modes, Effects and Criticality Analysis (FMECA) and Failure Tree Analysis (FTA). These methods, starting with design analyses, highlight critical issues in the system or subsystem that could lead to a dysfunction of the affected vehicle. From a practical point of view this leads to the generation of schemes that must be followed by flight operators in order to avoid certain types of choices that could lead to the malfunctioning of some subsystems or parts of it such as the cockpit producing catastrophic effects on populated areas. It is usually rare for a compromised element to generate a failure. In fact, what is considered is the fallout that the damage of a component has on its children. The purpose of the above mentioned techniques is precisely to understand the link between the damage of the component and the external and internal environment involving mechanical failure, software, human error and the system design. Going to identify what the regulations require within a reliability analysis document makes it easier to break down the work that should be done sequentially. This guidelines are provided by Office of Commercial Space Transportation (AST) in collaboration with National Aeronautics and Space Administration (NASA) and Institute of Electrical and Electronics

Engineers (IEEE).

1. **Item identification and description:** When one intends to proceed with a reliability analysis, what must be specified as the first element is the item with which we are dealing. This, as well as being understood as a system, must also be highlighted for the subsystems that make it up. Each product must be described in terms of its function and the level of performance it can provide. If hardware and software are present, they should be described in detail, including the relationships between them. In addition, there will be additional details from environments such as the human factor, which will include interfaces and operating conditions.

2. **Intended Use of the Reliability Analysis Results:** The use for which a reliability analysis is performed should be expressed. For example, one reason why this type of analysis could be useful in this thesis is to prevent the launcher from crashing in a populated area. It is important to specify which uses the outputs of the thesis should be used for and which should not.

3. **Analysis Methods:** The description of the methods used must include a summary of the methodology by which they are to be applied. It is important to specify the assumptions that were made within the project and the software used to implement the project.

4. **Analysis Inputs:** In this part the difference between the value of the data collected when the aircraft is in nominal phase compared to the data collected in that particular situation must be highlighted. This allows the study to be approached from a theoretical perspective.

5. **Analysis Results:** The report form should include in this section the findings, theoretical conclusions, and recommendations from the presenter. Figures of merit are used to express the magnitudes derived from the study in their entirety (e.g., by defining their range of existence). Tipically outputs expected are:

    - Failure modes and effects
    - Criticality of failure modes
    - Single-point failure modes
    - Areas requiring redundancy
    - Functions that cannot be tested
    - Mitigations to minimize or eliminate risk, and combinations of events that could lead to system failure.

A.2 Reliability analysis methods

This section describes the various reliability analysis methods, what kind of inputs and outputs they manage to generate, and how the results can be interpreted. Nomenclature of the quantities involved in the various methods will be introduced as these are encountered.

A.2.1 Reliability Block Diagrams

The Reliability block diagrams (RBD) technique allows you to interpret my system and the interconnections of its subsystems as blocks linked by elements of logic. This technique is useful for evaluating and confirming the reliability of my system in different configurations. The logic flow that must be followed is as follows requires the division of the system into blocks. Then, a certain reliability is assigned to the relevant block so that the global values of different system configurations can be tested. The simplest configurations involve arranging these blocks in series or parallel, or sometimes both. Using the electrical analogy, we are able to determine the overall reliability of the system according to Table A.1.

Type	System reliability (R_s)
Series	$R_s = R_A\ R_B$
Parallel	$R_s = 1$ - $(1\text{-}R_A)(1\text{-}R_B)$

Table A.1. RBD basic layouts

As a pratical example, a scheme which involves main and the redundant subsytems working together is an active parallel one. On the other hand, a system which has an alternate usage of these ones (stand-by redundancy) turns on the redundant system only when the main one is comprimised. This process is in fact called "switching" and, since it can fail, the linked realiability will need an additional study. The advantage of using this technique is the conceptual simplicity with which the system and the connections between its components are configured. However the fact to trace a good subdivision, above all as far as very complex systems are concerned, can be very expensive in terms of work and not all the systems are predisposed to this type of working. In addition, some parts such as the human variable and software components are particularly difficult to model. For these reasons, a study was conducted, qualitative rather than quantitative in nature, using other types of techniques that will be analyzed later.

A.2.2 Parts Count Analysis

Parts Count Analysis (PCA) is able to assess the reliability of a system when it is in the preliminary stage of its development and when therefore the number of parts is fixed. This technique allows at the same time to manage in a weighted way the risk that a certain failure will occur taking into account also elements such as the environment and the human factor. With a required amount of data, PCA is a tool that analyzes the electronic and mechanical components of my system in relative detail. Even if they are not, it is assumed that the components of my system are in series. It is shown in Figure A.2 the logical flow that this procedure requires to be applied. The generic PCA expression is as follows:

$$\lambda_s = \sum_{i=1}^{z} N_i(\lambda_G \pi_Q)_i \tag{A.1}$$

Where:

- λ_s is the system rate failure
- λ_{Gi} is the average failure rate of the i^{th} component
- πQi is the quality adjustment factor relative to the i^{th} component
- N_i is the quantity of the i^{th} component
- z is the number of the different part categories

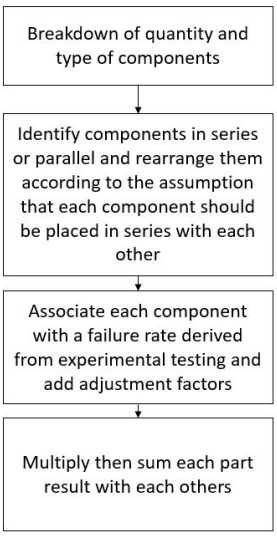

Figure A.2. System safety process

The major disadvantage of this procedure is that not all components can be interpreted according to the PCA declination. However, this tool allows to highlight areas of my system that could potentially create a failure.

A.2.3 Failure Modes, Effects, and Criticality Analysis

The FMECA analysis is a qualitative process based on a bottom-up approach analysis of the system architecture, the starting point is a class of failure scenarios. Throught this study the analyst is able to identify the various failures modes basing them on a gravity hierarchy and probability of occurance. This tool is useful in predicting the most critical areas of the system under consideration and highlights the points where failure is most likely to occur. The scheme that is adopted in applying this type of technique is as follows:

1. In the first phase is defined the main system, the subsystems that compose it and the active and passive interfaces between them. It is also necessary to highlight the various phases that constitute the mission performed by the main body.

2. A breakdown structure of the components and parts of my system is defined according to Figure A.3.

3. Associate to every part of my structure an identification coding.

4. It is necessary to associate a description of the type of failure that may occur to the relevant block both in terms of frequency and severity according to the two tables below.

5. It is necessary to associate a description of the type of failure that may occur to the related block both in terms of frequency (from unlikely to frequent) and in terms of severity (from neglible to catastrophic).

6. For each analyzed component:

   - Identify how the component may generate a failure.
   - Identify the consequences and the people and objects affected by this type of failure.
   - Identify the worst-case scenario.
   - Use the risk acceptability matrix to classify the risk class in order to decide either to mitigate or to move forward the failure.
   - Reiterate with the new measures. also be specified.

7. Document the analysis throught a FMECA worksheet (Example in Figure A.4).

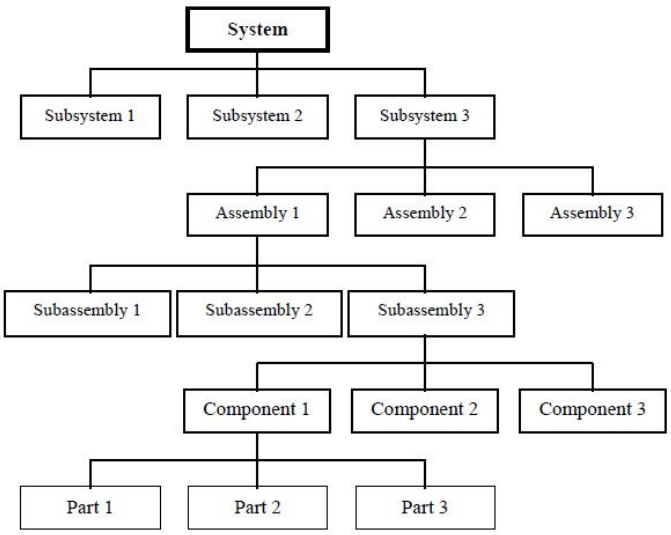

Figure A.3. FMECA breakdown structure

FAILURE MODES, EFFECTS, AND CRITICALITY ANALYSIS WORKSHEET

System: Upper Stage Propulsion System — Sheet 1 of 20

Mission: Satellite Delivery to GEO — Prepared by: John Smith

Phase: Orbital Insertion — Reviewed by: Janet Jones

Ref. Drawing: GTYD-1002B008 — Approved by: Sharon Jackson

Date: January 2, 2004

ID	Item	Failure Modes	Failure Causes	Failure Effects	Risk Assessment Sev.	Prob.	Risk	Detection Methods and Controls
2.0	Combustion Chamber	a. Coolant loss b. Seal failure	a. Manufact. process problem b. Cyclic fatigue	a. Reduced performance, burn-through, possible crash and injury to involved public b. Reduced performance	a.II b.III	a.C b.D	a.6 b.14	a. Inspect welds b. Seal redundancy

Figure A.4. Example of FMECA worksheet

Thanks to the military standard 1629A, FMECA is a technique that also produces quantitative results by adopting the following convention, system reliability is expressed as:

$$C_m = \alpha\beta\lambda_p t \tag{A.2}$$

$$C_r = \sum_{n=1}^{j} (C_m)_n \tag{A.3}$$

$$C_s = \sum_{i=1}^{k} (C_r)_i \tag{A.4}$$

Where:

- C_m is the critical number for a failre mode.
- C_r is the critical number for a failre item.
- C_s is the critical number for a failre system.
- α is the probability of failure due to the selected mode.
- β is the probability of the mission loss.
- λ is the part failure rate.
- t is the operational time.
- j is the part failure modes.
- k is the number of the parts.

The advantages involving the use of this technique allow to highlight the most critical points of the single component and to have a wider view on the reliability analysis resulting from the use of other techniques. However, it does not take into account what will be the effect over time of the failure considered (as we will see for future techniques) and also elements such as the external environment and/or the human factor require the use of other techniques.

A.2.4 Fault tree analysis

In contrary to FMECA, FTA is a technique with an up-down approach that allows to trace back, through the use of logic gates, the generation of a failure. The result of this chain of events takes the name of faul tree. It succeeds therefore to render clear the hierarchy of processes that carry to the generation of an undesired event. Typically, the logic gates that are used refer to the minimal use of "AND" or "OR" constructs and their combinations. The logical structure with which an FTA must be executed is then reported.

1. It is necessary to individuate the pivotal event, that is the undesired event that is wanted to be avoided in order to reconstruct the logical procedure of the actions that involve its occurrence. This type of element can derive from reliability studies such as FMECA.

2. Define the level of accuracy of the analysis by specifying whether the results produced should be qualitative, quantitative or both.
3. Define the chain of events from the high-level actions that lead to the creation of the failure.
4. Going down a level by analyzing the internal causes of the previous point.
5. Connecting second level contributions with first level contributions.
6. Repeat the previous processes until you are satisfied with the accuracy of the work.
7. Document process inputs and outputs.

A scheme of a possible operational failure involving the feed system of the launcher engine is reported as follows in Figure A.5 The parameters used to calculate indi-

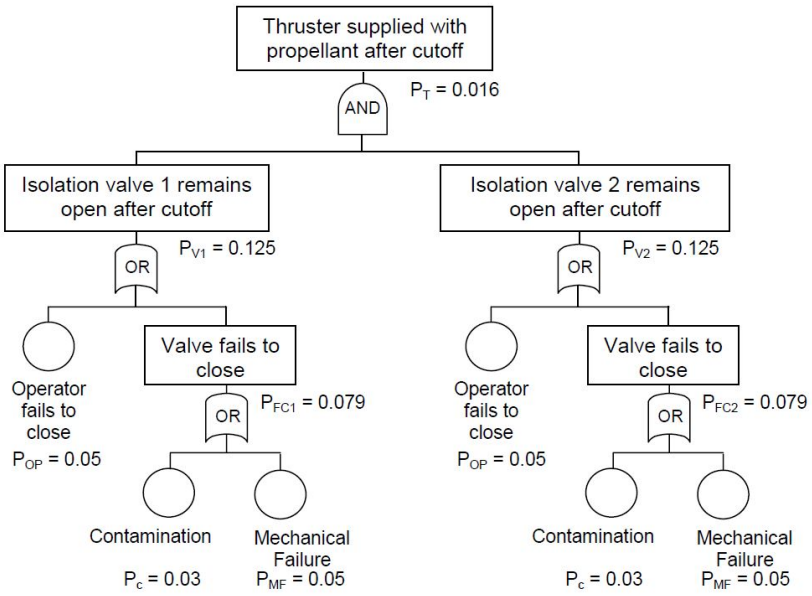

Figure A.5. Example of FTA scheme

vidual node data are difficult to find. Their inspection requires the use of advanced techniques, and the data is sometimes kept within the business circuit that performs the testing. This type of difficulty is addressed through the use of statistical analysis such as Monte Carlo simulations. In order to better understand how the single node influences the behavior of my logical construct sensitivity anlysis are conducted . If during the investigation the reader is looking for a type of analysis that allows risk assessment, to identify couplings and relationships between component operational errors and single-point failures, FTA is a consistent technique.

However, it must be remembered that finding the data necessary to use it is not easy and that to get a complete overview, FMECA or Event Tree Analysis (ETA) must be used.

A.2.5 Event Tree Analysis

Event Tree Analysis is a technique used to permorm inductive investigations on the reliability of the interested system. This kind of analysis includes either hardware or non-hardware components. The gravity of the event considered is not relevant, the only necessary action is to propagate the interested paths and combine the branches with other attempts results in order to notice any success or failure. One positive aspect is the fact that it can be tested combinations where some components may be on standby while others are not. By analyzing the results of the hazard analysis, it is possible to identify the "Success/Failure" ramifications starting from an "Initiating Event" of which one is interested in its propagation. The transition from one event to the next is dictated by engineering links (such as the activation of valves or sensors) that consequently bind a system to the environment in which it is inserted. Each branch can have one or two outputs depending on the event, once this process is complete it is useful to remove redundant information to have a clear structure. By assigning a probability that the branch under consideration tends towards success or failure, it will be possible to calculate which scenarios are more likely to occur than others, reporting everything on an appropriate documentation. In the example shown in A.6 an application case of a scenario of interest is shown.

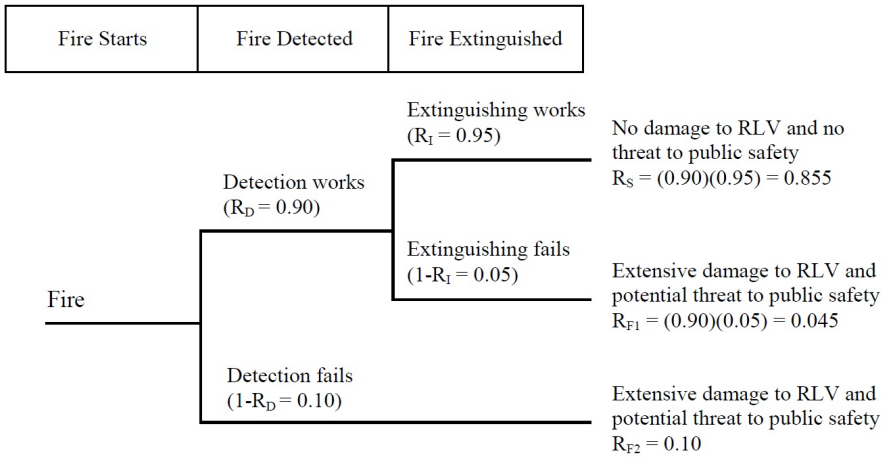

Figure A.6. Example of ETA scheme

Although this technique has some good points in its favor, it must be taken into account that the detailed analysis of a complex system requires many attempts

in order to identify as many scenarios as possible. Moreover, it remains strictly dependent on the results of techniques such as FTA or FMECA both in terms of statistics and providing new scenarios.
After this review it is natural to understand how a reliability analysis is necessary to want to better categorize the class of operational failures that could compromise my mission and the probability with which these could occur. Despite this, the availability of data is not easy for two reasons: the first is inherent to the large amount of combinations and data required by the techniques mentioned above, while the second is related to the availability of the first that is kept limited within the company environment. Implementation of the procedures resulting from these techniques would make for an even more accurate and simulatively reliable lens for IRATO.

Bibliography

[1] Avanzini G., *Entry, Descent, Landing and Ascent*, Politecnico di Torino, V Edition 2009

[2] Castellini F., *Multidisciplinary Design Optimization For Expendable Launch Vehicles*, 2012

[3] Dy D., Perrot Y., Pradal R., *Micro-launchers: what is the market?*, 2017

[4] Curry A., Haskell B., *The Method of Steepest Descent for Non-linear Minimization Problems*, Quart. Appl. Math. 2

[5] Kennedy J., Eberhart R.*Particle Swarm Optimization*, Neural Networks, 1995. Proceedings., IEEE International Conference

[6] Pontani M., Conway B.*Particle Swarm Optimization Applied to Space Trajectories*, Journal of Guidance, Control and Dynamics, 2010

[7] Pontani M., *Particle Swarm Optimization of Ascent Trajectories of Multistage Launch Vehicles*, Acta Astronautica, 2014.

[8] Olds J.R, Ledsinger L.A, *Multidisciplinary Optimization Techniques for Branching Trajectories*, AIAA, 1998

[9] ASTOS Solution, *Conventional Launcher Tutorial*

[10] FAA, *Guide to Reusable Launch and Reentry Vehicle Reliability Analysis*, Federal Aviation Administration, 2005.

[11] Ascent Optimization V_1 and V_2 are MatLAB tools developed by Jasmine Rimani.

[12] Electron Data

[13] Simulink Rocket Dynamic Model

[14] Gradient descent additionary notes

[15] PSO MatLAB Architecture

[16] Neural Network Architecture

[17] Artificial Intelligence Introduction

[18] fmincon Description

[19] POST II website