ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Department of Industrial Engineering

Second Cycle Degree in

AEROSPACE ENGINEERING

Development of Mission Analysis Toolkits

Dissertation in ING-IND/05:
Spacecraft subsystem and space mission design

Supervisor:
Dr. Paolo Tortora

Presented by:
Niccolò Carioli

Co-Supervisor:
Ing. Sergio Miguel Quiles
Ing. Biagio D'Andrea

Graduation session: February 12, 2025

Academic Year 2023/2024

“To confine our attention to terrestrial matters
would be to limit the human spirit.”
Stephen Hawking

This work was done at DEIMOS during an intership programme. The publication of this document is hereby authorised by DEIMOS.

Acknowledgements

I would like to express my sincere gratitude to Dr. Paolo Tortora for creating a bridge between myself and Deimos, allowing me to collaborate with such a prestigious company.
I am also deeply grateful to the entire EOMA team at Deimos for the trust they placed in me. A special thanks goes to Ale, Biagio, and Manuel for their continuous support throughout the entire internship. I would also like to thank Javier for the weirdest yet funniest coffee breaks ever.
Many thanks to all my friends in Madrid: Luke, Manu, Ricky, Rebe, and Mauri, for making me feel at home. These months flew by thanks to you.
I would like to sincerely thank all my university colleagues, the infamous "first row," for making these past two years the best of my university experience. In particular, I am grateful to Mich and Vale; despite all the neurons lost because of you, I will never forget our breaks, study sessions, and all the moments we shared.
A big thanks to my parents, I can feel your support no matter where I am. I will keep striving to make you proud. A special thought goes to my grandparents, whom I know are watching over me and will be with me on my graduation day.
A very special thanks goes to my partner, Greta. Thank you for your perfectly timed words, your invaluable advices in a world that was new to me, and for always being by my side. You make me so happy that I feel I can overcome anything in life.
Last but not least, I want to thank myself for pushing my limits and achieving what I deserve.
Never stop dreaming.

Abstract

This thesis presents the development of a mission analysis toolkit for simulating controlled re-entry of spacecraft. The work was conducted during an internship at Deimos Space and aims to improve and expand an existing software solution. The developed tool, implemented in Python, provides a comprehensive framework for simulating sequences of perigee-lowering maneuvers and incorporates new functionalities, including orbit propagation and timeline generation. The software has been tested extensively to ensure accuracy and reliability.

A key focus of the thesis is the computation of the Δ V needed and the optimization of maneuver strategies, adhering to ESA guidelines for controlled re-entry. The tool allows for detailed scenario analysis, addressing complex mission constraints, such as casualty risk mitigation and precise landing area targeting. In addition to serving as a report on the work performed, this thesis also acts as a user manual, offering comprehensive guidelines for the proper use of the developed software within Deimos Space. The developed toolkit remains available for future mission analyses at Deimos Space.

By enhancing operational capabilities and improving the simulation of controlled re-entry scenarios, this thesis contributes to sustainable space operations and the reduction of space debris in low Earth orbit.

Contents

List of Figures

List of Tables

List of Acronyms

ΔV Delta-Velocity.

desEO Design Engineering Suite for Earth Observation.

DLA Designated Landing Area.

DMS Deimos.

DRA Declared Re-Entry Area.

DRAMA Debris Risk Assessment and Mitigation Analysis.

ECEF Earth-Centered Earth-Fixed.

ECI Earth Centered Inertial.

EIP Earth Interface Point.

EO Earth Observation.

EOL End-Of-Life.

EOMA Earth Observation Mission Analysis.

ESA European Space Agency.

GEO Geostationary Earth Orbit.

GNSS Global Navigation Satellite Systems.

GUI Grafical User Interface.

LEO Low Earth Orbit.

MEE2000 Mean Earth Equator at J2000.

MoD Mean Of Date.

RSO Resident Space Object.

S/C Spacecraft.

SPOUA South Pacific Oceanic Unhabited Area.

SRA Safety Re-Entry Area.

SSO Sun Synchronous Orbit.

ToD True Of Date.

1 | Introduction

Our planet is surrounded by Spacecraft (S/C) carrying out important work to study our changing climate, deliver global communication and navigation services and help us answer scientific questions. In recent years the number of Resident Space Object (RSO) has grown exponentially. While this is a positive symptom of scientific and technological improvement, it also endangers our future capability to access the space environment. Earth's orbital environment is a finite resource and it is rapidly becoming overcrowded. We are moving to the non-return point predicted by Kessler, the in-famous Kessler syndrome after which a-cascade collisions will be inevitable, turning Earth's space environment into a graveyard of deadly, fast moving pieces of defunct S/Cs and rockets. It is thus fundamental to manage the formation of new debris, by adopting strategies to better handle the disposal of existing S/Cs and by developing rules and guidelines to keep a cleaner environment.

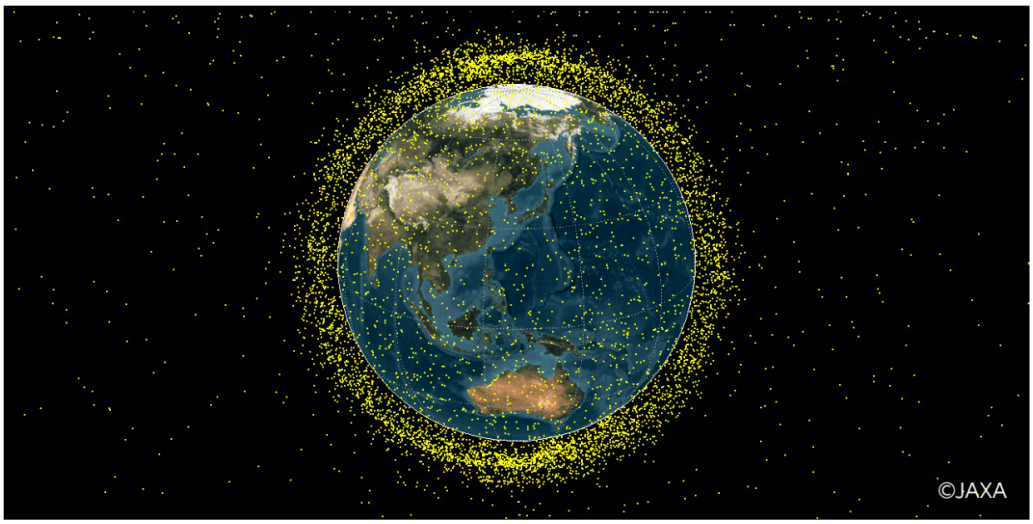

Figure 1.1: Space debris around Earth [1].

1.1. Space debris situation

Computer simulations have shown that the orbital debris population already present on orbit is self-propagating, that is, the orbital debris density will continue to increase through random collisions alone, unless reduced by outside efforts [7]. This may well result in a cascade effect that eventually renders some orbits impractical for space operations. The rapid growth of space activities over recent decades, shown in Figure 1.2, has led to an increasing awareness of the challenges posed by orbital debris. Defunct S/Cs, rocket stages, and fragments from collisions threaten operational S/Cs and future space missions. The awareness of the risk of the accumulation of man-made objects became significant in the late 1970s, and since then, a number of space debris mitigation guidelines have been published by various organizations. As a result, the space community has emphasized the importance of sustainable practices, including End-Of-Life (EOL) management for S/Cs. This process is critical not only for compliance with international space debris mitigation guidelines such as[3, 8], but also for the long-term sustainability of space activities. Since it is not yet economically practical to remove a significant amount of existing debris from their orbit, it is critical for current and future missions to practice a responsible EOL, thus potentially mitigating the rate of increase of mission-lethal debris objects in commonly used orbits.

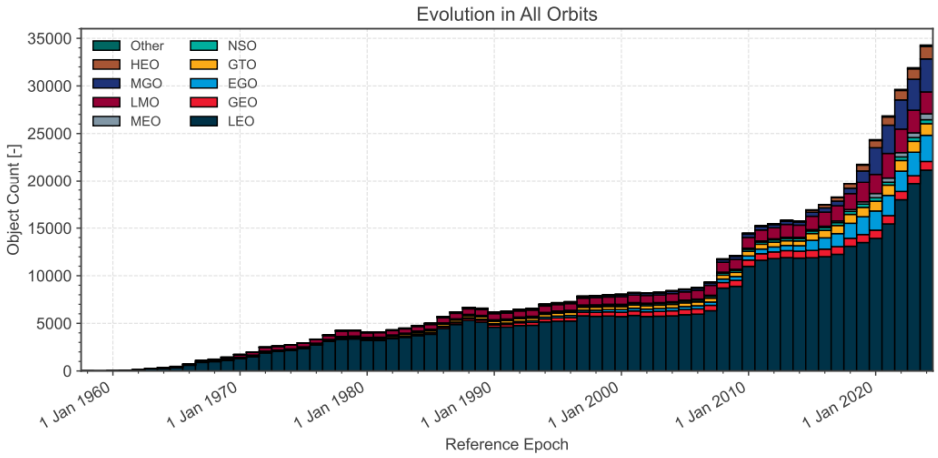

Figure 1.2: Evolution of number of RSO [2].

In particular, [8] defines two protected regions, where the guidelines must be strictly followed. The regions are shown in Figure 1.3 and are defined as follows:

- LEO protected region: spherical region that extends from the Earth's surface up to an altitude of 2,000 km.

- Geostationary Earth Orbit (GEO) protected region: a segment of the spherical shell defined by:
  - lower altitude: geostationary altitude minus 200 km
  - upper altitude: geostationary altitude plus 200 km
  - latitude: between -15 and +15 degrees

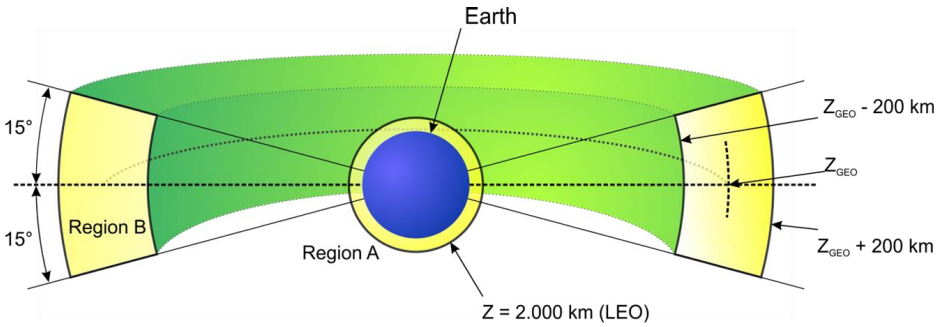

Figure 1.3: LEO and GEO protected regions. [3]

1.2. EOL disposal

A significant amount of effort is dedicated to planning and executing successful mission operations. However, it is equally important to consider the end-of-mission phase during the planning, design and operational stages of any space mission. EOL refers to the processes and strategies implemented to ensure the safe and controlled disposal of a S/C once its operational life has concluded. According to international guidelines [3] a S/C must limit its post-mission presence in the LEO protected region to a maximum of 25 years after the end of its mission and to a maximum of 100 year in the GEO. In this work we will focus mostly on the guidelines for the LEO protected region. Figure 1.4 shows the importance of implementing the 25-year deorbiting strategy. Even tough its application is crucial, is not yet sufficient in mitigating the accumulation of space debris. Due to this reason, new guidelines such as [8], lowered the deorbiting time in LEO protected region to a maximum 5 years, following the so called “Zero Debris” European Space Agency (ESA) policy. In addition to that, other improvements where made, particular to the system passivization and to the collision avoidance maneuvers.

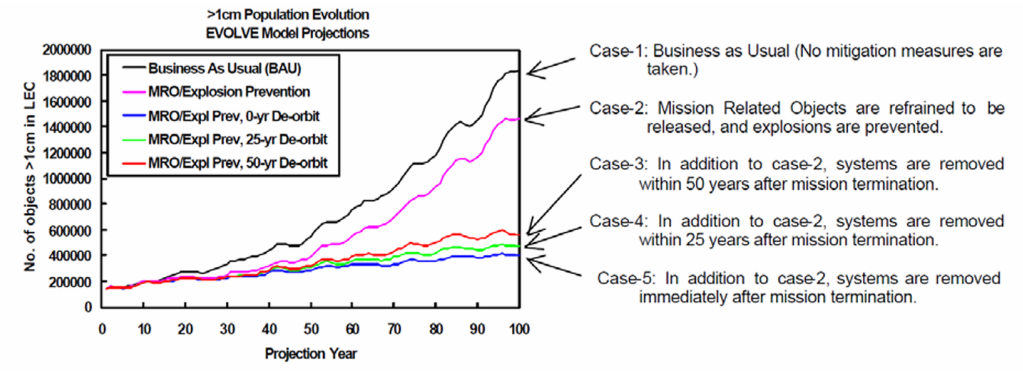

Figure 1.4: Debris evolution using the 25 years rule [4].

When dealing with the re-entry of a S/C, a fundamental figure of merit to consider is the casualty risk. The casualty risk is defined as the statistical probability that one or more fragments surviving the atmospheric re-entry of a space object, such as a S/C or rocket stage, may cause injuries, fatalities, or property damage to people or infrastructure on the Earth's surface. The casualty risk is a key parameter for assessing the safety of S/C re-entries and is regulated by international standards [9], which typically set an acceptable risk threshold around 10^{-4} (1 in 10,000 chance). So, the re-entry must be fast and safe. The general aim of the guidelines [3, 8, 9] is to reduce the growth of space debris by ensuring that space systems are designed, operated and disposed of in a manner that prevents them from generating debris throughout their orbital lifetime and assures sustainable space utilization. A part of the re-entry analysis must be accounted in the ΔV budget, as explained in section 1.3. There are several strategies for managing the EOL of a S/C, depending on its orbit, mass, and mission parameters. The main strategies for the disposal of a S/C, as explained by [8] are:

1. Controlled re-entry: a faster and safer re-entry. It allows for the re-entry in less than a month while controlling the impact point on Earth's surface.

2. Uncontrolled re-entry: a slower re-entry, since it follows the 5 years rule stated in [8], and doesn't allow the control of the impact point on Earth's surface.

These strategies are different by nature, and they will be explained later. For both of them however, the complete re-entry analysis consider a lot of aspects, such as passivization of the systems, management of payloads and others, as explained in [9, 8]. Another fundamental part is the atmospheric simulation, where the fragmentation of the S/C happens. This work will focus on the part related to the ΔV analysis, although some hints to the atmospheric part will be given in section 4.3.

1.2.1. Uncontrolled re-entry

Uncontrolled re-entry refers to the descent of S/C back to Earth without any active control over its trajectory. Debris release limitation requirements intend to reduce the risk of a debris collision occurring by controlling the presence of S/C components in the LEO protected region over a period that exceeds 5 years. If necessary, the disposal strategy of a LEO S/C will then foresee a manoeuvre to lower the orbit perigee to an altitude that guarantees safe uncontrolled decay within 5 years. Usually, the EOL strategy consists in a perigee-lowering manoeuvre, aiming to minimize the required propellant. If needed, the EOL disposal manoeuvre consists in an impulsive tangential manoeuvre (Hohmann transfer), lowering the perigee, or both the perigee and apogee, to the selected altitude. For missions that are planning for an uncontrolled re-entry, but do not comply to the casualty risk requirements by a narrow margin, a set of measures can be applied.

The measures mentioned above can be included in two categories:

- Refinement of S/C modelling: usually following ESA guidelines [10].
- Design-for-demise (D4D): represent different engineering solutions to aid demise (changing the component's material, triggering a partial break-up of the S/C structure during re-entry, ...)

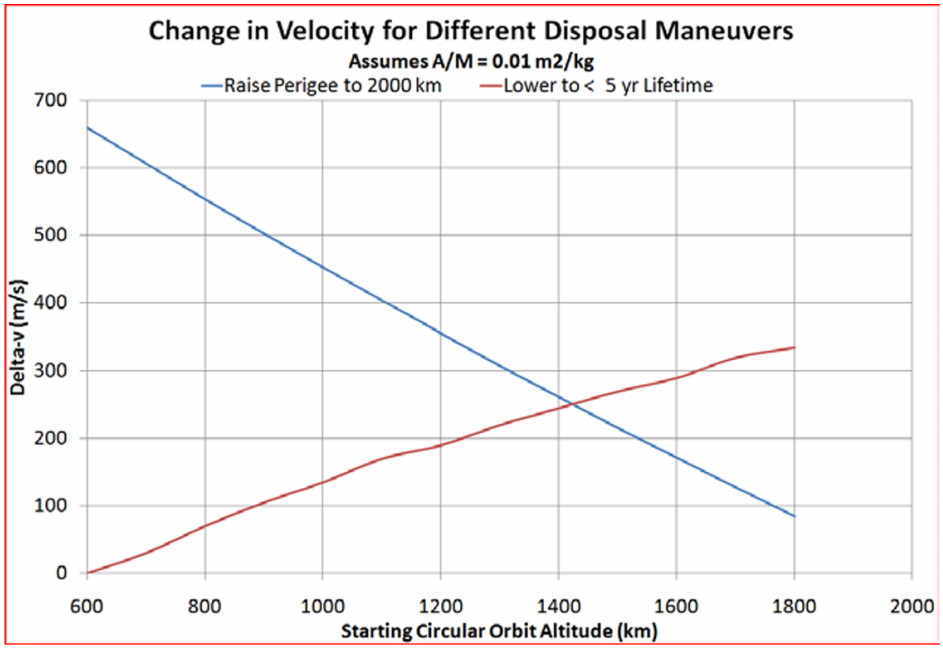

Figure 1.5: ΔV needed for uncontrolled re-entry in LEO [4].

Figure 1.5 illustrates the typical ΔV required to transition from a circular LEO orbit to an orbit that will

re-enter the atmosphere within 5 years. It is important to note that solar flux, which varies significantly over the solar cycle, is a critical factor in predicting re-entry timelines, and it adds complexity to these calculations.

1.2.2. Controlled re-entry

For missions incompatible with an uncontrolled re-entry scenario due to non-compliance with the casualty risk requirements, a controlled re-entry should be performed. For a controlled entry the compliance with the ESA guidelines [9, 8] is ensured by the selection of the timing and entry gate state for the de-orbit manoeuvre to conduct debris fall-out towards non-populated areas (open ocean, e.g. South Pacific Oceanic Unhabited Area (SPOUA)). In practice, controlled re-entry is best performed using at least three separate maneuvers. This is done for mainly two reasons. Firstly, to ensure a limit on the gravity losses per maneuvre and secondly, to better control and refine the orbit, with a final perigee of less than 50 km, to prevent atmospheric skip [6]. To accomplish this, the S/C design must incorporate sufficiently large thrusters to ensure adequate control authority at low altitude. Controlled re-entry also requires the vehicle to save sufficient fuel to reliably perform the reentry maneuvers at the end of the mission, which will result in a larger fuel mass at launch. The evaluation of this is the key point around which this work focuses.

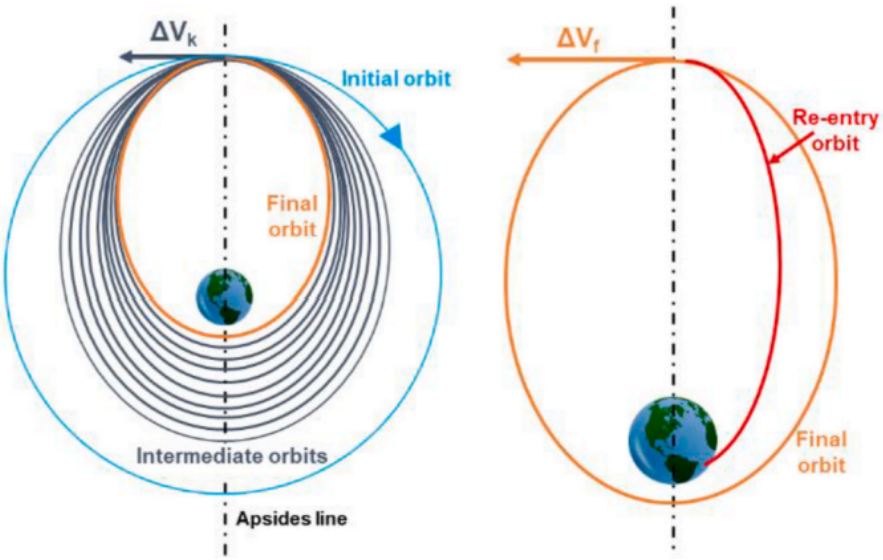

Figure 1.6: Controlled Re Entry strategy [5].

The idea of the controlled re-entry is to perform a sequence of perigee lowering maneuvres following a given scheme, such as the one shown in Figure 1.6. During this phase the maneuvres, as well as the

time between them, can be tuned in order to obtain the desired behavior.

1.3. ΔV budget

The ΔV budget is an estimate of the total change in velocity required for a space mission. Since the change in velocity is strictly related to the propellant consumption, the result of this analysis is not only the distribution of ΔV among the different stages of a mission, but also the plot of propellant consumption, as well as the initial wet mass of the S/C. ESA guidelines [11] describe the steps that must be followed for the computation of the ΔV budget. This is a fundamental concept in space mission design, representing the total change in velocity required to perform all planned maneuvers throughout a S/C's lifetime. This includes a wide range of operations, such as orbit insertion, station-keeping, collision avoidance, deorbiting, and, in some cases, interplanetary trajectory corrections. The ΔV budget is directly tied to the amount of propellant a S/C needs, making it a critical parameter for both mission planning and S/C design. Each maneuver in a mission contributes to the total ΔVbudget. For example:

- Orbit Acquisition: The maneuver to transition from the launcher parking orbit to the intended operational orbit.
- Station-Keeping: Small adjustments to maintain a S/C's position and inclination
- Collision Avoidance: Unplanned maneuvers to mitigate risks of conjunction with other objects in space.
- EOL Operations: ΔV required for deorbiting.

Proper management of the ΔV budget ensures that the S/C can fulfill its mission objectives while maintaining flexibility for contingencies. Factors influencing the ΔV budget include the S/C's mass, propulsion system efficiency, orbital parameters, and mission duration. Additionally, advancements in propulsion technology, such as electric propulsion, have enabled more efficient use of ΔV, allowing S/Cs to achieve longer operational lifetimes and greater mission capabilities.

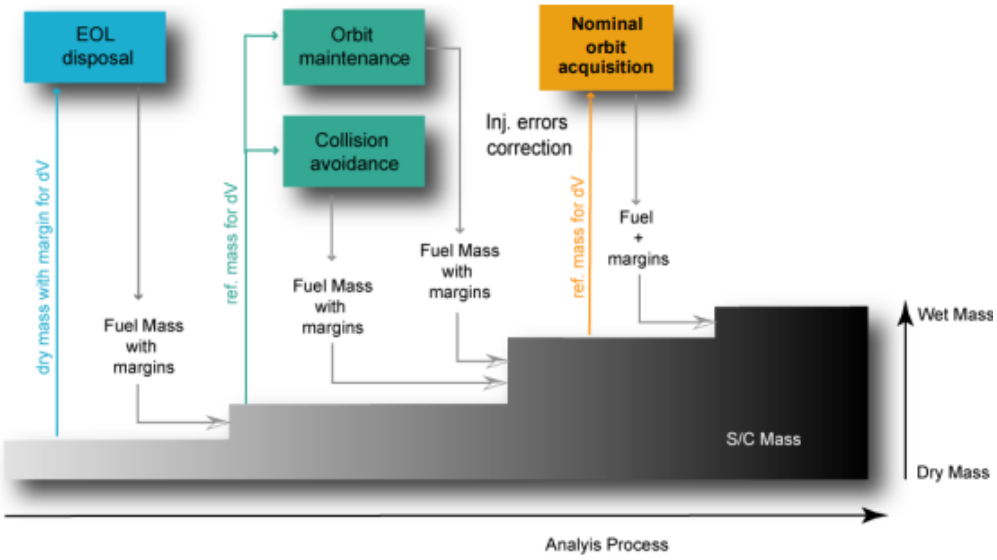

Figure 1.7: Approach for ΔV and fuel budget computation in LEO.

Referring to Figure 1.7, it can be seen that the way in which this analysis is performed is backward, according to [11]. This means that even though the EOL disposal will happen last in the mission timeline, it is actually the first analysis to be performed, since its output wet mass will be the input dry mass for the next analysis. In addition to that, some margins are applied in order to account for uncertainties. In particular, referring to the controlled re-entry, ESA guidelines [11] impose to consider a 15% margin on the last maneuvre. For the other phases margins, one should always refer to [11]. As briefly said in section 1.2, the common idea between the two re-entry strategy is the use of maneuvres to lower the initial orbit, even tough for the uncontrolled re-entry this is not always necessary, but it depends on the initial orbit itself. The computation of the propellant needed for those maneuvre is the part that interest the ΔV budget.

1.4. Scope of the thesis

This work focuses on the development of a toolbox capable of simulating the sequence of maneuvers needed to perform the controlled re-entry providing both an update and an expansion to a previous software. Specifically:

- The existing MATLAB code base has been re-implemented into Python scripting language for the integration with the exiting tool explained in Figure B.

- The code has been expanded to handle more complex scenarios.

.

In particular, the original MATLAB code only allowed for the simulation of the sequence of maneuvres, without accounting for the time elapsed between them and the effect of the perturbation. In fact, it could only be used for early mission analysis phases, in particular from phase 0 to phase B. Furthermore, the code was poorly commented and detached from the main tool environment. The scope of the work of my internship was to transport this software in the correct framework while enhancing its functionalities and output capabilities to handle more complex scenarios. In addition to that, the idea is to make the tool more operational, guaranteeing the possibility to obtain a timeline of the whole re-entry process and making it a viable software for the latest mission analysis phases.
The objective is to have a flexible tool that could:

- Simulate both easy and complex controlled re entry scenarios
- Output the quantities of interest
- Provide a timeline of the mission
- Allow for fast post-processing
- Abilitate the communication between different departements to fullfill the complete re-entry analysis.

The software was then used on real ESA mission of the Copernicus programme, during phase C, and will remain available to Deimos (DMS) for the analysis of more mission, from phase 0 to phase C. This thesis will report both the problem and the solution used to tackle it.

2 | Problem formulation

The aim of the software is to simulate the controlled re-entry of a S/C and compute the required ΔV. The simulation requires a sequence of perigee-lowering maneuvers, which are separated by a given time. Before delving into the modeling approach employed, it is essential to establish the underlying assumptions and to define the required inputs and outputs.

The necessary inputs are shows in Table 2.1 and they will always be considered as known.

Inputs	Description
Dry Mass	Final satellite mass
Initial Orbit	Set of parameters that defines the orbit (keplerian element, repeat cycle, ... see Appendix A)
Propulsive System	Thrust level and specific impulse associated with each maneuver
Propagation Time	Time elapsed between two consecutive maneuvers
Target Perigees	Sequence of perigees to be followed

Table 2.1: Simulation inputs.

It is then assumed that the thrust, F, specific impulse, I_{sp} and the mass flow rate, $\dot{m}$, that is computed as $\dot{m} = \frac{F}{I_{sp} \cdot g_0}$, are all constant within the same maneuvre, even tough they can change among different ones. In addition, some problem simplifications will be described when needed.

The required outputs are various, but the main ones are showed in Table 2.2.

Output	Description
Wet Mass	Initial satellite mass
Parameter Evolution	Evolution of keplerian parameters during the maneuvers and propagation
Maneuver Execution	Maneuver-associated quantities (ΔV, propulsive arc duration, gravity losses, ...)
Mission Timeline	Epoch and state before and after an event on the mission

Table 2.2: Simulation Outputs.

Next, we will talk about each step of the modelization process.

2.1. Maneuvre modellization

First of all, it's important to highlight that the S/C state will be always rapresented using mean keplerian elements expressed in the True Of Date (ToD) frame, defined as illustrated in Appendix A.
According to the guidelines set by ESA [11], the implemented maneuver must be modeled as a continuous process. Specifically, the simulated maneuver is an in-plane, perigee-lowering maneuver. This means the maneuver is performed around the apocenter, with the ΔV vector aligned with the orbital velocity, but in the opposite direction to reduce the semi-major axis. The rationale behind this modeling approach is to approximate a finite maneuver as a sequence of impulsive and ideal maneuvers. To achieve this, the first step is to define the number of ideal maneuvers per second, denoted with f, to be used for the approximation.
Let us assume, for the moment, that the duration of the maneuver T, is known. Also, let's assume that the maneuvre is symmetric with respect to the absidis line, that is the line connecting apocenter and pericenter. Since the maneuvre must be performed around the apocenter, we can easily find the initial mean anomaly at which the maneuvre should start. Let's call T_0 the orbital period and define the mean orbital angular velocity as $n = \sqrt{\frac{\mu}{a^3}}$, then we can use Equation 2.1 to compute the initial mean anomaly M_0.

$$M_0 = n_{ref} \cdot \frac{T_0 - T}{2} \tag{2.1}$$

Next, we can define the total number of infinitesimal maneuvres performed during the propulsive arc as $round(T \cdot f)$, and then discretize it into infinitesimal arcs of duration dT, that can be calculated using Equation 2.2. It has been found out that a good value for f, that achieves the right value between computational velocity and accuracy is $f = 2$, thus each infinitesimal arc duration will last dT=0.5 seconds.

$$dT = \frac{T}{round(T \cdot f)} \tag{2.2}$$

At the beginning of each arc, the state vector of the S/C is given by $\vec{\boldsymbol{x}} = [a, \vec{e}, \omega, M]$, where a is the semi-major axis, $\vec{e}$ the eccentricity vector, ω is the pericenter argument, and M is the mean anomaly. Since this is an in-plane maneuver, the inclination, i, and the right ascension of the ascending node, Ω are not considered, as they remain unaffected.
At the start of each infinitesimal arc, the S/C's position in the orbit is defined by M, whereas a and e define the shape of the orbit, and ω defines the orbit rotation. At this point, an impulsive infinitesimal dV, computed using Equation 2.4, is applied, with m_p expressing the propellant mass used per infinitesimal arc.

$$m_p = \dot{m} \cdot dT \tag{2.3}$$

$$dV = I_{sp} \cdot g_0 \cdot \ln \frac{M}{M - m_p} \tag{2.4}$$

After the application of the dV, that is assumed impulsive, the position vector, $\vec{r}$, remains the same, whereas the velocity modulus is updated according to $V_{new} = V_{old} - dV$, causing a change in the state $\vec{x}$. In fact, we can write $\vec{r}$ and $\vec{V}_{new}$ in the orbital frame; next using Equation 2.5 and Equation 2.6 we can obtain the updated semi-major axis and eccentricity vector.

$$a_{\text{new}} = \frac{-\mu}{V_{\text{new}}^2 - \left(\frac{2\cdot\mu}{r}\right)} \tag{2.5}$$

$$\vec{e}_{new} = \frac{\vec{V}_{\text{new}} \wedge \left(\vec{r} \wedge \vec{V}_{\text{new}}\right)}{\mu} - \frac{\vec{r}}{\|\vec{r}\|} \tag{2.6}$$

The eccentricity vector holds two different information. Its norm defines the new eccentricity of the infinitesimal arc, whereas its direction gives information about the rotation of the new infitesimal arc. Referring to Figure 2.1, where the red quantities are the post dV quantities, it can be seen how the change in eccentricity direction directly affects both ω and the true anomaly Θ. The changes can be computed according to equations Equation 2.7 and Equation 2.8. Obviously, the change in Θ can easily be reported as change in M using the standard angle conversion formulas.

$$\Theta_{new} = \arctan_2\left(\frac{\vec{r}\cdot\vec{e}_{new}}{r\cdot e_{new}}, \frac{\|\vec{r}\wedge\vec{e}_{new}\|}{r\cdot e_{new}}\right) \tag{2.7}$$

$$\Delta\omega = \Theta - \Theta_{new} \tag{2.8}$$

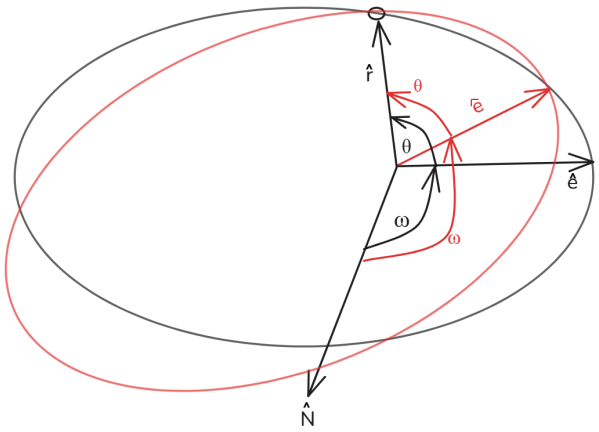

Figure 2.1: Schematization of the effect of eccentricity direction change.

So, after the impulsive dV is applied, the infinitesimal arc is updated according to the changes in the state $\vec{x}$ described above, thus the S/C finds itself in a new updated orbit. In order to account for the gravity losses, the S/C now stays in this updated orbit for a time dT, changing the state, in particular the anomaly and the radius, according to the standard keplerian model. Then, the end conditions of a given arc will be the initial condition for the subsequent arc. At the beginning of it a new impulsive

dV is applied, causing the state, and thus the arc, to vary. Then the satellite spans again that arc according to the keplerian motion. This is repeated until the last infinitesimal arc. Figure 2.2 shows the portion of the orbit where the maneuvre happens that is around the apocenter. Figure 2.3 shows a portion of the propulsive arc discretization. Lastly, Figure 2.4 shows the discontinuity caused by the impulsive infinitesimal dV that changes the orbit. At the end of the upper arc the dV is applied, then the subsequent arc is updated and the spacecraft spans it for a time dT.

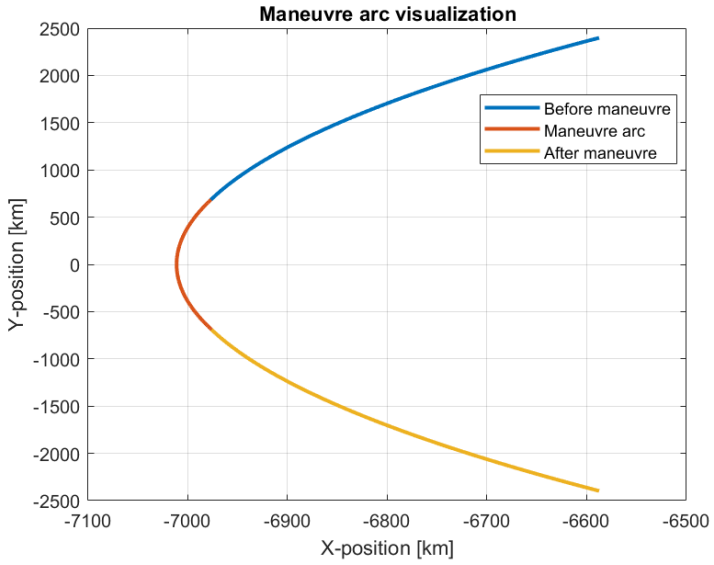

Figure 2.2: The maneuvre arc happens around the apocenter.

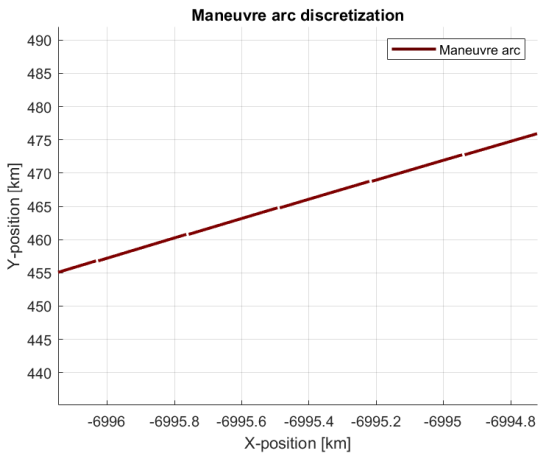

Figure 2.3: Portion of the maneuvre arc discretizion.

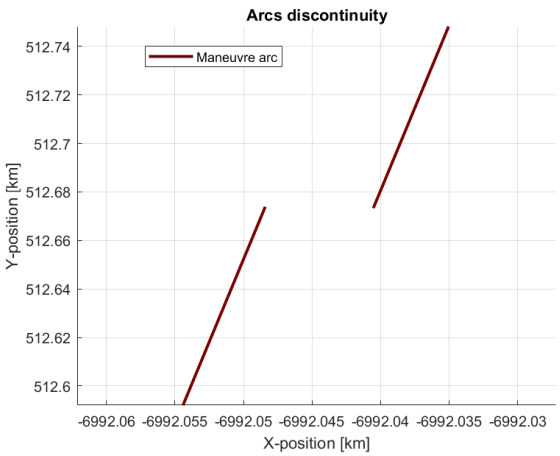

Figure 2.4: Discontinuity between two consecutive arcs.

If we simulate 3 maneuvres with the perigees drop shown in Figure 2.6, the obtained results are shown in Figure 2.5 and the gravity losses are reported in Figure 2.7. These results highlight how the gravity

losses accumulate over time. Thus the bigger the maneuvre is the higher the ΔV will deviate from the ΔV_{ideal}, due to gravity losses.

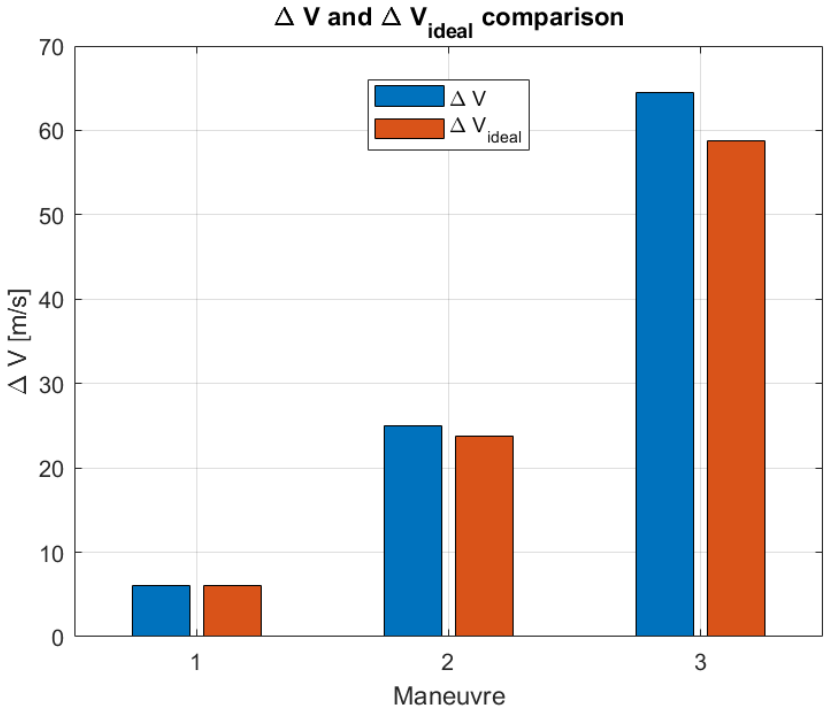

Figure 2.5: ΔV and ΔV_{ideal}.

Index	Perigee drop [km]
Maneuvre 1	21
Maneuvre 2	98
Maneuvre 3	200

Figure 2.6: Perigees sequence.

Man ID	Gravity losses %
1	0.08
2	1.03
3	9.69

Figure 2.7: Gravity losses.

Then, in order to take into consideration the perturbations during the whole propulsive arc, an heuristic approach is used. This method relies on the fact that the orbit is propagated bewtween two subsequent maneuvres and that the propagation is not stopped at the initial maneuvre point, but goes on at least for another orbit. During the propagation the perturbations are taken into consideration, refining the results. During the maneuvre, it is considered that the semi-major axis and the eccentricity are effected only by the ΔV, since the changes caused by it are much higher than the ones given by the

perturbations. For i and Ω instead, only the effect of the perturbation is considered, since they are out of plane quantities and thus not affected by the maneuvre. For ω instead, a superposition is used and the final value is obtained by summing the value at the end of the propulsive arc given by the propagation, that however do not consider the maneuvre, and the sum of all the $\Delta\omega$ during the propulsive arc. Further details will be provided in section 2.3. Obviously, following this approach, we are making a lot of simplifying assumption. In fact a component in the value of ω is obtained trough the propagation on an orbit that is not the real one, and this also hold for i and Ω. However, even with all these assumptions, this method is able to simulate finite manoeuvres with good accuracy, without the need to solve differential equations. The accuracy of the maneuvre simulations have been tested, comparing its results to a DMS internal software. The latter simulates the maneuvre considering both the ΔV and the perturbations inside differential equations, and then solving them at each integration step, guaranteeing a better accuracy. Nonetheless, the comparison shows a percentage relative error below 1 % in all orbital parameters, and other maneuvre associated quantities, ensuring the maneuvre simulation to be good enough for mission analysis purposes.

2.2. Optimal maneuvre duration

The maneuvre simulation explained in section 2.1 relies on the fact that the maneuvre duration is known. Unfortunately, that is not known a priori and only the starting and ending conditions in terms of perigee altitude of the maneuvre are known.

Let's recall that the maneuvre is a perigee lowering one, so the idea is that we fix the next perigee to be reached and we perform a maneuvre to reach it.

This means that the reached perigee altitude is the parameter that we need to control in order to find the optimal maneuvre duration. If the maneuvre is too long, then the perigee altitude would be too small with respect to the one chosen, since we have slowed down too much; conversely, if the duration is too short the perigee would end up being too high, meaning that we haven't slowed down enough.

Thus, in order to find the optimal duration, an iterative method can be set up. A simple and reliable way is to use a bisection method, using as check parameter the perigee altitude reached after the maneuvre.

In order to implement this logic we need first of all to set the boundaries, around which the bisection will occur.

The idea is to use the time needed to perform the maneuvre considering the ΔV_{Ideal}, and then build some boundaries around it. To begin, we need to compute ΔV_{Ideal}. This can be easily done using

standard orbital mechanics formulas. Calling with the pedix in the initial quantities and with the pedix $target$ the final ones, we can compute ΔV_{ideal} using Equation 2.9, where Ra stands for apocenter radius.

$$\Delta V_{ideal} = \sqrt{\mu \cdot (\frac{2}{Ra_{in}} - \frac{1}{a_{in}})} - \sqrt{\mu \cdot (\frac{2}{R_{a_{in}}} - \frac{1}{a_{target}})} \tag{2.9}$$

Inverting Equation 2.4 we can find the amount of propellant needed, following Equation 2.10, where M_0 is the initial mass.

$$M_p = M_0 \cdot [1 - \exp\left(\frac{-\Delta V}{I_{sp} \cdot g_0}\right)] \tag{2.10}$$

Then, knowing the mass flow rate relative to the considered maneuvre, the burnout time can be easily computed as $T_b = \frac{M_p}{\dot{m}}$ Then, the bounds are set as follows:

- lower bound = $0.8 \cdot T_b$.
- upper bound = $3 \cdot T_b$.

Note that, the bounds are not symmetric as it is expected that the real duration to be larger than the ideal one. Once the bounds are set, the bisection can be run, thus the initial duration guess is set as the middle point of this interval and then the maneuvre is simulated according to the procedure described in section 2.1. After the maneuvre is simulated, the reached pericenter is calculated and the pericenter error $\epsilon_p = rp_{target} - rp_{reached}$ is computed. Based on the value of ϵ_p, one half of the initial interval is discarded, the bounds are changed and a new duration guess is used. The procedure stops when $\epsilon_p < 1$ m. Figure 2.8 summarized the iterative process required for the computation of the maneuvre optimal duration.

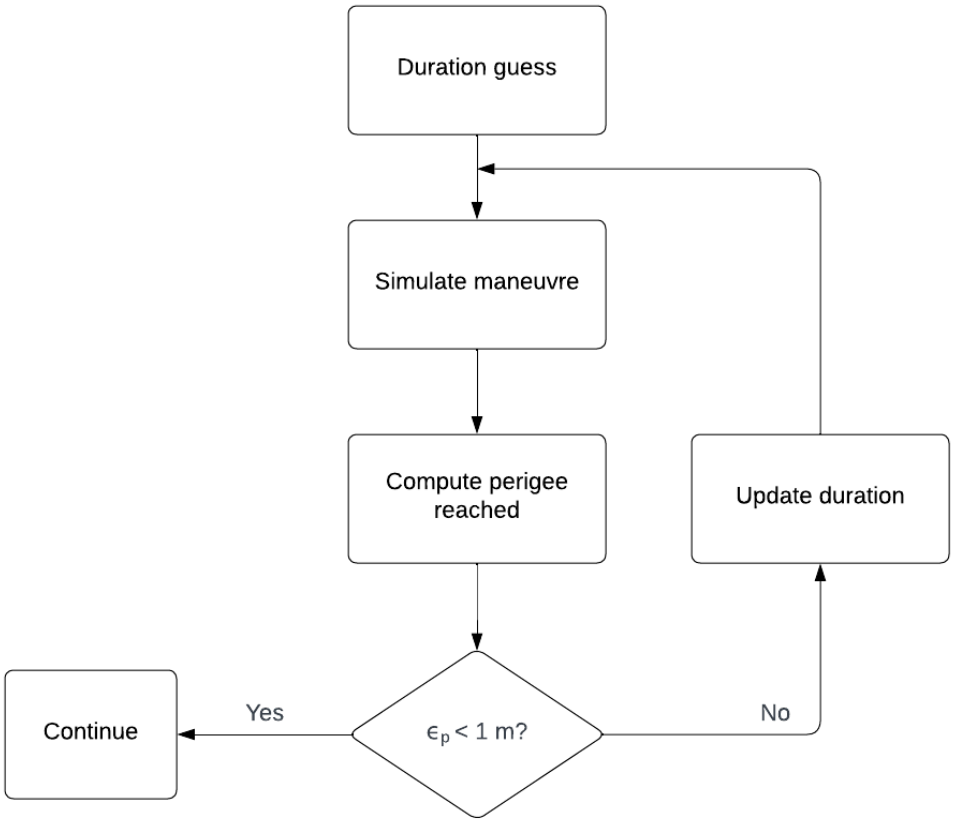

Figure 2.8: Block diagram of optimal duration.

2.3. Orbit propagation

As previously cited, the possibility to consider the time elapsed between two consecutive maneuvres propagating the orbit is a fundamental new feature added to the exiting tool capabilities. The propagation is done trough a DMS internal tool. The tool consists of an high precision numerical integrator that allows for the propagation of the orbit considering gravitational perturbations, as well as third body, drag and solar radiation pressure. This tool is a Design Engineering Suite for Earth Observation (desEO) executable (see Appendix B) and it needs a proper xml input file, where one must specify all the needed informations accepted by the program. In particular, one must select the perturbations to consider, which degree and order of the gravity field to consider, information about the satellite like mass, C_d, initial orbit, epoch, etc... The issue comes with the final point of the propagation. In fact, the propagation starts at the end point of the previous maneuvre, and it ends at the starting point of the following maneuvre, that is not known a priori since the maneuvre duration is found trough the procedure described in section 2.2. Thus, in order to be conservative and to implement the procedure stated

in section 2.1, the initial state considered for the maneuvre simulation is the last apocenter reached. This means that the last apocenter state is used as initial condition for the maneuvre simulation, even though it is not the real initial point of the maneuvre. However, the propagated state after the apocenter is still saved because it will be used to take into consideration the perturbations, as explained in section 2.1. By doing so, we are ignoring the effect of the perturbations between the apocenter and the maneuvre starting point. In reality, we are considering that the perturbations act on the reference orbit also for half of the maneuvre duration as such, the approach is said to be conservative. Even with this approximation, the results are satisfactory for the scope of the tool. To conclude, the real propagation duration never match exactly the input duration, since from that number we need to subtract the amount of time needed to reach the last apocenter.

2.4. Optimal mass

It is defined optimal the initial mass that, after the execution of all the maneuvres, has consumed all the propellant. So, we are looking for the minimum amount of propellant mass that is needed to fullfill all the maneuvre. Each set of maneuvres is performed with a given initial wet mass; however, since the maneuvre duration is not known a priori and since the duration itself depends on the initial mass, as it affects the dV in each arc, the initial wet mass is not known a priori. In addition to that, the propagation can lead to changes in the apocenter and pericenter, causing the maneuvres to change their intensity in a unpredicatble way. Also in this case the problem is not linear, thus an analytical expression for the initial mass doesn't exists; therefore, as it was for the maneuvre duration, a bisection method is used to find the optimal mass using as performance index the residual mass. As we did for the maneuvre duration, we need to set the boundaries. In this case the boundaries are set as follows:

- lower boundary: S/C dry mass.
- upper boundary: problem-knowledge-based. Note that the upper boundary is not so important; if it is too large the code will need more iterations to converge.

As usual, the wet mass guess is taken as the middle point. Then, all the maneuvres are performed, and after each maneuvre the amount of fuel needed is subtracted from the wet mass. After the simulation of all of the maneuvre, the residual mass is computed according to Equation 2.11. It shall be pointed out that for the last maneuvre, the margin explained in section 1.3 is taken into consideration.

$$M_{residual} = M_0 - \sum M_{propellant} - M_{dry} \tag{2.11}$$

This value is then compared with a threshold, called residual fuel, that represents the maximum amount of unusued fuel remaining. The number was set based on guidelines [11], and its value was 10^{-2} Kg. If the residual mass is below this value, then we have found the optimal value for the initial mass. Otherwise, a new initial mass is chosen according to the bisection method. Figure 2.9 summarized the iterative process required for finding the optimal initial mass.

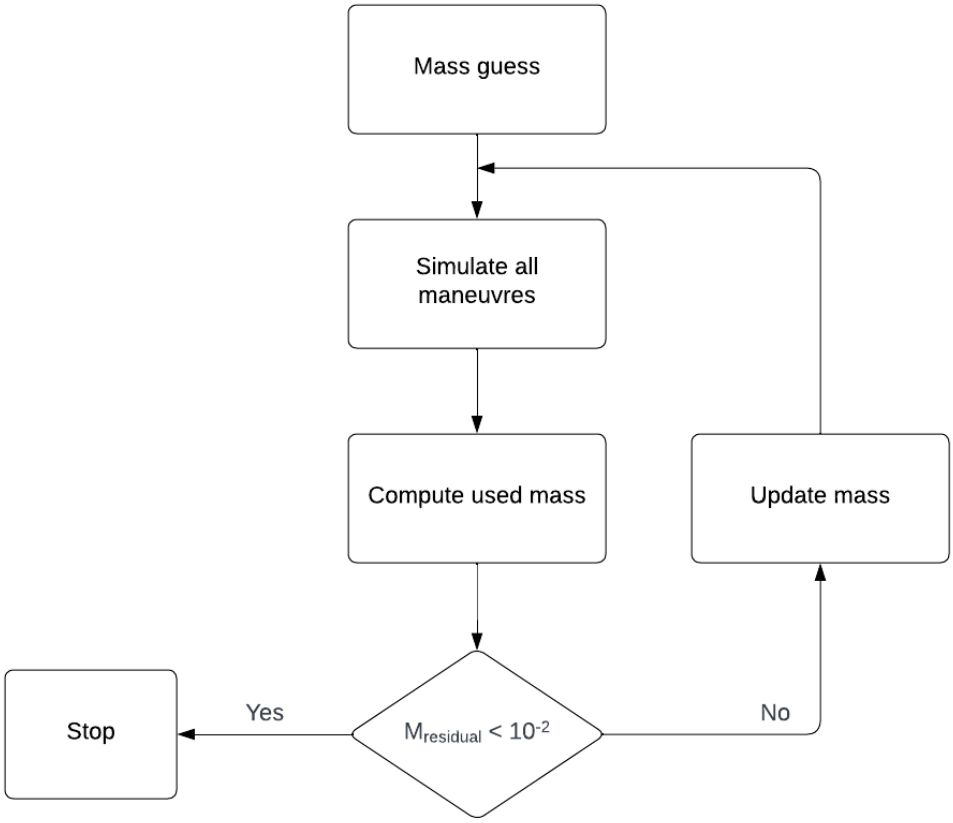

Figure 2.9: Block diagram of optimal mass.

3 | Software implementation

In this chapter the structure and performance of the code will be analyzed. In order to have a software that is in line with the DMS framework, it was developed in Python. To obtain a really flexible code that could easily accepts a lot of inputs, a class was developed, called ControlledReEntry. A UML scheme of the class is shown in Figure 3.1.

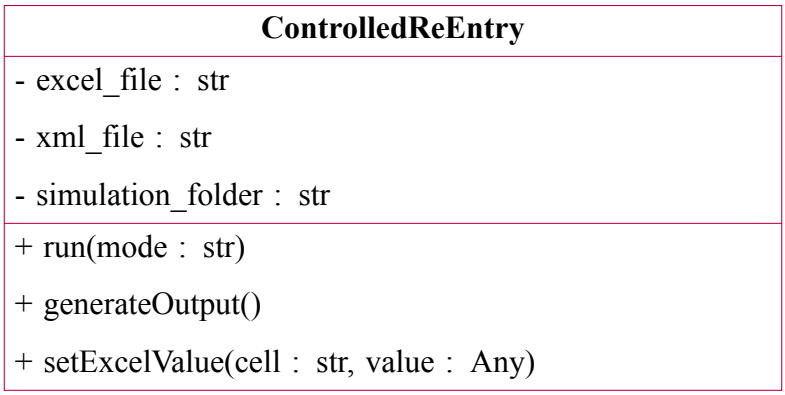

Figure 3.1: UML class diagram of the ControlledReEntry class.

As one may have understood, the code needs a lot of inputs to be run since it requires informations about the propulsive system, the maneuvre sequence, the propagation time as well as the input files needed to run the desEO executables. To facilitate the inputs definition, two files are created: an `Excel`, which contains all the information about the controlled re-entry, and an `XML`, which run the desEO executables. These input files will be better described in section 3.1. They are then fed to the ControlledReEntry class, and once an istance of the class is created, three main methods can be run.

- `run(mode)`: Executes the re-entry simulation.
- `generateOutput()`: Produces output data in a structured format.
- `setExcelValue(cell, value)`: Allows the modification of specific values in the Excel input

file.

All of the methods will be better described in section 3.2. This type of structure guarantees an easy and intuitive interface for the user, allows for either complex or simple analysis, and permit to tune parameters in case of parametric analysis.

3.1. Software input

An instance of the class is generated as follows:

```
excel_file = "name.xlsx"
xml_file = "name.xml"
simulation_folder = "path_to_simulation_folder"
EOL = ControlledReEntry(excel_file, xml_file, simulation_folder)
```

Where:

- `excel_file`: it is the file that defines the maneuvre input.
- `xml_file`: it is the file that setups the desEO executables.
- `simulation_folder`: it is the path to the folder where to store the simulation results. Note that by default this is set to the current directory.

3.1.1. Excel file

The excel file defines the inputs needed for the maneuvre, a snapshot of it can be seen in Figure 3.2.

Dry mass [kg]	Maximum mass [kg]	Thrust [N]	Specific impulse [s]	Number of Thrusters	Number of manoeuvres	Intermediate perigee altitude [km]	Final perigee altitude [km]	Propagation Duration [days]
1600	2000	60	223	1	4	410	50	0
		58	222			340		2
		56	221			270		5
		54	220			250		3

Figure 3.2: Excel input file structure

Note that the structure of this file is fixed. The flexibility is obtained by allowing the user to change as he wants the values, but the structure must be unchanged. In particular, the user can be personalise the inputs as follows:

- **Thrust**, as well as **Specific Impulse**, can be given as scalar or as vectors, and they rapresent the values that will be considered for a specific maneuvre. If given as a scalar, the value will be considered constant for all the maneuvres. If instead vectors are given, at each maneuvre the corresponding value will be considered. Note that for the case of input vector the length must be equal to the number of maneuvres.

- **Intermediate perigee** rapresent the sequence of perigee altitudes, to be followed. It can be given as scalar or as vector. If given as a scalar, one must input the second-to-last perigee altitude, and the perigee sequence will be computed as an equispaced vector from the initial altitude to the intermediate perigee given; whereas, if given as a vector, one shall put all the intermediate perigees up to the second-to-last. Note that in the vector case, the intermediate perigee must have as first entry the starting altitude.

- **Propagation duration** must be a vector with length equal to the number of burns, with the first entry rapresenting the propagation before the first maneuvre, and the remaining representing the propagation after each maneuvre. If a propagation should not be considered, one must put 0 has the duration.

The rest of the inputs is self-explanatory. It is worth noticing, however, that the **final perigee** altitude is set to 50 km as recommended by [12], and it represents the last perigee altitude to be reached, such that the last maneuvre will be from the last entry of intermediate perigee to the final perigee. Note that the code has internal checks on both the excel structure and the input vector length, so it's fundamental to accurately build the excel file.

3.1.2. XML file

The XML file is needed in order to run the executables, as explained in Appendix B. The code uses 3 executables, namely

- `OrbitWizard`: it allows for the all the conversions among orbit definitions.

- `OrbitPropagation`: it allows for the propagation of the orbit.

- `PropagationTransformations`: it allows for the conversion of the output of `OrbitPropagation`.

To make the code as flexible as possible, the executable `OrbitWizard` is used. This allows for the

definition of the orbit in any manner (keplerian elements, repeat cycle, etc...) and then transforms the input to mean keplerian elements expressed in the ToD frame, which, as explained in section 2.1, is fundamental for the code to work. Then, `OrbitPropagation` is needed to propagate the orbit, as explained in section 2.3. However, since the propagation is numerical, it outputs the results in cartesian elements expressed in the Mean Earth Equator at J2000 (MEE2000) frame. That's the reason why `PropagationTransformations` is needed, as it allows to convert back to mean keplerian elements in the ToD frame the output of the propagation. For this file, the user needs to define the initial orbit in the `OrbitWizard` node, and the integration settings in the `OrbitPropagation` node. A snapshot of the structure of the XML is shown below.

```
<algorithms>
    <propagation>
        <orbitPropagation>
            <scenarioItems>
                <satellite/>
            </scenarioItems>
            <propagationType>
                <numericalPropagation>
                    <epoch>
                        ...
                    </epoch>
                    <duration unit="day"> ... </duration>
                    <timeStep unit="s"> ... </timeStep>
                    <perturbations>
                        ...
                    </perturbations>
                </numericalPropagation>
            </propagationType>
        </orbitPropagation>
    </propagation>

    <tools>
        <propagationTransformations>
            <input>
                ...
            </input>
            <output>
```

```
                <stateVector>
                    <referenceFrame>TrueOfDate</referenceFrame>
                    <elementsType>MeanKeplerian</elementsType>
                </stateVector>
            </output>
        </propagationTransformations>

        <orbitWizardCalculation>
            <epoch> ... </epoch>
            <orbitDefinition> ... </orbitDefinition>
            <stateVector>
                <outputReferenceFrame>TrueOfDate</outputReferenceFrame>
                <outputElementsType>MeanKeplerian</outputElementsType>
            </stateVector>
        </orbitWizardCalculation>
    </tools>
</algorithms>
```

3.1.3. __init__ method

As soon as the instance of the class is created, the $\_\_init\_\_$ module is run. This module simply reads the inputs and run the OrbitWizard tool; then, it stores the results in a dictionary. The dictionary is the object that will be updated step by step during the simulation.

3.2. Software functionality

Once the instance of the class is created, the methods stated in chapter 3 can be run as follows:

```
EOL.run(mode)
EOL.generateOutput()
EOL.setExcelValue(cell, value)
```

The `.run` method needs one additional input, that is the mode to be considered. In fact, as explained in section 1.4, the code extends the functionalities of the original MATLAB code; however, depending on the phase of the mission analysis, one may not need the enanched features, so also the basic simulation

can be run. So, again, to provide the most flexibility, two modes can be run. Mode 1 allows for the simulations of the sequence of maneuvres only, like the MATLAB code did, a feature commonly used in the first phases of a mission analysis. Conversely, mode 2, allows for the simulations of the maneuvres considering also the delay among them. This feature is used for finer simulation, usually in more advanced phases of the mission. It has been found out that running mode 2 with large propagation time can lead to huge software run time, since for each mass iteration all the propagations should be performed. Thus, when mode 2 is run, actually the mode 1 is launched, but its results are not stored a part from the wet mass. This, in fact, will be used as a refined initial guess for the mass bisection method. From this value, symmetric bounds based on experience and previous results are calculated (a value of ± 30 kg was used), and the simulation goes on. It this way, the heavy iteration, where also the propagation is accounted for, are limited, thus guaranteeing a faster code.

The `.generateOutput()` method instead generates all the output files, which will be discussed in depth in section 4.2.

The method `.setExcelValue(cell,value)` allows the user to change the value of a cell in the `excel` file. This allows for an automated routine in case of parametric analysis. In fact, as it will be discussed in section 4.3, it turns out that the propagation time is an important tuning parameter to ensure the proper landing in a given Designated Landing Area (DLA). The method `.setExcelvalue` was coded to permit a fast and easy way to obtain the optimal condition.

3.3. Software perfomances

The software was developed in Python as an evolution of the original MATLAB code. While MATLAB proved to be significantly faster, this difference in performance is largely due to intrinsic disparities between the two languages. By profiling the Python code, it was discovered that some functions, such as those for angle conversions (e.g., from true anomaly to mean anomaly), were called over 2 million times during the simulation of 9 maneuvers. This behavior was consistent in both MATLAB and Python, yet MATLAB consistently outperformed Python in execution speed.
To quantify the difference, a simple test was conducted: an angle conversion function was executed 2 million times in both languages. The MATLAB implementation completed the task in approximately 0.5 seconds, while Python required around 6 seconds. However, by leveraging Python's `math` package for optimized mathematical operations, the runtime was reduced to about 2 seconds, as shown in

Table 3.1

It is important to note that Python is free and open-source, unlike MATLAB, which requires a license. This makes Python a more accessible and cost-effective choice despite its slower performance. Furthermore, the accuracy of the results produced by the Python implementation was identical to that of the MATLAB version, ensuring that the change in language did not compromise precision.

Number of Calls	**Python Runtime (s)**	**MATLAB Runtime (s)**
2,000,000	6.0	0.5
2,000,000 (with `math`)	2.0	0.5

Table 3.1: Comparison of runtime for different implementations and optimizations.

3.4. Software testing

An extensive testing campaign has been conducted to assess the validity of both modes. In particular, the critical point were the maneuvre model for mode 1 and the state update for mode 2. In fact, the maneuvre model should match exatcly the model implemented in MATLAB, that was tested against some DMS internal software, whereas the state update should be coherent among all the different maneuvres, and no conflicts should arise.

3.4.1. Mode 1 testing

The reference to test mode 1 was the MATLAB code. In order to test the maneuvre simulation, the same inputs were given to both codes; then a `.txt` file with the quantities to be checked was proceduced and compared. For a more effective comparison, a set of 9 maneuvres, including small and big ones, were given. The output `.txt` file contains several informations, including ΔV_{ideal}, ΔV, propelled arc duration, gravity losses, sequence of the perigee reached, sequence of the semi-major axis reached, sequence of the eccentricity obtained and the wet mass. It was then checked that all the quantites matched up to the 5th decimal digit.

3.4.2. Mode 2 testing

Finding a reference software for mode 2 was not straightforward as it was for mode 1; in fact, no code aimed at the simulation of a sequence of maneuvres and propagations existed in the DMS mission analysis framework explained in Appendix B. The answer was found in an operational software, called Flight Formation Simulation, that is not build for the re-entry, but can simulate a sequence of maneuvres considering also the effect of perturbation. This operational software needs an accuracy level higher than the one developed; in fact, the finite maneuvre is not approximated as the sum of infinitesimal ideal maneuvre, but the contribution of the ΔV is directly considered inside the differential equations, together with the perturbation, and step by step the solution is integrated. Also, the propagation is performed with different means rather than using the desEO executables, so an exact match of all the quantities was not possible. What was checked was the relative error, computed using Equation 3.1, where X is a generic quantity, rapresenting either a keplerian element or a maneuvre-associated quantity. In order to consider the testing a success, the relative error among all the quantities must remain below 1%.

$$\frac{\|X_{ffsim} - X_{python}\|}{\|X_{ffsim}\|} \tag{3.1}$$

In this case, the input given to both software was a sequence of propagation, maneuvre, propagation and maneuvre. To check if errors would accumulate over time, the first maneuvre was set as a small 30 km perigee lowering, whereas the second one was a large 150 km perigee lowering.

4 | Application Case

Let's consider as an example the re-entry of a S/C whose orbit is a Sun Synchronous Orbit (SSO), with frozen condition and whose mean keplerian elements, referred to the ToD frame, are the one shown in Table 4.1.

Semi-major axis [km]	**Eccentricity [-]**	**Inclination [deg]**	Ω **[deg]**	ω **[deg]**
7010.271	0.001054	97.938	142.431	90.0

Table 4.1: Initial orbital parameters

Assume that the re-entry is due to start on February, 12, 2025 at 12:00. Now, assume that the number of maneuvres to be performed is 6, and that some constraints exist such that the choice of the sequence of perigee altitudes is not free. For example, let's consider the following setup:

- 1st and 2nd maneuvre shall be equal in ΔV with a precision level of 10%.
- 3rd and 4th maneuvre shall be equal in ΔV but its value shall be 85% of the previous one with a precision level of 10%.
- 5th maneuvre shall be small, so that any errors in previous states can be easily corrected, and bring the S/C to a perigee altitude of 250 km.
- 6th maneuvre shall bring the perigee to an altitude of 50 km.

For this sequence of maneuvre, consider that the propulsive system is the one described in Table 4.2.

Man ID	Thrust [N]	Specific Impulse [s]
1	66.5	213.5
2	65	213
3	64	212.7
4	62	212.4
5	60.5	212.1
6	58	212

Table 4.2: Propulsive system performance

In addition, let's suppose that also the propagation time must take into consideration some constraints. For example, let's say that:

- Before the 1st maneuvre at least 3 days shall pass to start the passivation of the systems.
- The whole re-entry shall be not longer than 2 weeks.
- After the 1st and 2ndmaneuvre 2 days shall pass to guarantee good orbit determination.
- After the 3rd and 4th maneuvre 1 days $\pm$ 12 hours shall pass. The uncertainty in the propagation time is needed to adjust the landing area of the S/C and shall be found.
- After the 5th maneuvre 10 orbits shall pass.

In addition to that, the perturbation to consider, a part from the gravitational ones, are drag and the solar radiation pressure. With these informations, the input `excel` and `xml` file can be easily filled and the simulation can be run.

To generate the outputs the method `.GenerateOutput()` can be run, and a series of folders and documents will be generated, according to the structure reported in Figure 4.1. In the following sections each folder and file will be described.

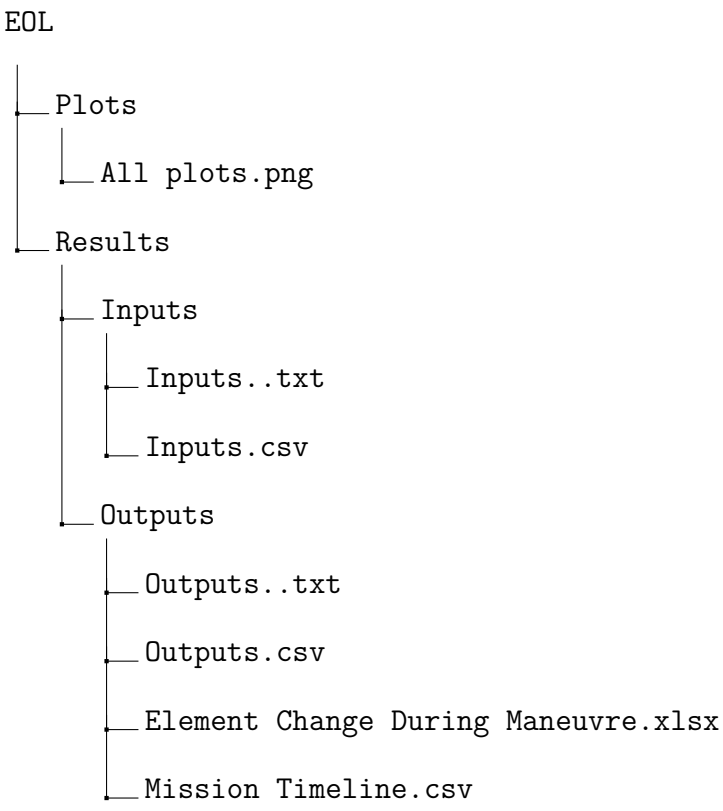

Figure 4.1: Directory structure generated by the `generateOutput()` method.

4.1. Plots

The generated plots aim at showing the variation of the main quantities during all the phases of the simulation. The main events happening are the maneuvres, so the focus is on the value of the quantities before the maneuvre (which coincide with the end condition of the propagation) and after the maneuvre. In particular, plots of all the mean keplerian quantities, except the anomaly, are produced, as well as the perigee altitude, Figures 4.2 and 4.3 show the semi-major axis trend. It is worth noticing that based on the mode run, slight changes happens among the plots. The differences between the two graphs are mainly formal. In fact, with mode 1 the epoch is not treated, so the x-axis shows only the maneuvre ID to be considered, and the variation of the quantity is instantaneous; instead, for mode 2, where the epoch is considered and updated, the x-axis correctly shows it, while also considering the true separation between dates. This is the reason why some lines are almost vertical, as the maneuvre duration is in the order of minutes, whereas the propagation is in the order of days.

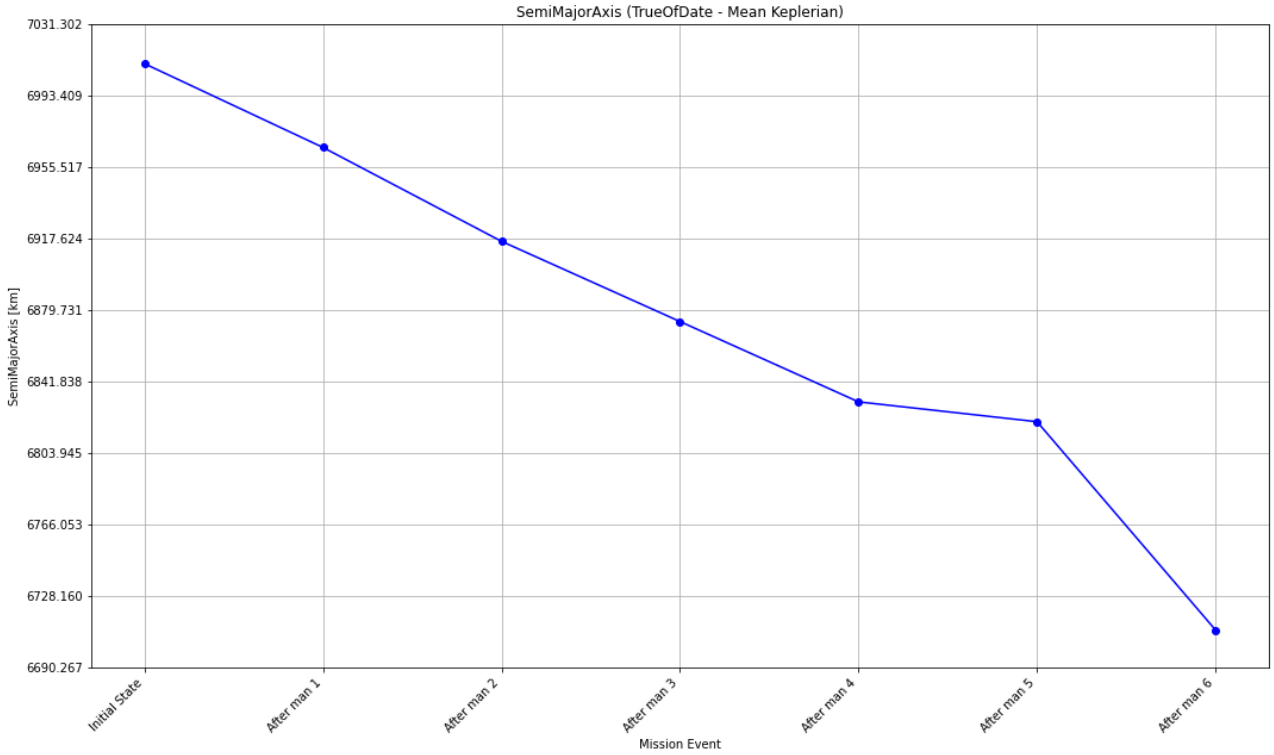

Figure 4.2: Mode 1 semi-major axis plot.

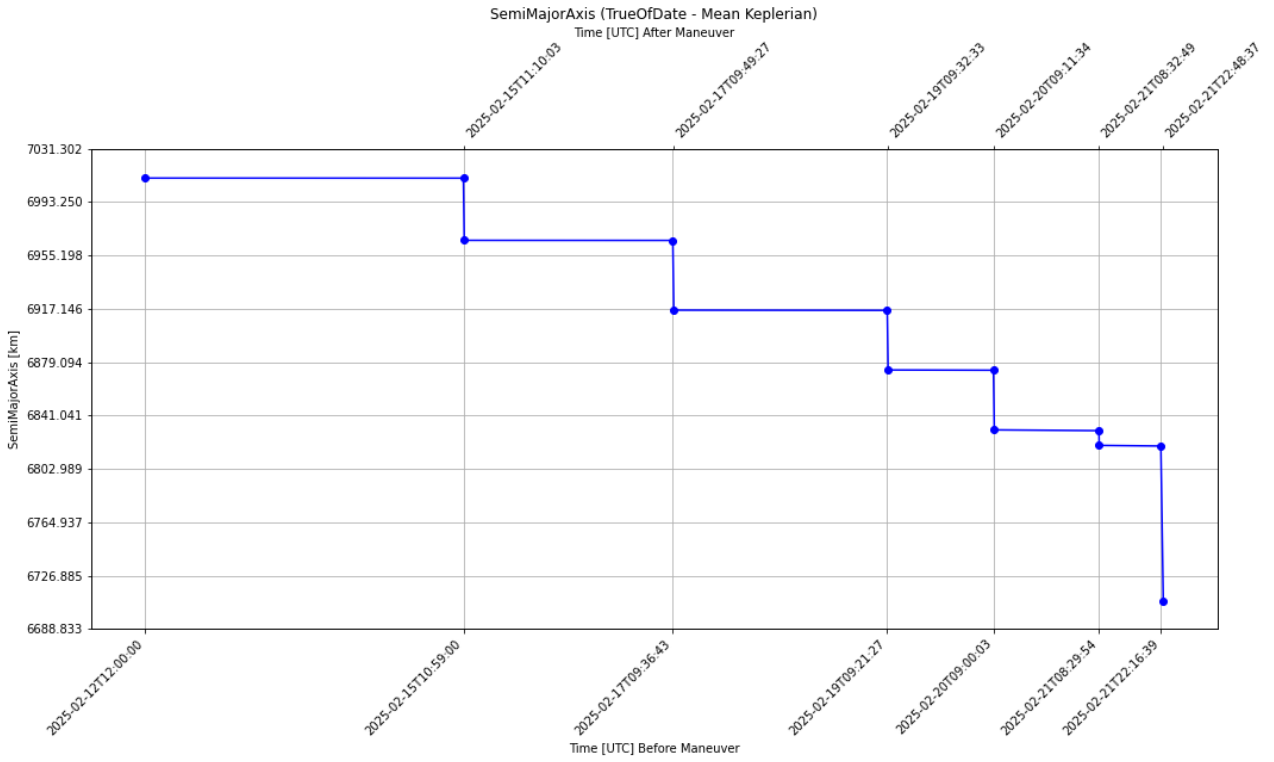

Figure 4.3: Mode 2 semi-major axis plot.

4.2. Results

Despite their usefulness for interpreting a simulation outcome, plots provide solely a qualitative glance on the results.

To assess the results obtained, correct tuning the parameter and making considerations or corrections, precise numbers are needed. To do so, various file are generated within the output folder, whereas a copy of the input file is generated inside the input folder. It is worth noticing that the redundancy of files is present due to precise mission analysis reasons.
For example, even tough the inputs are specified through the `excel` and `xml` files, it's important to create a further input file that allows for the repeatability of the simulation. In fact, the `excel` and `xml` file are meant to be changed for each simulation performed, thus the corresponding input file may be lost if not immediately saved. This is why the input file is generated, so that each simulation has attached within itself a file stating the input. Nonetheless, two versions of the input file are generated, a `.txt` and a .csv one. This is done because the `.txt` is clearly readable, whereas the `.csv` is easily imported in any software if some post-processing is needed. To maintain the order, the output and input folders are put inside a result folder.

4.2.1. Input subfolder

In order to asses the inputs used and to guarantee the repeatability of the simulation, a file with the input used is generated. This file must contain all the values used, and it must be readable. Table 4.3 shows a snapshot of the input file in csv version.

SIMULATION INPUTS	
Parameter	**Value**
Initial altitude [km]	632.135
Initial eccentricity [-]	0.001054
Final perigee altitude [km]	50.0
Specific impulse (1st maneuver) [s]	213.5

Specific impulse (2nd maneuver) [s]	213.0
Specific impulse (3rd maneuver) [s]	212.7
...	...
Thrust (1st maneuver) [N]	66.5
Thrust (2nd maneuver) [N]	65.0
Thrust (3rd maneuver) [N]	64.0
...	...
Target pericenter (1st maneuver) [km]	537.0
Target pericenter (2nd maneuver) [km]	439.0
Target pericenter (3rd maneuver) [km]	355.0
...	...
Initial propagation duration [days]	3.0
Time between maneuvers 1 and 2 [days]	2.0
Time between maneuvers 2 and 3 [days]	2.0
...	...
Dry mass [kg]	1700.0
Number of burns [-]	6

Table 4.3: Input file. Dots indicate omitted data for brevity.

4.2.2. Output subfolder

This folder sligtly changes its elements based on the type of simulation that has been run. Note that even though the differences between the two modes output are stated, all the reported results refer to mode 2. Independently of the mode run, a file called *Output* is generated, both as a `.txt` and as a `.csv` file. In this file the maneuvre quantities are reported, such as ΔV, propulsive arc duration, gravity losses, etc... In addition to that, a sort of written representation of the plots is presented. This means that keplerian quantities are stated before and after the maneuvre. If mode 2 is run, also the effect of the propagation is shown in the file. A snapshot of the `.csv` version of the file is shown in Table 4.4.

SIMULATION OUTPUTS	
Parameter	**Value**
Starting altitude [km]	632.135
Total ΔV for DeOrbiting [m/s]	170.0165
Total ΔV for DeOrbiting with 15% margin [m/s]	179.681
Deorbiting period [min]	91.147
Initial mass [kg]	1844.438
INITIAL PROPAGATION (DURATION: 3.0 days)	
Orbital Quantities	**Before - After**
Semi major axis [km]	7010.271 - 7010.227
Eccentricity [-]	0.001054 - 0.001053
Perigee altitude [km]	624.746 - 624.704
BURN 1	
Parameter	**Value**

ΔV [m/s]	27.275
ΔV_{ideal} [m/s]	26.807
Gravity losses [m/s]	1.0287
Arc duration [min]	11.054
Propellant mass used [kg]	21.059
Orbital Quantities [Units]	**Before - After**
Semi major axis [km]	7010.227 - 6965.906
Eccentricity [-]	0.001053 - 0.00728
Perigee altitude [km]	624.704 - 537.000
PROPAGATION 2 (DURATION: 2.0 days)	
Orbital Quantities	**Before - After**
Semi major axis [km]	6965.906 - 6965.842
Eccentricity [-]	0.00728 - 0.00728
Perigee altitude [km]	537.000 - 536.990
...	

Table 4.4: *Output.csv*. The dots indicate that the table has been truncated for brevity.

Several information can be extracted from this file. For example, Table 4.5 shows the initial mass and the ΔV needed to complete the controlled re-entry. This value will be used as dry mass for the next part of the ΔV budget, as explained in section 1.3. Obviously, the ΔV information is strictly related to the mass needed, and it is compliant with the guidelines and margins cited in section 1.3.

Wet Mass [kg]	Total ΔV [m/s]
1844.438	179.681

Table 4.5: Wet mass and ΔV results.

Also, it can be seen how the requirement on the ΔV are satisfied. The graph in Figure 4.4 shows that the first two maneuvers have very similar ΔV values. The second two maneuvers also have similar ΔV values, which are approximately 85% of the ΔV of the first two. In contrast, the fifth maneuver has a significantly smaller ΔV.

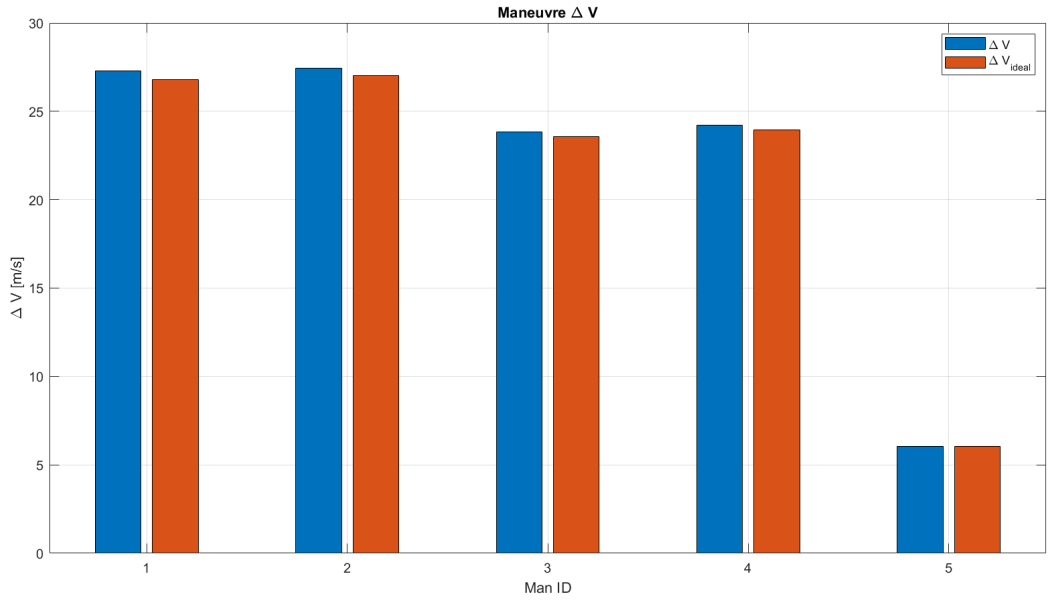

Figure 4.4: ΔV required to perform the first 5 maneuvres.

In addition, Table 4.6 summarizes the maneuvre result, again showing that the requirement are fully satisfied.

Man ID	**1**	**2**	**3**	**4**	**5**	**6**
Arc duration [min]	12.055	12.739	11.108	11.519	2.925	31.967
ΔV	27.275	27.426	23.839	24.225	6.045	64.429
Gravity losses [%]	1.40	1.43	1.10	1.19	0.08	9.69
Propellant mass [kg]	23.060	23.776	20.443	20.565	5.102	53.490
Perigee reached [km]	537.000	439.001	355.000	271.000	250.000	50.001

Table 4.6: Maneuvre associated quantity result.

Another file generated, independently from the mode run, but with slight changes in its form depending on it, is the file *Element Change During Maneuver*. This is an excel file with a number of sheets equal to the number of maneuvres. In each sheet, the variation of the parameters affected by the maneuvre is shown. Basically, as explained in section 2.1, the propulsive arc is discretized in infinitesimal pieces, and at each piece the state is updated. This file collects all of this information and store them nicely. This file basically rapresents the integration process done during the maneuvre, and it is useful to compare the maneuvre implementation step by step. Note that the epoch, again, is only treated if mode 2 is run.

A snapshot of the file can be seen in Table 4.8. The table has been divided for readability, but the first row of both tables is common. The table shows the first rows of the sheet relative to the 1st maneuvre; similar tables are generated for all maneuvres. This file is useful to have a glance on what's really happening during the maneuvre and can be used as a "sanity check" on the hypothesis used. For example, as said in section 2.1, the maneuvre is assumed symmetric with respect to the absidis line. This file shows that this is not exactly true, since the apocenter condition, that is anomaly = 180°, is not reached at half the maneuvre duration. However it's close enough for the needed precision of the software.

Epoch [UTC]	**Time elapsed [min]**	**SMA [km]**	**Ecc [-]**	**Anomaly [°]**
2025-02-15T10:58:59	0	7010,233578	0,0010535	159,5762842

Epoch [UTC]	Time elapsed [min]	SMA [km]	Ecc [-]	Anomaly [°]
2025-02-15T10:59:00	0,012495926	7010,200103	0,001057982	159,6973317
2025-02-15T10:59:00	0,020826544	7010,166628	0,001062467	159,8174886
2025-02-15T10:59:01	0,029157161	7010,133153	0,001066955	159,9367654
2025-02-15T10:59:01	0,037487779	7010,099679	0,001071447	160,0551726
…				

AoP [°]	AoP Variation [°]	Perigee Altitude [km]	Mass [kg]	Mass burnt [kg]
89,9071865	0	624,8482946	1844,438477	0
89,81694423	-0,090242273	624,7834398	1844,422606	0,015870229
89,72759281	-0,089351415	624,7185608	1844,406736	0,015870229
89,63912171	-0,088471101	624,6536578	1844,390866	0,015870229
89,55152053	-0,087601178	624,5887313	1844,374996	0,015870229
…				

Table 4.8: *Element change during manoeuvre*. The dots indicate that the table has been truncated for brevity.

Lastly, the file *Mission Timeline.csv* is only generated if mode 2 is run, which is also the most useful file. It shows the state, epoch, and mass, before and after an event of the mission, that is, a maneuvre. This file allows for quick and clear insights on the whole procedure, and also it allows for an easy collaboration with other divisions to ensure the correct execution of the controlled re-entry, as it will be discussed in section 4.3. A snapshot of the files can be seen in Table 4.10.

Time [UTC]	Mean Sma (ToD) [km]	Mean Ecc (ToD) [-]	Mean Inc (ToD) [deg]
2025-02-12T12:00:00	7010.271	0.00105	97.9389
2025-02-15T10:59:00	7010.227	0.00105	97.9383
2025-02-15T11:10:03	6965.906	0.00728	97.9383
2025-02-17T09:36:43	6965.842	0.00728	97.9377
...	...	...	...

Mean Ω (ToD) [deg]	Mean ω (ToD) [deg]	Mean Anomaly (ToD) [deg]	Mass [kg]
142.431	90.0	270.120	1844.438
145.346	89.921	159.581	1844.438
145.354	89.936	200.605	1823.378
147.304	84.462	156.163	1823.378
...	...	...	...

Table 4.10: *Mission Timeline.csv*. The dots indicate that the table has been truncated for brevity.

This file is really representative of the whole mission. In fact it summarizes the mission in steps and allows to check all the requirement. In fact, not only the wet mass is shown, but also the re-entry duration can be easily computed looking at the first and last row. The total duration is about 9 days, satisfying the requirement.

4.3. Iterative process to complete the controlled re entry

As it has been said in subsection 1.2.2, the idea of the controlled re-entry is to make sure that the re-entry of the S/C happens in a controlled way and over a DLA. A common DLA is the SPOUA, a portion of the pacific ocean that is inhabited.

In order to address that, two more things are needed. First of all, we need a plot of the S/C ground track; secondly, we need a fragmentation model to assess the point where the fragments will end up. These parts are out of the scope of the presented software, since they are dealt by the atmospheric flight department, which implements the physics needed to deal with this problem. Nonetheless, to achieve the correct execution of the controlled re-entry, a collaboration between the mission analysis team and the atmospheric team is needed.

To resolve the landing area problem, one is mainly interested in the last maneuvre point. In particular, the point of main interest is called *intermediate perigee*, that is the second-to-last perigee reached. This point can be characterized by its state using the keplerian elements and by its epoch. However, since the anomaly of the perigee is fixed, it is substituted with the epoch. Considering the example that is being analyzed, the *intermediate perigee* state is shown in Table 4.11, and can be easily extracted as third-to-last row in the mission timeline output file.

Semi-major axis [km]	**Eccentricity [-]**	**Inclination [deg]**	**Ω [deg]**	**ω [deg]**	**Epoch [UTC]**
6819.728	0.02809	97.937	151.461	71.678	2025-02-21T08:32:49

Table 4.11: *Intermediate perigee* state.

The state at the intermediate perigee is assumed to be very precise, since previous errors can be corrected before and will serve as reference state for the following analysis. As stated in chapter 4, before making the last maneuvre 10 orbits shall pass. When the last maneuvre is performed, it is assumed that the control over the S/C is no more present. As discussed in subsection 4.2.2, the needed ΔV, the propellant mass, and the maneuvre duration are known. These information will then be passed to the atmospheric re-entry department, that will then continue the simulation according to ESA guidelines [12] and [6].

The next analysis consists in varying a list of parameters to account for different scenarios and ensure that all surviving fragments land in uninhabited areas with the statistical compliance required by the applicable standards and requirements. A list of dispersed parameters is identified to cover all the uncertainties at system level that cannot be corrected before the last perigee-lowering manoeuvre. The uncertainties related to the disposal manoeuvres are considered in orbital simulations, resulting in a set of dispersed initial conditions at the Earth Interface Point (EIP) (defined at 120 km as the interface

between exo-atmospheric and endo-atmospheric conditions). Besides that, other uncertainties related to the atmospheric re-entry are considered, such as uncertainties on the atmospheric density, break-up altitude, etc. All of this is used as input for a simulation using an ESA approved software, called Debris Risk Assessment and Mitigation Analysis (DRAMA). The output of the simulation consists of the on-ground footprint of the surviving fragments. Based on the resulting footprint, safety boxes are defined for which the probability that a fragment falls outside the boundaries of these safety boxes is below a certain threshold. For a controlled re-entry, the DRA, that represents the area where the debris are enclosed with a probability of 99%, and SRA, that represents the area where the debris are enclosed with a probability of 99.999%, should be computed. An example of those areas is reported in Figure 4.5.

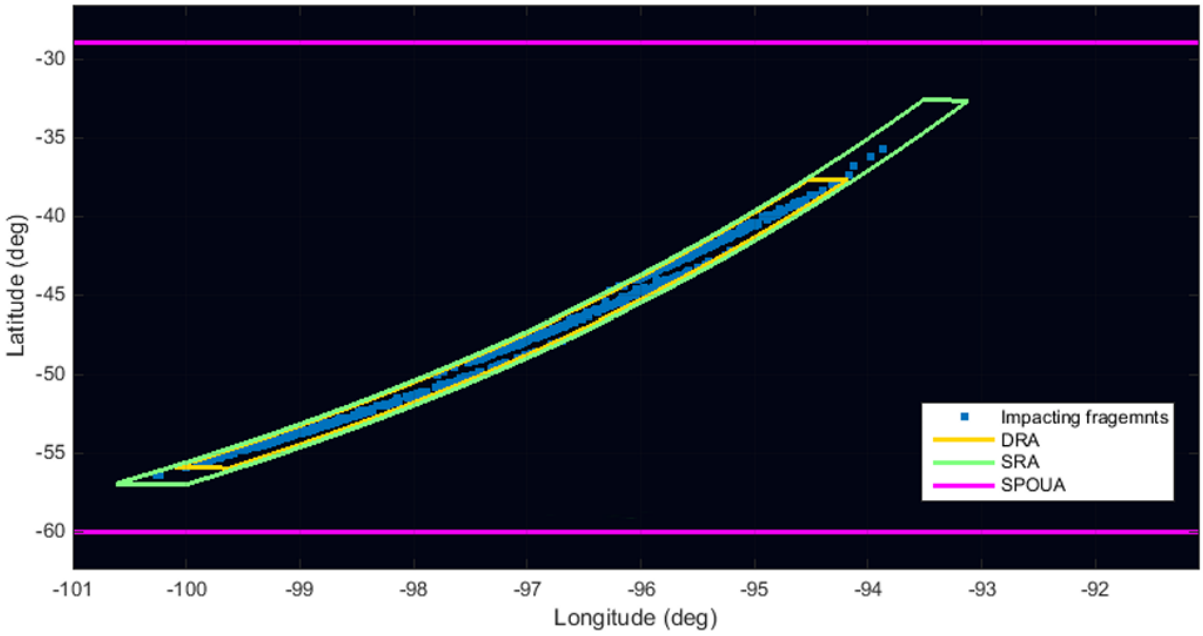

Figure 4.5: Example of DRA and SRA [6]

So, a typical iteration consists in simulating the sequence of maneuvres and then passing the intermediate perigee state, as well as the last burn characteristic, to the atmospheric simulation. Then those quantities will be used as nominal and perturbations will be inserted on the EIP to run a Monte Carlo simulation to make sure that the landing area is reached in a statical sense.

A typical output ground track plot with the end point reached is shown in Figure 4.6.

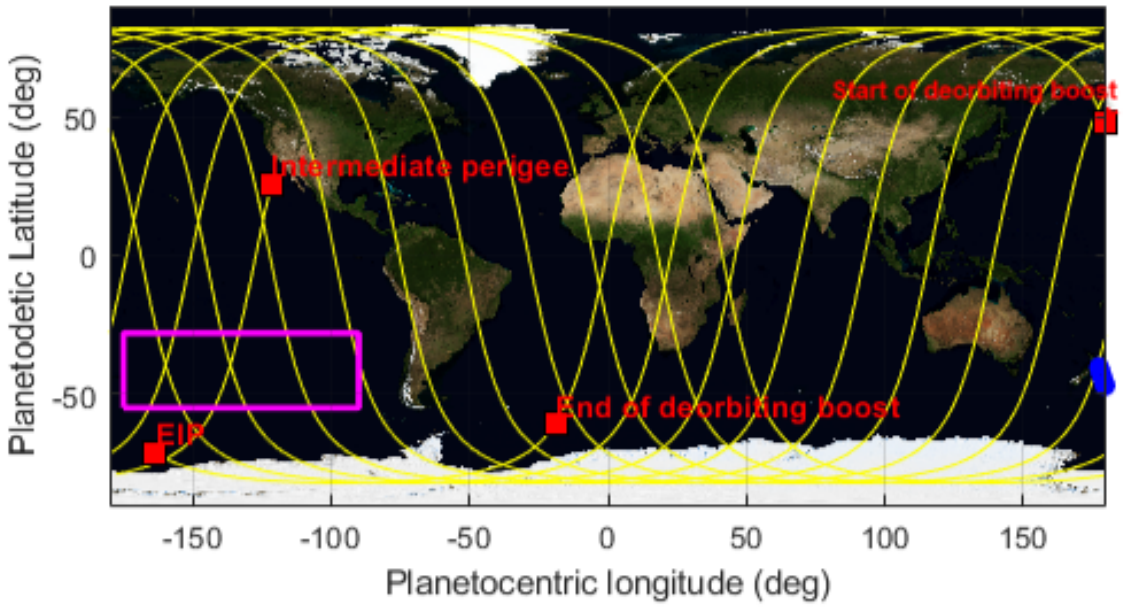

Figure 4.6: Ground track plot from the *intermediate perigee* to the fragmentation. The blue dots represents the fragments.

In the graph the *intermediate perigee* is shown as well as other point of interest such as the *Start of deorbiting boost*, *End of deorbiting boost*, and *EIP*. Basically, from the *End of deorbiting boost*, the orbit is propagated until the EIP is reached. There, more uncentarties are added and a Monte Carlo procedure is run to see where the fragments end up.

As it can be seen from the graph, the debris fell outside the SPOUA. In this case we need to adjust some parameters to make sure that the landing actually ends up in the right place. Since everything after the *intermediate perigee* is more or less fixed, as the maneuvre is fixed and varies only slightly, while the propagation time is fixed, we need to tune the planetodic latitude and planetodic longitude of the *intermediate perigee*. This is obtained by tuning some parameters in the *intermediate perigee* state.

The planetodic longitudine λ and latitude ϕ can be computed using Equation 4.1 and Equation 4.2.

$$\lambda = \arctan\left(\frac{Y_{\text{ECEF}}}{X_{\text{ECEF}}}\right) \tag{4.1}$$

$$\phi = \arcsin\left(\frac{Z_{\text{ECEF}}}{r}\right) \tag{4.2}$$

Where r is the orbit altitude, whereas the vector $\vec{X}_{\text{ECEF}} = [X, Y, Z]_{ECEF}$ is the cartesian position vector in the Earth-Centered Earth-Fixed (ECEF) frame. To obtain the $\vec{X}_{ECEF}$ vector from the orbital elements we need first to transform the keplerian state into $\vec{X}_{ECI}$, the cartesian vector in the Earth Centered Inertial (ECI) frame (see Appendix A for the frame definitions). Defining as Θ the true anomaly, the orbital state $\vec{X}_{ORB}$ is obtained as $\vec{X}_{ORB} = [r \cdot \cos\Theta, r \cdot \sin\Theta, 0]^{\text{T}}$ Then, using the 3-1-3 rotation of Euler angles, we can easily obtain the cartesian vector in the ECI frame as shown in Equation 4.3.

$$\vec{X}_{ECI} = \overline{\overline{R}}_z(\omega) \cdot \overline{\overline{R}}_x(i) \cdot \overline{\overline{R}}_z(\Omega) \cdot \vec{X}_{ORB} \tag{4.3}$$

Then, in order to pass to the ECEF frame, we need to take into consideration the Earth rotation. This is done using Equation 4.4, where Θ_{GST} is the Greenwich-Sidereal-Time, defined as the angle between the Greenwich meridian and the inertial X axis at a given epoch.

$$\vec{X}_{\text{ECEF}} = \overline{\overline{R}}_z(\Theta_{\text{GST}}) \cdot \vec{X}_{\text{ECI}} \tag{4.4}$$

It's important to point out that we are looking at the condition to change λ and ϕ of the *intermediate perigee*, whose altitude is fixed at 250 km. This means that any combination of semi-major axis and eccentricity should lead to that value. This leads to the conclusion that none of semi-major axis and eccentricity play a role in changing the *intermediate perigee* position.

Instead, Euler angles and the epoch play an important role in changing the coordinates. In particular, the Euler angles change the orbital plane, changing consequently the position of the intermediate perigee on the Earth, wheareas the epoch do not interfere with the orbital plane, but accounts for the Earth rotation.

Table 4.12 summarizes the effects of those parameters.

Parameter	**Effect on λ**	**Effect on ϕ**
Ω	Small	None
i	Small	High
ω	High	Small
Epoch	High	None

Table 4.12: Effect of parameters on λ and ϕ.

In summary, the effect of the parameters can be stated as follows:

- Ω: it rotates the orbit around the z-axis, so there is no effect on ϕ, but we have a contribution on λ.
- i: it changes a lot ϕ influencing the maximum ϕ that can be reached. It also has a smaller effect on λ due to the rotation of the plane.
- ω: it changes the pericenter position and has an effect on both ϕ and λ. However, the effect on ϕ is more direct, whereas the effect on λ is non-linear and difficult to predict.
- Epoch: a change in the epoch causes a change in Earth rotation, causing a change in λ

It is important to point out that this table should be read as how much a change in a given parameter affects λ and ϕ. For example, looking at the effect on λ, a change of $5°$ in Ω has a smaller effect than the same change in ω.

Now, looking again at Figure 4.6, we need to figure out what we can do to adjust the *intermediate perigee* position.
Obviously, each case should be studied on his own. In our case the latitude ϕ is ok, since it falls inside the SPOUA bounds. If that is not the case, looking at Table 4.12, we can only play with i and ω. Some solutions might be:

- To consider an out-of-plane maneuvre to change i. This solution is really expensive in terms of ΔV and not really viable.

- To consider a pure $\Delta\omega$ maneuvre. This solution is not so expensive in terms of ΔV but a further maneuvre should be performed.

- To consider to increment/decrement the propagation time to allow the perturbation to naturally change ω until the wanted value is reached. This solution is obviously the best in terms of ΔV, but it's not always applicable due to time constraint.

In the end, the solution is not so trivial and it really depends on the application.

Luckily, in our case the ϕ is not a problem; however we need to fix the λ. To tune this parameter, again looking at Table 4.12, we can play with basically all the parameters, however Ω and i should be immediately discarded since their effect is too small to be easily implemented. It turns out that the easiest parameter to play with is the epoch. In fact, λ is really sensitive to the epoch. In fact, the epoch effect is proportional to the Earth rotation velocity, that is around 15 deg/hour. Also, it is not worth to act on ω to change λ since it has combined non linear effect also on ϕ.
In the requirement, stated in chapter 4, the third and forth propagation could be adjusted with ± 12 hours, meaning that we can use this to adjust λ. Also, since we need to adjust the epoch just by a couple of hours, the value of the other parameters won't be affected so much, since the perturbation don't change so much over such small period. In particular, we expect an increment of the propagation time to lead to an increase in λ, thus the *intermediate perigee* will move to the right. Let's add, for example, 3 hours on the third propagation and run again the simulation. Now the *intermediate perigee* state will be the one shown in Table 4.13.

Semi-major axis [km]	**Eccentricity [-]**	**Inclination [deg]**	Ω **[deg]**	ω **[deg]**	**Epoch [UTC]**
6819.728	0.02809	97.937	151.461	71.678	2025-02-21T11:32:49

Table 4.13: *Intermediate perigee* updated state.

After the atmospheric department simulation, we get the plot shown in Figure 4.7. By modifying the propagation time we adjusted the position of the *intermediate perigee* just enough so that now the fragments ends right in the middle of the SPOUA.

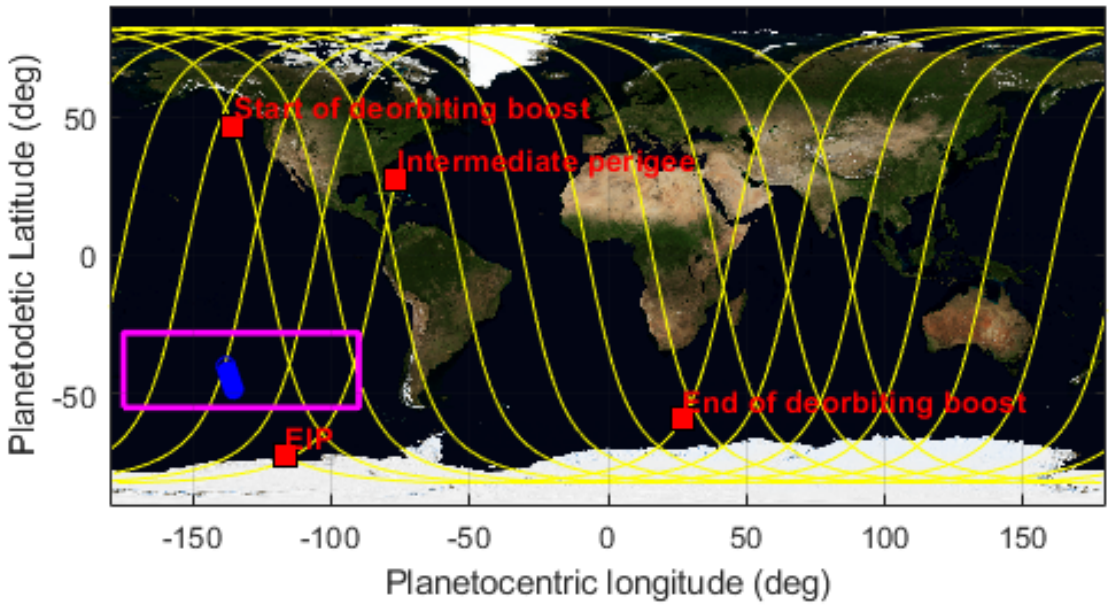

Figure 4.7: Ground track plot from the *intermediate perigee* to the fragmentation.

In summary, a back and forth process between the maneuvre simulation and the atmospheric analysis is needed to obtain the optimal conditions of the parameters to make sure that the landing is correct. Of note, each case should be treated differently, and ad hoc solutions must be implemented for each case. This procedure explains the reasoning that one should do in order to change the *intermediate perigee* conditions.

5 | Conclusion

The rapid increase in the number of artificial satellites in Earth orbit over recent decades has brought about significant challenges, particularly the growing issue of orbital debris. These debris fragments pose a substantial collision risk to operational satellites, threatening the safety and sustainability of space operations. To mitigate these risks, international guidelines and best practices have been developed, aiming to reduce the proliferation of debris and manage EOL scenarios effectively.
Among these guidelines, specific recommendations address the disposal of satellites at the end of their operational lives. Two primary strategies are typically considered for EOL disposal: uncontrolled re-entry and controlled re-entry. While uncontrolled re-entry relies on natural orbital decay processes to bring the satellite back into Earth's atmosphere, controlled re-entry involves precise maneuvering to ensure the satellite's re-entry occurs over designated uninhabited areas, minimizing risks to people and property.

This thesis focuses on the controlled re-entry strategy, with a particular emphasis on its application within the context of a specific software framework used in satellite mission design, developed during an internship at the Earth Observation Mission Analysis (EOMA) team in DMS. The main objective of the work was to develop a robust, flexible, and easy-to-use software module that facilitates the planning and execution of controlled re-entry maneuvers.
The ease of use of the software stems from the development of a dedicated class, allowing the entire program to run with just a few lines of code. Flexibility was achieved through the straightforward customization of inputs, enabling the software to cover a wide range of application cases.The module underwent a rigorous testing campaign to ensure its reliability and accuracy under various operational scenarios. As a result, the module has been successfully integrated into the company's proprietary software suite and is currently being utilized for an ESA mission in the design phase.

By addressing the challenges associated with controlled re-entry and delivering a practical solution,

this thesis contributes to the advancement of sustainable space operations and aligns with international efforts to mitigate the risks posed by orbital debris.

A | Orbital definitions and reference frames

To describe the motion of a satellite in space, it is crucial to define its orbit using appropriate parameters and frames of reference. An orbit can be characterized in several ways, depending on the desired level of precision, the context of the analysis, and the type of mission. The most common methods involve the use of Mean Keplerian elements, Osculating Keplerian elements, repeat cycle, and cartesian elements. Additionally, a proper understanding of the reference frames used to express these elements is fundamental for interpreting orbital data. This chapter provides an overview of these methods and reference frames.

Orbital definition

Cartesian elements

Cartesian elements describe the position and velocity of a satellite in a three-dimensional space. These are represented by the following six parameters:

- Position (x, y, z): The satellite's location in a chosen reference frame.
- Velocity (v_x, v_y, v_z): The satellite's velocity components along the x, y, and z axes.

These elements are often used in numerical simulations and are particularly advantageous for integration in differential equation solvers. While cartesian elements provide a straightforward representation of a satellite's state, they are less intuitive for understanding orbital characteristics like shape, orientation, and period.

Mean Keplerian elements

Mean Keplerian elements provide an averaged representation of an orbit over time. They are derived by removing the short-period oscillations caused by perturbations, such as those due to Earth's oblateness, solar radiation pressure, or gravitational influences from the Moon and Sun. These elements are particularly useful for long-term analyses and mission planning, as they describe the secular (long-term) and periodic (medium-term) evolution of the orbit without focusing on rapid variations.

The six classical mean Keplerian elements are:

- Semi-major axis (a): Defines the size of the orbit.
- Eccentricity (e): Represents the shape of the orbit.
- Inclination (i): The tilt of the orbital plane with respect to the reference plane.
- Right Ascension of the Ascending Node (Ω): Describes the orientation of the ascending node.
- Argument of Perigee (ω): Specifies the orientation of the orbit within its plane.
- Mean Anomaly (M): Indicates the satellite's position along the orbit at a specific time.

Mean elements are often used for initial orbit design and mission analysis, where short-term perturbations are not critical.

Osculating Keplerian elements

Osculating Keplerian elements represent the instantaneous orbital parameters at a specific epoch. They describe the orbit the satellite would follow if all perturbative forces were removed, leaving only the two-body gravitational interaction with the central body. These elements capture the actual state of the satellite at any given time, including the effects of perturbations. The six classical osculating Keplerian elements use the same parameters as the mean ones, but instead of the mean anomaly, they use the true anomaly. Osculating elements are ideal for precise orbit determination and short-term predictions. However, their values can vary rapidly due to perturbations, making them less suitable for long-term analysis without additional processing.

Repeat cycle

A repeat cycle orbit is designed so that the satellite passes over the same ground track after a specific number of orbital revolutions and days. These orbits are particularly important for Earth observation missions, where consistent imaging of the same area is required. Repeat cycle orbits are defined by:

- Revisit Period: The number of days before the satellite repeats its ground track.
- Revolutions per Day: The integer ratio of orbital revolutions to Earth rotations that ensures repeatability.

These orbits require precise tuning of the semi-major axis and inclination to achieve the desired repeatability, often balancing the effects of perturbations to maintain consistency.

Reference systems

ECI

The ECI coordinate system is a reference frame in which the origin is at the center of the Earth, and the axes are fixed with respect to distant stars. The axes are typically defined as follows:

- The x-axis points toward the vernal equinox, which is the point in the sky where the Sun crosses the celestial equator moving from south to north.
- The y-axis lies in the plane of the celestial equator perpendicular to the x-axis
- The z-axis is aligned with the Earth's rotation axis, pointing towards the North Pole.

This system is inertial, meaning it does not experience accelerations or rotations relative to distant stars, providing a stable and non-rotating frame of reference. It is used in orbital calculations, such as those for satellites or space trajectory tracking, as it allows for the description of an object's motion in space without accounting for the Earth's rotation.

ECEF

The ECEF coordinate system is a rotating coordinate system that is fixed to the Earth. Its origin is at the Earth's center, and the axes rotate along with the Earth, meaning the system is aligned with the

Earth's surface at any given time. The axes in the ECEF system are defined as:

- The x-axis points toward the intersection of the equator and the Prime Meridian (Greenwich).
- The y-axis lies in the equatorial plane and is 90° east of the x-axis.
- The z-axis is aligned with the Earth's rotation axis, pointing toward the North Pole.

The ECEF system rotates with the Earth, making it ideal for applications related to positioning, navigation, and geospatial data. It is commonly used for tasks such as GPS positioning and geodetic measurements, where the Earth's rotation must be accounted for.

MEE2000

The MEE2000 frame is a widely used inertial reference frame based on Earth's mean equator at the epoch J2000.0 (January 1, 2000, 12:00 TT). In this frame:

- The x-axis points toward the vernal equinox.
- The z-axis is aligned with Earth's rotational axis at the epoch J2000.
- The y-axis completes the triad and lies in the equatorial plane xy.

The MEE2000 frame is particularly suitable for long-term analyses and is often the default reference frame used in celestial mechanics and orbit design.

Mean of Date

The Mean Of Date (MoD) frame is defined similarly to the MEE2000 frame, but it is referenced to the considered epoch and averages out short-period variations of Earth's spin axis, such as nutation. This frame accounts only for periodic and secular perturbations. In the MoD frame:

- The x-axis points toward the vernal equinox.
- The z-axis is oriented toward the averaged Earth's spin axis.
- The y-axis completes the triad and lies in the equatorial plane xy.

This frame is commonly used when focusing on longer-term orbital dynamics while neglecting short-

term variations.

True of Date

The ToD frame is defined by Earth's orientation at a specific moment in time. In this frame:

- The x-axis points toward the vernal equinox.
- The z-axis aligns with Earth's rotational axis at the given time.
- The y-axis completes the triad and lies in the equatorial plane xy.

The ToD frame accounts for precession and nutation, offering an accurate representation of Earth's instantaneous orientation. It is commonly used for applications requiring precise alignment with Earth's current rotational position.

B | Deimos Space

Deimos Space is a leading European aerospace company and part of Elecnor Group, even though recently bought from Indra, a multinational corporation with extensive experience in engineering and technology. Founded in 2001, Deimos Space specializes in providing cutting-edge solutions for space systems and applications. The company is known for its expertise in the design, development, and operation of mission-critical software and engineering systems across various domains of the space sector. Deimos Space stands out for its advanced capabilities in trajectory design and optimization, mission analysis, and ground segment systems, including mission control, satellite operations, and data processing. The company has been involved in several high-profile missions in collaboration with major space agencies such as ESA, contributing to programs in the realm of Earth observation, satellite navigation, and space exploration. Key strengths of Deimos Space include:

- Earth Observation and Remote Sensing: Development of advanced algorithms and processing tools for extracting valuable insights from satellite data.
- Satellite Navigation: Expertise in Global Navigation Satellite Systems (GNSS), including precise orbit determination and timing solutions.
- Space Mission Design: Proficiency in trajectory optimization and mission planning for interplanetary, orbital, and re-entry missions.
- Ground Segment Systems: Comprehensive solutions for ground-based satellite monitoring, control, and data distribution.
- Software Development: High-reliability software for mission-critical applications, including real-time systems and onboard processing.

Deimos Space differentiates itself through its strong emphasis on innovation, customer-driven solu-

tions and a versatile portfolio that spans the entire space value chain. Its multidisciplinary teams and commitment to quality make it a key player in the development of state-of-the-art technologies that address current and future challenges in the aerospace industry. During my internship I had the possibility to collaborate with the EOMA team. My main duties were to develop some mission analysis toolkits that the team could use during their analysis. During my 7-months internship I developed 2 main toolkits:

- `ControlledReEntry` class: A Python class that simulates the sequence of maneuver needed to perform the Controlled Re-Entry strategy
- `TransformationTool` class: A Python class that transforms the state among different definitions (mean keplerian, osculating keplerian and cartesian) and among different frames (ToD, MoD and MEE2000).

This thesis focused on the functionality and development only of the `ControlledReEntry` class, since due to time constraints, the `TransformationTool` could not be finished. In particular, the testing campaign was not finished, and the software was not fully checked. For these reasons it was not discusses in this work. In addition to that I also had the opportunity to collaborate with the team on a real mission, working on the re-Entry of a satellite that for privacy reasons will not be cited. During this collaboration I had the pleasure to discuss the obtained results, debate about solutions to adopt, and collaborate with other department to fulfill the objective of the mission. The next sections explain briefly the software environment in which my project was inserted. In fact, the mission analysis team, even though it can rely over a large variety of softwares, it mainly uses the functionalities of the desEO software, which are embedded in a very precise enviroment.

Software environment

The mission analysis software environment can be divided as follows:

- `SMAT`: A C++ code, whose aim is to generate executables, which are launchable application, covering over 40 possible analyses such as OrbitPropagation, CoverageAnalysis, GroundStationContact, etc...
- `desEO`: A Java coded software that presents as a very user friendly Grafical User Interface (GUI) where the SMAT generated executables can be run in an easy fashion.

- `pydesEO`: An internal Python API that allows to launch the SMAT generated executables directly from Python.

In a few words, the SMAT code represents the core of the environment, where all the physics and formulas are written for all the possible analyses. desEO and pydesEO are simply interfaces that relies on the use of executables to perform simulations. The main difference between the two is simply on the flexibility. In fact pydesEO can be seen as a ”pythonification” of the GUI, having the plus of working in a flexible environment, giving the possibility to perform easy post-processing, parametric analyses or generate new plots.

DesEO

DMS was responsible, under ESA guidelines, to develop a software, called desEO that contained all the main analyses relative to Earth Observation (EO). desEO is a software toolkit to support mission analysis and preliminary system/subsystem design activities of EO missions, aimed at providing a unique instrument for supporting the work of system and mission analysts. desEO has been designed to be used by mission and system engineers throughout all phases of an EO mission (from Phase 0 to Phase E), whenever they need accurate and fast quantitative results to support design trade-offs and assessment analyses [13]. desEO is a tool capable of covering the most common and repetitive analyses that system and mission analysts encounter in their everyday work in EO missions. It currently comprises 40 modules, i.e. applicable analisys, that can be operated via a GUI. The analyses include orbit propagation, attitude computation, atmospheric analyses, coverage analyses, ground station contact analyses, power budget analysis and basic astrodynamics computations (geometric calculations, analytical formulas) and others. The desEO GUI is divided into sections, as shown in Figure B.1.

The content of each area is summarized below, with reference to the numbers in Figure B.1.

- Area 1: it shows the project name, the selected analysis, and contains the scenario items needed for the project, displaying those applicable to the selected analysis. A drop-down menu allows changing the analysis.

- Area 2: it lists all the already executed analyses in the current project, grouped by analysis name.

- Area 3: it lists the history of all the already executed simulations of the analysis selected in Area

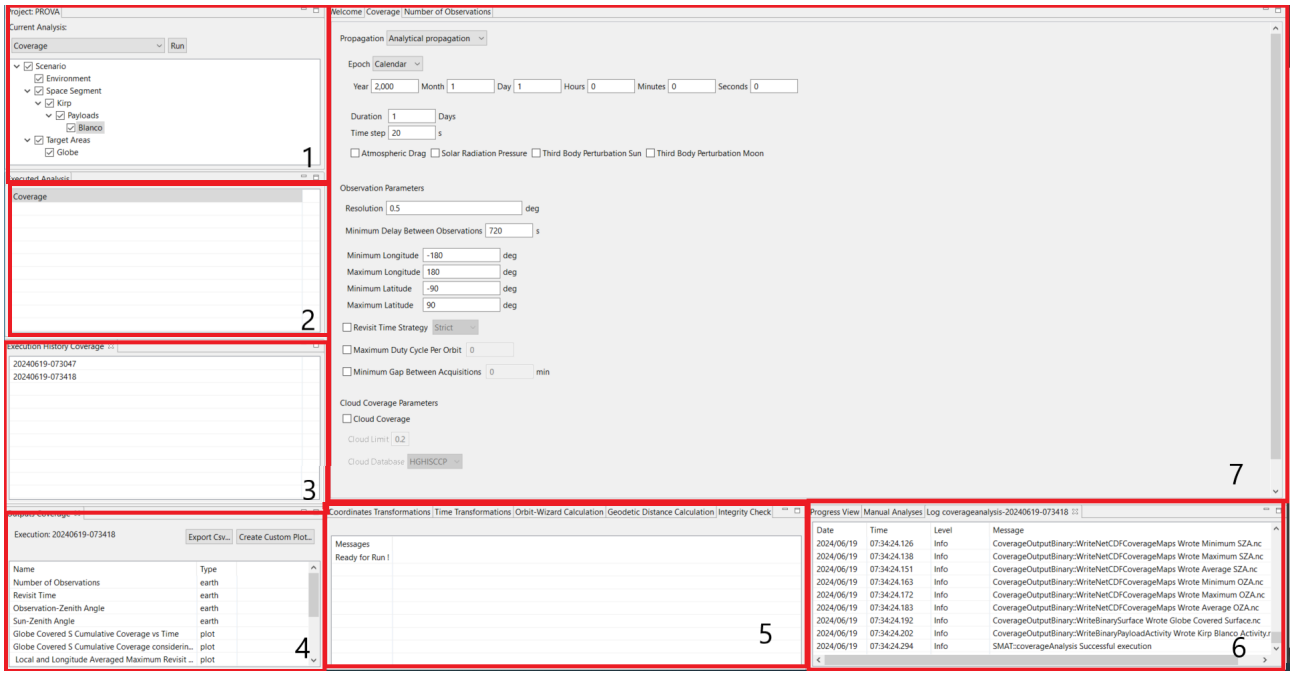

Figure B.1: desEO GUI

2. They are timestamped and each entry can be renamed.

- Area 4: it lists all the available outputs of the simulation selected in Area 3. When double-clicking on one output, the corresponding plot is opened in Area 7. This area also contains a button to export results to CSV format and a button to create user-customized plots.

- Area 5: it contains 4 panels with quick analysis tools (Coordinates Transformations, Time Transformations, Orbit-Wizard calculation, and Geodetic Distance Calculation) and the Integrity Check panel, which shows messages to the user to complete all the required inputs before running a simulation.

- Area 6: it contains 3 panels: a panel with the log of the simulation run, a panel to run an analysis manually, and a panel showing the progress of the simulation.

- Area 7: it contains by default a welcome panel, and upon user request, the input and output panels of the analyses.

SMAT

This software, also internally developed by DMS, is responsable to generate the executable upon which desEO relies on and rapresents the core of every possible analysis. Basically, desEO is simply a user

friendly interface of SMAT. In order to run the executables, an input XML file shall be generated. Those files must have a very precise structure, that must validate against some preset schemas. The idea is to have a general form that works for almost every simulation, but then to add specific nodes relative to the analysis that one wants to perform. In general, the main nodes that one shall specify are:

- Scenario: Defines the setup of the analysis.
- Algorithm: Defines the algorithm chosen.

Inside the scenario node, one shall specify:

- Space segment: Defines the satellite orbits and parameters.
- Environment: Defines the central body characteristics.

Inside the algorithm node, one shall specify a node depending on the analysis that one wants to perform and all the specifics that that analysis requires (e.g. target area for Coverage Analysis, integration time for OrbitPropagation) and no general rule exists.

pyDESEO

Even tough the executables are very strong and widely applicable, they are somewhat limited, not in terms of amount of analysis, but in terms of combinations, parametric analysis and postprocessing. This is why pyDESEO was developed. PyDESEO allows the user to run the executables directly from python. This is achieved by reading the input XML file relative to the analysis that one wants to perform and then giving it as input to a function that launches the right executable. The Python flexibility allows the user to setup iterative and automated routines, in an easy and fast fashion, while maintaing the accuracy of the desEO simulations. In addition to that, Python offers the possibility to build structures, such as classes, upon those executable, enanching the flexibility and possibility of the analysis. Obviously, the toolkits developed during my intership were built according to the library standards. In this way, these added analyses could be directly imported and launched.

Bibliography

[1] JAXA. Satellite lifetime and orbit trivia, 2024.

[2] European Space Agency. Esa's annual space enviroment report, 2024.

[3] IADC. Iadc space debris mitigation guidelines, 2021.

[4] NASA. End of mission considerations, 2015.

[5] Aurelie Bellucci,Jean-François Goester, Deborah Hazan. Swot re-entry: Maneuver strategy and risk computation, 2023.

[6] Anca Maria Stan, Andreea Burlou, Gabriele de Zaiacomo, and Florin Tache. Methodologies and tools for satellite re-entry analysis. In *NMAS*, 2023.

[7] James Cameron Bennett and Jizhang Sang. Modelling the evolution of the low-earth orbit debris population. In *11th Australian Space Science Conference, Canberra, Australia*, 2011.

[8] European Space Agency. Esa space debris mitigation requirements, 2023.

[9] ESA and ESTEC. European code of conduct for space debris mitigation, 2004.

[10] H. Klinkrad. Methods and procedures for re-entry prediction at esa, 2014.

[11] European Space Agency. Guidelines for the computation of delta-v and propellant budget, 2019.

[12] ESA. Re-entry safety requirements, 2017.

[13] F. Letterio, S. Tonetti, S. Cornara, and G. Vicario. desEO – DESIGN ENGINEERING SUITE FOR EARTH OBSERVATION.