Politecnico di Torino

Department of Mechanical and Aerospace Engineering (DIMEAS)

MSc in Aerospace Engineering - Space Orientation

Master Thesis

Development of a mission analysis and trajectory optimization software via ESA GODOT

Supervisor:

Prof. Lorenzo Casalino

Author:

Andrea Musacchio

Co-supervisor:

Ing. Francesco Castellini

Academic Year

2021-2022

Alla famiglia che ho lasciato a Portocannone,

A quella che ho trovato a Torino

”Verso l’infinito... e oltre!” - Buzz Lightyear

Abstract

This thesis' objective is to describe the development process, in collaboration with the start-up AerisGate, of a set of python applications for mission analysis and trajectory optimization by explaining the theory behind these applications and showing the results of tests conducted.

The applications are based on GODOT (General Orbit Determination and Optimisation Toolkit), a European Space Agency flight dynamics software for orbit determination and analysis, that allows to model the universe and perform spacecraft's trajectory propagation. By combining this tool with the optimization library PyGMO, the developed applications aim to be as generic as possible, in order to analyse many types of missions. The two main optimization algorithms applied are SLSQP (Sequential Least Squares Programming) and IPOPT (Interior Point OPTimizer). These are non-linear optimization Local algorithms, thus they aim to find the local minima of the function studied. This mean that an initial guess must be given to the algorithms and it has to be in the space of the solutions, otherwise these algorithms won't converge. To perform the preliminary analysis required to obtain a good initial guess, the Lambert's problem has been solved to obtain a mission's porkchop plots.

To show the capabilities of the applications three test missions have been analysed and successfully optimized: a Mars orbit insertion departing from Earth via B-plane targeting; a free-return trajectory from Earth to Moon; an in-plane station-keeping manoeuvres analysis for Sun-Synchronous orbits via the implementation of a bisection algorithm.

Contents

List of Figures

List of Tables

List of Algorithms

Introduction

The Space industry is a sector in constant growth and evolution. The global space economy rose to 447 billion dollars in 2020, an increase of 55% from 2010 according to The Space Report 2021 Q2, demonstrating the high interest it is receiving.

In this context the idea of the New Space economy take place, where this industry evolves from centralized and exclusive to globalized and private. NewSpace ventures arise, founding startups working in a wide range of space fields, from rockets to satellites constellations, aiming to reinvent the traditional space industry.

Among these participants of the new (space) gold era AerisGate, a Germany-based startup company founded by three ESA-ESOC employees, made room for itself. This thesis born as a collaboration between the candidate and AerisGate to develop a Python-based tool of space mission analysis. By implementing the ESA GODOT software, this works aims to create a set of apps able to analyze virtually any orbit and, in combination with the optimization PyGMO library, optimize spacecrafts' trajectories.

The thesis presents the fundamental theory on which the libraries and apps are based and exposes the tools' functionalities with a set of test cases.

Chapter 1

Theoretical background and Python packages

As one of the businesses of AerisGate is to offer consulting services in the Space Flight Dynamics field for other companies or universities, it may be extremely valuable to have a software robust enough to perform analysis, like spacecraft's trajectory propagation as well as orbit visualization and data plotting, on a generic mission. If the software is also able to optimize such trajectory, we are in front of a convenient tool to speed up the analysis work.

The aim of this thesis is to develop such tool by coding applications in Python programming language. These applications are primarily based on two libraries developed by ESA: GODOT and PyGMO/pagmo. The following sections will describe these two main libraries, both showing some of the theory behind as well as how the software is built. Other Python libraries used during the development of the applications will be cited briefly.

1.1 ESA GODOT flight dynamics software

Figure 1.1: GODOT logo

GODOT (General Orbit Determination and Optimisation Toolkit) [1] is a flight dynamics software developed by ESA/ESOC for orbits analysis and optimization. It is coded in C++ and has a Python interface. The apps in this thesis will use version 0.7.0, but the software is still in development so improved versions of the library are being developed. The software's formulations of light time, relativistic effects and dynamical models are based on the orbit determination technical report Moyer 1971 [2]. The report describes the mathematical models of the Double-Precision Orbit Determination Program (DPODP), which allows the calculations of the required parameters to describe a spacecraft trajectory for lunar and planetary missions based

on the spacecraft's data retrievable from Earth.

As GODOT is made of many sub-libraries, it is worth to focus on the functionalities of the sub-libraries employed in this project.

The main tools are the Universe class, the Trajectory class and the Problem class, created from a configuration file with .json or .yml extension, and the Propagator.

1.1.1 Universe class

The Universe class allows the user to setup the environment of the simulation and it is constructed via various plugins. There is again a vast choice of plugins but, for the sake of brevity, only the one used during this project will be described.

Spacetime system

The first one is the definition of the relativistic system. It is possible to choose between the barycentric (BCRF) or geocentric (GCRF) celestial reference system, both created from the International Astronomical Union (IAU) [3] to analyze orbits respectively far or near Earth. It is possible to convert between these reference systems via the SOFA system [4]. As GODOT can handle various coordinate time scale, based on the choice of reference frame the propagator will use Barycentric Dynamical Time (TDB) or the Terrestrial Time (TT) [5].

Ephemeris

For the determination of planets or asteroids state and physical data the ephemeris plugin is implemented. It takes as inputs kernel files, which are astronomy data built using the information system SPICE, built by the Navigation and Ancillary Information Facility under NASA directions [6]. For this project, the two kernels used are the DE432 for the ephemeris, which is an update of DE430[7], and gm-DE431 to assign mass parameters to planets.

Frames

The frames plugin allows to analyze state vectors with different axes and define single points in space [8]. The default axes are the Inertial Celestial Reference Frame[9] and the Earth Mean eCliptic frame, while the only predefined point is the Solar System Barycentre. There can be user-defined reference axes or points, or built-in ones, as the IAU reference frames [10].

Bodies and Gravity

The bodies plugin allows to choose which bodies will influence the mission and is a baseline for the gravity plugin, which allows to analyze the gravitational acceleration, by choosing the right Center of Integration based on the satellite's position in respect to the sphere of influence of a body.

Atmosphere

The atmosphere plugin allows the modelling of air drag. It is explain in more details in Section 3.1

Dynamics

The dynamics plugin is extremely useful as it is possible to create various dynamic models to use during a trajectory propagation, especially for analysis purpose. For example it is possible to create a reference orbit by disabling Solar Radiation Pressure and air drag or the effect of a planet's spherical harmonics.

There are other useful plugin that has not been used during this thesis, but can be found in GODOT's Wiki page.

Below an example of a Universe file:

```
# UNIVERSE class
---
spacetime:
```

```
  system: BCRS

ephemeris:
  - name: de432
    files:
      - ../de432.jpl
  - name: gm431
    files:
      - ../gm_de431.tpc

frames:
  - name: ephem1
    type: Ephem
    config:
      source: de432
  - name: ITRF
    type: AxesOrient
    config:
      model: IERS2000
      nutation: ../nutation2000A.ipf
      erp: ''
  - name: Mars
    type: AxesOrient
    config:
      model: MarsIAU2009

constants:
  ephemeris:
    - source: gm431

bodies:
  - name: Earth
    point: Earth
    gravity:
      - EarthGrav
    gm: EarthGrav
  - name: Sun
    point: Sun
  - name: Moon
    point: Moon

gravity:
  - name: solarSystem
    bodies:
      - Earth
```

```
      - Moon
      - Sun

dynamics:
  - name: solarSystemGravity
    type: SystemGravity
    config:
      model: solarSystem

  - name: srp
    type: SimpleSRP
    config:
      mass: GeoSat_mass
      area: GeoSat_srp_area
      cr: GeoSat_srp_cr
      occulters:
        - Earth
        - Moon

  - name: solarSystem
    type: Combined
    config:
      - solarSystemGravity
      - srp

spacecraft:
  - name: GeoSat
    mass: 1200 kg
    srp:
      area: 10 m^2
      cr: 1.7
    thrusters:
      - name: main
        thrust: 100 N
        isp: 320 s

sphericalHarmonics:
  - name: EarthGrav
    type: File
    config:
      point: Earth
      degree: 16
      order: 16
      axes: ITRF
      file: ../eigen05c_80_sha.tab
---
```

1.1.2 Trajectory class

The Trajectory class is the other main modelling tool of GODOT. It allows the description of orbital trajectories via four different elements:

Control

Control points allow the user to choose a state vector and an epoch. These are the initial states for the propagator computation and are the main elements from the mission analysis standpoint. These points allows the user to set spacecraft's states as targets for the propagation and optimization of a trajectory.

Manoeuvre

Used to apply accelerations to the spacecraft. Three data are required to describe an impulsive manoeuvre in terms of the direction vector and the magnitude, in $\Delta V\ m/s$, of the engine burn. The reference frame of the resulting vector is also required. In case of finite manoeuvres, the magnitude of the burn is replaced by the time of burn of constant intensity, as the throttle is not implemented in GODOT. The spacecraft can be modelled with multiple propulsion systems that can be implemented during the manoeuvre analysis.

Match

These elements are needed when there is more than one control point to get a continuous trajectory. These are especially useful during the optimization of a trajectory, as the aim of these elements is to match the state of a spacecraft in a given epoch between two control points when a forward and backward propagation are happening simultaneously. This allows to break up the trajectory in segments and perform a multiple shooting method to solve the optimization problem [11].

Point

Can be used to have a reference during the mission analysis or to apply a discontinuity in

the states. It is also used to define the start or end point of a propagation if there are no match, control or manoeuvre points in the desired epoch.

An example of a Trajectory class is the following:

```
# TRAJECTORY class
---
timeline:

  - type: control
    name: launch
    epoch: 0 TDB
    state:
      - name: GeoSat_center
        body: Earth
        axes: ICRF
        dynamics: solarSystem
        value:
          rpe: 6678 km
          rap: 80000 km
          inc: 18 deg
          ran: 0 deg
          aop: 0 deg
          tan: 0 deg
      - name: GeoSat_mass
        value: 1000 kg
      - name: GeoSat_dv
        value: 0 m/s

  - type: manoeuvre
    name: man1
    model: finite
    input: GeoSat
    thruster: main
    config:
      start:
        body: Earth
        tan: 180 deg
      end:
        dv: 1100 m/s
      direction:
        body: Earth
        axes: TCN
        ras: 0 deg
```

```
      dec: 0 deg

- type: match
  name: match1
  input: GeoSat
  point:
    epoch: 1.0 TDB
  body: Earth
  vars: equi

- type: manoeuvre
  name: man2
  model: finite
  input: GeoSat
  thruster: main
  config:
    start:
      body: Earth
      axes: ICRF
      tlo: 0 deg
    end:
      dv: 600 m/s
    direction:
      body: Earth
      axes: TCN
      ras: 180 deg
      dec: 0 deg

- type: control
  name: arrival
  epoch: 2 TDB
  state:
    - name: GeoSat_center
      body: Earth
      axes: ICRF
      dynamics: solarSystem
      value:
        slr: 42165.0 km
        ecx: 0
        ecy: 0
        inx: 0
        iny: 0
        tlo: 180 deg
    - name: GeoSat_mass
      value: 600 kg
```

```
    - name: GeoSat_dv
      value: 1500 m/s

---
```

1.1.3 Problem class

The problem class is the bridge between GODOT and PyGMO. It allows the user to optimize a trajectory by declaring a number of parameters and an optimization's objective.

The parameters are divided in three linked categories:

Free

The free parameters are every trajectory element that can be modified during the optimization process. An example are the magnitudes of the components of a manoeuvre's velocity vector, or the time of a burn, as well as Keplerian orbits' data that are not constrained by the mission.

Bounds

The bounds set the lower and upper limits of the free parameters. This allows to choose a confined space of solutions. Moreover, if the optimization process is based on global algorithms, later explained in Section 1.2, bounds are mandatory for every free parameter, in order to give the algorithm a confined space in which choose a random number.

Scales

The scales are scaling factor applied to the parameters, so that all the parameters are normalized. The scales are useful when working with variables that have a great difference in order of magnitude, to help the local optimization algorithms in finding the proper descent direction.

The **objective** is the value that will be optimized. It is required to express if the aim is to

maximise or minimise it and in which point of the trajectory we are interested. For the majority of the mission the aim is to maximise the mass, or minimise the ΔV, of the satellite at the end of a trajectory, which means saving fuel and thus reducing the mission's costs. The output of a problem optimization is the optimized trajectory file.

An example of a Problem file can be found below:

```
# PROBLEM class
---
parameters:
    free:
    - man1_start_tan
    - man1_ras
    - man1_dec
    - man1_end_dv

    - man2_start_tlo
    - man2_ras
    - man2_dec
    - man2_end_dv

    - arrival_GeoSat_mass
    - arrival_GeoSat_dv

    bounds:
      man1_start_tan:[0 deg, 360 deg]
      man1_ras:[-15 deg, 45 deg]
      man1_dec:[-20 deg, 10 deg]
      man1_end_dv:[0 m/s, 5000 m/s]

      man2_start_tlo:[0 deg, 40 deg]
      man2_ras:[150 deg, 210 deg]
      man2_dec:[-5 deg, 5 deg]
      man2_end_dv:[0 m/s, 1000 m/s]

      arrival_GeoSat_mass:[0 kg, 1000 kg]
      arrival_GeoSat_dv:[0 m/s, 2000 m/s]

    scales:
      man1_start_tan: 1 deg
      man1_ras: 1 deg
      man1_dec: 1 deg
      man1_end_dv: 1000 m/s
```

```
        man2_start_tlo: 1 deg
        man2_ras: 1 deg
        man2_dec: 1 deg
        man2_end_dv: 100 m/s

        arrival_GeoSat_mass: 100 kg
        arrival_GeoSat_dv: 100 m/s

objective:
    type: maximise
    value: GeoSat_mass
    point: arrival
---
```

1.1.4 Propagator class

The propagator class is responsible to evaluate the state vector of the spacecraft in a given time range, considering the environment modelled with the Universe class and the states and manoeuvres from the Trajectory class. There are two available integration algorithm in GODOT: Runge-Kutta and Adams, although for the app proposed in the thesis only Runge-Kutta has been used.

1.1.4.1 Seventh-order Runge-Kutta formula

The Runge-Kutta method is an iterative algorithm used to approximately solve systems of differential equations by using discretization to calculate the solutions in small stepsize. There are many variants of the method. In GODOT the Runge–Kutta–Verner method of seventh-order (RK78) [12, 13] with a 7-th order interpolating polynomial [14] is implemented. It is an explicit method, which means that the solution of the step n+1 is obtained via the solution of the step n. To shorten the explanation, we will consider for this analysis only one differential equation, but the formulas can be used for systems of differential equations.

To begin the computation, we need an initial condition (1.1) and the differential equation (1.2):

$$y(t_0) = y_0 \tag{1.1}$$

$$y' = f(t, y) \tag{1.2}$$

where y is a vector function of time t that we want to approximate and y', its derivative, is a function of y and t. We then can write the Runge-Kutta formula:

$$\begin{aligned} f_0 &= f(t_0, y_0) \\ f_k &= f(t_0 + \alpha_k h, y_0 + h \sum_{\lambda=0}^{k-1} \beta_{k\lambda} f_\lambda) \;\; (k = 1, 2, ..., 12) \end{aligned} \tag{1.3}$$

and

$$\begin{aligned} y &= y_0 + h \sum_{k=0}^{10} c_k f_k \quad + O(h^8) \\ \hat{y} &= y_0 + h \sum_{k=0}^{12} \hat{c}_k f_k \quad + O(h^9) \end{aligned} \tag{1.4}$$

where h is the integration stepsize. The other coefficients $\alpha_k, \beta_{k\lambda}, c_k, \hat{c}_k$ have to be determined so that equations in (1.4) represent respectively a seventh- and an eight-order Runge-Kutta formula. It is possible to evaluate these therms as they have to satisfy certain equations of conditions listed in [15]. For further reading the technical report [12], as well as [13], describe in great detail the solution of the equations of condition for these coefficients.

By having two different order formulas in equations (1.4) it is possible to perform a step-size control [16], thus not requiring a fixed step-size, as the difference $y - \hat{y}$ provides an approximation of the leading eight-order truncation error of the seventh-order Runge-Kutta formula. After defining $\Delta toll = toll_{abs} + toll_{rel} * \mid y \mid$, which allows the user to control the tolerance based on the analysis, the control can be implemented: if the error is higher than $\Delta toll$ we can halve the step size and redo the step, if it is smaller than $(\frac{1}{2})^8 \Delta toll$ we can double the step size, thus

gaining on computational time.

1.2 PyGMO Parallel Global Multiobjective Optimizer

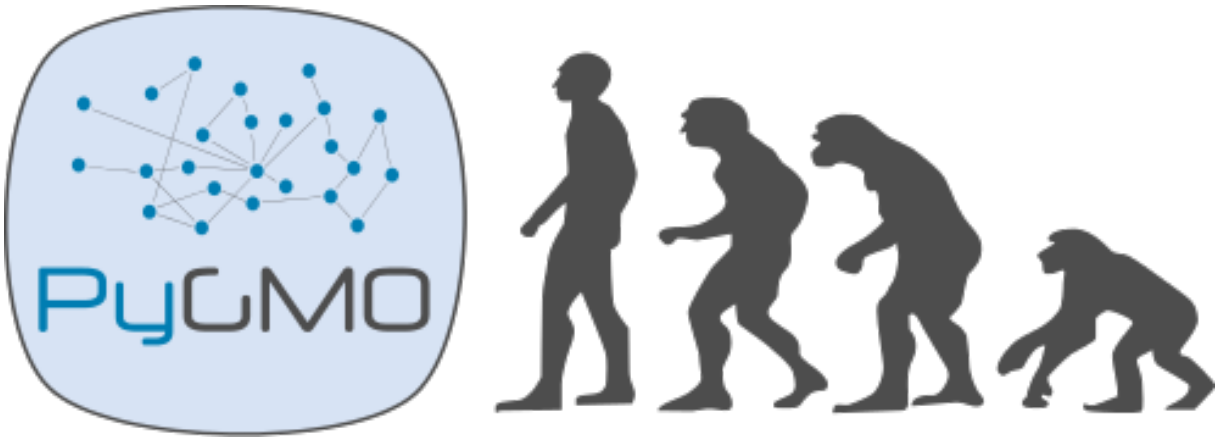


Figure 1.2: Pygmo/pagmo logo

Pagmo [17] is an optimization library written in C++ programming language, transpose in Python language as PyGMO. It implements a vast array of optimization algorithms, both local and global.

Global optimization algorithms [18] aim to find the global solution of a function, usually by means different than analytical methods. The global algorithms use a set of, often random, initial guesses, called population, and function to evaluate the goodness of the population, called fitness. The fittest guesses, or individuals, of the population get selected and from them, in a fashion dependent by the algorithm, by mixing or mutating a new population i generated. The generation of a new population is called evolution. The evident underlying theme is the parallelism with the evolutionary theory of natural selection by Charles Darwin [19]. This jargon is used throughout PyGMO also for local algorithms.

Local optimization algorithms find the optimal solution in a specific region of the search space, thus finding the local minima or maxima of a function given a single initial guess of the

solution.

The applications developed for this thesis use the local optimization algorithms present in PyGMO, in particular SLSQP and IPOPT. For each optimization both are applied and the solution that better maximise/minimise our objective is picked. A brief presentation of both algorithms is presented in the following sections. Also the COBYLA algorithm is presented, as it has been analyzed during the evaluation of algorithms in PyGMO, but ultimately discarded due to poor performances.

1.2.1 SLSQP

The Sequential Least-Squares Quadratic Programming (SLSQP) [20, 21] algorithm is part of the Non-Linear optimization algorithms of PyGMO [22]. Sequential Quadratic Programming is an optimization method based on the generation of steps by solving quadratic subproblems. In order to shortly explain the algorithm, the procedure from [23] will be presented.

We shall start by stating the problem. A problem is made up of an objective function $f(x)$, of which we want to find the minimum value, and, in case of constrained problems, of equality and inequality constraints (1.5)

$$\begin{aligned} &min\ f(x) \\ &h(x) = 0 \\ &g(x) \leq 0 \end{aligned} \tag{1.5}$$

where x may be a vector of many variables, so $h(x)$ and $g(x)$ can be systems, and by nonlinear problems we mean that any of these equations may be nonlinear.

It is beneficial to reformulate the problem via a Lagrangian function, which combines the equations in (1.5) into one, using the Lagrangian multipliers λ and μ for equality and inequality constraints respectively:

$$L(x, \lambda, \mu) = f(x) + \sum_i \lambda_i h_i(x) + \sum_i \mu_i g_i(x) \tag{1.6}$$

The local minimum of the function can be found by searching the values so that the gradient is zero. It is possible to present the resultant equation via the Karush-Kuhn-Tucker (KKT) Conditions:

$$\nabla L = \begin{bmatrix} \frac{dL}{dx} \\ \frac{dL}{d\lambda} \\ \frac{dL}{d\mu} \end{bmatrix} = \begin{bmatrix} \nabla f + \lambda \nabla h + \mu \nabla g^* \\ h \\ g^* \end{bmatrix} = 0 \tag{1.7}$$

The third condition, g^*, represent the inequality constraints that are active, as the ones not near the optimal solution are not relevant; μ is zero for inactive constraints, as the Lagrange multipliers describe the change in the objective function with respect to a change in a constraint.

To effectively tackle nonlinar functions, the Newton's method is implemented to improve the guess value. The idea is to take into consideration the rate of change of the function and how the function is accelerating at the guess. The improvement steps are then the following:

$$x_{k+1} = x_k - \frac{\nabla f}{\nabla^2 f} \tag{1.8}$$

We can now take advantage of the Lagrange formulation to rewrite the iteration:

$$\begin{bmatrix} x_{k+1} \\ \lambda_{k+1} \\ \mu_{k+1} \end{bmatrix} = \begin{bmatrix} x_k \\ \lambda_k \\ \mu_k \end{bmatrix} - \nabla^2 L_k^{-1} \nabla L_k \tag{1.9}$$

where ∇L_k is the label KKT conditions in (1.7), while $\nabla^2 L_k$ can be written as:

$$\nabla^2 L_k = \begin{bmatrix} \nabla^2_{xx} L_k & \nabla h & \nabla g \\ \nabla h & 0 & 0 \\ \nabla g & 0 & 0 \end{bmatrix} \tag{1.10}$$

This way in theory there are no nonlinear equations to solve. In practice the divergence may not be an invertible matrix, thus the improvement direction of the Newton's Method is found by solving with quadratic algorithms a quadratic minimization sub-problem:

$$p = \frac{\nabla L}{\nabla^2 L} = \frac{(\nabla L)p}{(\nabla^2 L)p} \tag{1.11}$$

where p is the increment. By noticing the resemblance with a two-term Taylor Series, it is possible to obtain a minimization sub-problem by decomposing the equations within this system and cutting the second order term in half to match Taylor Series concepts. Although this sub-problem is quadratic, is easier to solve than our initial problem, as it has only one variable:

$$\begin{aligned} min(p) \;\; & f_k(x) + \nabla f_k^T p + \frac{1}{2} p^t \nabla^2_{xx} L_k p \\ & \nabla h_k p + h_k = 0 \\ & \nabla g_k p + g_k = 0 \end{aligned} \tag{1.12}$$

1.2.2 IPOPT

The Interior Point OPTimizer (IPOPT) solver is a software package for large-scale nonlinear optimization. In particular, it uses a primal-dual interior-point algorithm with a filter line-search method. The paper of the software's authors [24] fully describe the method that will be explained below. For coherence, some variables will be renamed to comply with the notation used in Section 1.2.1.

As before, we can state the problem's formulation:

$$\begin{aligned} min\ f(x) \\ h(x) = 0 \\ x_L \leq x \leq x_U \end{aligned} \tag{1.13}$$

Differently from the problem in (1.5), here we have the lower and upper bounds of the variables: x_L, x_U. Although the inequality constraints aren't directly in the equation, these can be added by introducing slack variables.

Interior points methods are also referred as barrier methods, as the method can be generalized using barrier functions, that are continuous functions whose values increase to infinity as the point approaches the boundary of the feasible region. The algorithm evaluates the solutions for a sequence of barrier problems:

$$\begin{aligned} min\ \phi_\beta(x) := f(x) - \beta \sum_{i=1}^{n} ln(x^{(i)}) \\ h(x) = 0 \end{aligned} \tag{1.14}$$

where β is a barrier parameter converging to zero. We can reforulate the equation in (1.14) as applying a homotopy method to the primal-dual equations:

$$\begin{aligned} \nabla f(x) + \nabla h(x)\lambda - z = 0 \\ h(x) = 0 \\ diag(x)diag(z)e - \beta e = 0 \end{aligned} \tag{1.15}$$

again with β driven to zero. Here z is the Lagrangian multiplier for the bounds constraints and e is a vector of all ones. If we consider in equation (1.15) $\beta = 0$ and $x, z \geq 0$ we remain again with the KKT conditions for the original problem. The optimality error can be defined as:

$$E_\beta(x,\lambda,z) := max\{\frac{\|\nabla f(x)+\nabla h(x)\lambda - z\|_\infty}{s_d}, \|h(x)\|_\infty, \frac{\|diag(x)diag(z)e-\beta e\|_\infty}{s_c}\} \tag{1.16}$$

where $s_d, s_c \geq 1$ are scaling parameters. We consider as the optimality error of the original problem $E_0(x,\lambda,z)$ for $\beta = 0$. The algorithm ends if a solution $(\tilde{x}_*, \tilde{\lambda}_*, \tilde{z}_*)$ satisfy:

$$E_0(\tilde{x}_*, \tilde{\lambda}_*, \tilde{z}_*) \leq \epsilon_{tol} \tag{1.17}$$

where ϵ_{tol} is a user defined tolerance.

The definition of s_d, s_c and the optimization for a faster local convergence are explained in the before referenced paper [24].

To solve the barrier problem in (1.14) for a fixed value of β_j it is again used a Newton's method, this time damped, to the equations (1.15). We can write:

$$\begin{bmatrix} \nabla^2_{xx}L_k & \nabla h(x_k) & -I \\ \nabla h(x_k)^T & 0 & 0 \\ \nabla diag(z_k) & 0 & diag(x_k) \end{bmatrix} \begin{bmatrix} d^x_k \\ d^\lambda_k \\ d^z_k \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) + \nabla h(x_k)\lambda_k - z_k \\ h(x_k) \\ diag(x_k)diag(z_k)e - \beta_j e \end{bmatrix} \tag{1.18}$$

where L_k is the Lagrangian function

$$L(x_k, \lambda_k, z_k) = f(x_k) + h(x_k)^T\lambda - z_k \tag{1.19}$$

and $(d^x_k, d^\lambda_k, d^z_k)$ are the search directions.

The method first evaluate the smaller symmetric linear system

$$\begin{bmatrix} \nabla^2_{xx} + diag(x_k)^{-1} diag(z_k) & \nabla h(x_k) \\ \nabla h(x_k)^T & 0 \end{bmatrix} \begin{bmatrix} d^x_k \\ d^\lambda_k \end{bmatrix} = - \begin{bmatrix} \nabla \phi_{\beta_j}(x_k) + \nabla h(x_k)\lambda_k \\ h(x_k) \end{bmatrix} \tag{1.20}$$

and then evaluates the last search vector

$$d^z_k = \beta_j diag(x_k)^{-1} e - z_k - diag(x_k)^{-1} diag(z_k) d^x_k \tag{1.21}$$

The equation (1.20) is modified to ensure that the matrix in the top-left is positive definite and that $diag(x_k)$ is full rank:

$$\begin{bmatrix} \nabla^2_{xx} + diag(x_k)^{-1} diag(z_k) + \delta_w I & \nabla h(x_k) \\ \nabla h(x_k)^T & \delta_c I \end{bmatrix} \begin{bmatrix} d^x_k \\ d^\lambda_k \end{bmatrix} = - \begin{bmatrix} \nabla \phi_{\beta_j}(x_k) + \nabla h(x_k)\lambda_k \\ h(x_k) \end{bmatrix} \tag{1.22}$$

where $\delta_w, \delta_c \geq 0$ are scalars.

Once the search directions are computed from (1.21) and (1.22), in order to obtain the next iterate we need to evaluate the step-size $\alpha_k, \alpha^z_k \in (0, 1]$:

$$\begin{aligned} x_{k+1} &:= x_k + \alpha_k + d^x_k \\ \lambda_{k+1} &:= \lambda_k + \alpha_k + d^\lambda_k \\ z_{k+1} &:= z_k + \alpha^z_k + d^z_k \end{aligned} \tag{1.23}$$

The different stepsize in the z variable is used as it is more efficient. This property is valid for any iterate, as in case of an optimal solution of the barrier problem (1.14) both x and z are positive. The stepsize α_k is obtained via a backtracking line-search filter method procedure [25] that explore a sequence of trial stepsize

$$\alpha_{k,l} = 2^{-l}\alpha_k^{max} \quad (l = 0, 1, 2, ...) \tag{1.24}$$

to ensure global convergence.

A complete implementation of the method is presented in the Algorithm A (Line-search Filter Barrier Method) of the paper [24].

1.2.3 COBYLA

Constrained optimization by linear approximation (COBYLA) [26] is another optimization method useful to solve constrained problems. Its advantage is the ability to perform the optimization of an objective function without knowing the function's derivative or, for a multivariable function, its gradient. The main disadvantage lies behind its slowness with respect to the other presented algorithms that use the gradient of the objective function. During test runs it has been found that the time required for an optimization with COBYLA is almost twice as much compared to the time spent for the same analysis with IPOPT or SLSQP. As GODOT automatically evaluate the gradient of the objective function, the advantage of COBYLA become irrelevant, so it has not been used. The main concepts of this algorithm are presented anyway as it may be useful for future works.

To sum up the algorithm, it approximate the constrained optimization problem with linear programming problems at each iteration. The linear programming problem is solved and a candidate for the optimal solution is obtained. This candidate is then evaluated with the real objective and constraints functions, generating a new point in the optimization space. Then a new iteration can be performed with the new data, so that there is an improvement in the approximating linear programming problem. The stepsize is reduced when no improvement is achieved. Once the stepsize is smaller than a defined value, the algorithm ends. Below the iteration process is described.

As always, the problem is presented

$$\begin{aligned} &min\ f(x) \\ &g(x) \geq 0 \end{aligned} \tag{1.25}$$

where x can be a vector and $g(x)$ can be a system, and we state that $x \in \mathbb{R}^n$. Then we borrow the idea of using the vertices $x^{(j)} : j = 0, 1, ..., n$ of a nondegenerate simplex in $\mathbb{R}^n$ from the simplex method presented in [27] to find the next vector of variables. We also approximate the problem with the linear programming problem

$$\begin{aligned} &min\ \hat{f}(x) \\ &\hat{g}(x) \geq 0 \end{aligned} \tag{1.26}$$

where $\hat{f}$ and $\hat{g}$ represent linear functions that interpolate $f(x)$ and $g(x)$ at the vertices of the simplex. We can now solve the linear programming problem, which is simple computationally speaking. The vertex $x^{(l)}$ of the simplex, so that $f(x^{(l)}) \geq f(x^{(j)} : j = 0, 1, ...n; j \neq l$, is the one at which the objective function is worst. This vertex is replaced by the solution found x_{new} of the linear problem at each iteration.

Some considerations have been made to improve the algorithm. A trust region bound is set, in which the variables can change, as it solve the fact that the linear problem (1.26) may have no finite solution. Moreover, the trust region radius is reduced when no improvement to the objective function is detected, until it reaches a value set by the user and the algorithm end. Another clever idea is to promote the improvement of the shape of the simplex during the iterations, rather than blindly improving the solution, with the aid of the merit function

$$\Phi(x) = f(x) + \mu[max\{-g(x)\}]_+,\ x \in \mathbb{R}^n \tag{1.27}$$

that is used to compare the goodness of two vectors of variables. In this equation μ is automati-

cally adjusted during the iterations, while the $+$ subscript implies that what is inside the square brackets is replaced by zero if the value is negative, so in this case we are left with $\Phi(x) = f(x)$ when x is feasible. A vector of solution x_a is better than another x_b when $\Phi(x_a) \leq \Phi(x_b)$.

1.3 Other tools

Here for completeness a list of the other python libraries and apps used is presented.

Python libraries:

Matplotlib

Matplotlib [28] is a 2D graphics package used for Python for application development, interactive scripting, and image generation. All the plots generated by the candidate of this thesis derive from matplotlib code.

Numpy

Numpy [29] is a package for scientific computing, especially for array managing and matrix calculations.

Scipy

Scipy [30] is a package of algorithms for scientific computing. It has been used for data interpolation.

Yaml and Json

Are two human readable data-representation languages used for inputting the universe, trajectory and problem data in GODOT.

There are also some tools used based on the NAIF SPICE kernels explained in Section 1.1.1:

brief

Is an utility program to get a data summary of an SPK file.

mkspk

Another utilty program, this time used to create SPK file from the data obtained from GODOT trajectory propagation.

Cosmographia

Is a 3D visualization tool to visualize trajectories using SPICE data as input. It is possible to program a Python script to perform videos of the trajectories in a chosen reference frame.

Chapter 2

Trajectory optimization application

In the field of mission analysis, orbit optimization is crucial to minimise the required ΔV for a trajectory, as it defines heavily a satellite's design. So the tool presented in this section has been developed to easily perform trajectory analysis and ΔV optimization for a generic mission.

The application takes as inputs a Universe, Trajectory and Problem files as well as a configuration file, all in .json or .yaml format. The configuration file allows the user to choose between the algorithms cited in Section 1.2 and set a custom tolerance for each constraint of the problem. It also allows to set a dispersion of the initial state vector, with some degree of randomness, which is useful to represent the uncertainty of a real case scenario and to perform a Monte Carlo analysis of the mission, later explained in Section 2.2. The outputs trajectories can then be analyzed and plotted combining GODOT and Matplotlib libraries.

To evaluate the tool's effectiveness, two test are presented: an interplanetary trajectory from Earth to Mars and a Lunar free-return trajectory.

2.1 Interplanetary trajectory with orbit circularization

An analysis of an interplanetary mission has been conducted to test the capability of the optimization app. The proposed mission is a Mars orbit insertion from an orbit around Earth via a B-plane targeting.

In order to optimize the trajectory, it is mandatory to first define the desired final orbit, as well as the constraints and some spacecraft's data. The departure data are chosen so that we don't have to worry about the launch phase from Earth's surface, while the arrival orbit can be considered as a starting point for any Mars' observation mission.

Desired arrival Keplerian orbit data:

Semy-mayor axis : $8\,000\ km$

Eccentricity : 0

Inclination : $0\ deg$

Arrival time : January 2021

The constraints are the initial orbit data of the satellite around Earth:

Semy-mayor axis : $6678\ km$

Eccentricity : 0

Right ascension angle : $0\ deg$

Argument of periapsis : 115

For what concerns the satellite, the initial data are the following:

Mass : $30\,000\ kg$

Specific impulse : $320\ s$

As stated in Section 1.2, local optimization algorithms are used, which means an initial guess must be given. The guess has to be reasonable, as otherwise the risk of getting out of the space of the solution, thus having the algorithm to not converge, arise. To find a decent initial guess, the steps have been the following:

1. Plot a porkchop plot by solving the Lambert's problem
2. Analyze a departure from Earth to obtain a Mars B-plane targeting
3. Optimize the B-plane targeting to match the Mars orbit circularization
4. Optimize the whole orbit

These steps will be described in the next sections.

2.1.1 Lambert's problem and porkchop plots

The Lambert's problem [31] is a boundary-value problem that allows to determine an orbit from two position vectors and an elapsed time. It is a boundary-value problem as only some of the states are known at different amount of time. In this case we know the initial and final position of the spacecraft, $r1$ and $r2$ in Figure 2.1 at two set times, but we don't know for example the velocity in these two times.

The Lambert's theorem states that the time of flight t required to get from $r1$ to $r2$ with a Keplerian orbit is a function of the sum $r1 + r2$, of the length of the chord c that connect these two point, and of the orbit's semi-major axis. It is based on the formulation of the two body problem when one of the mass is infinitesimal:

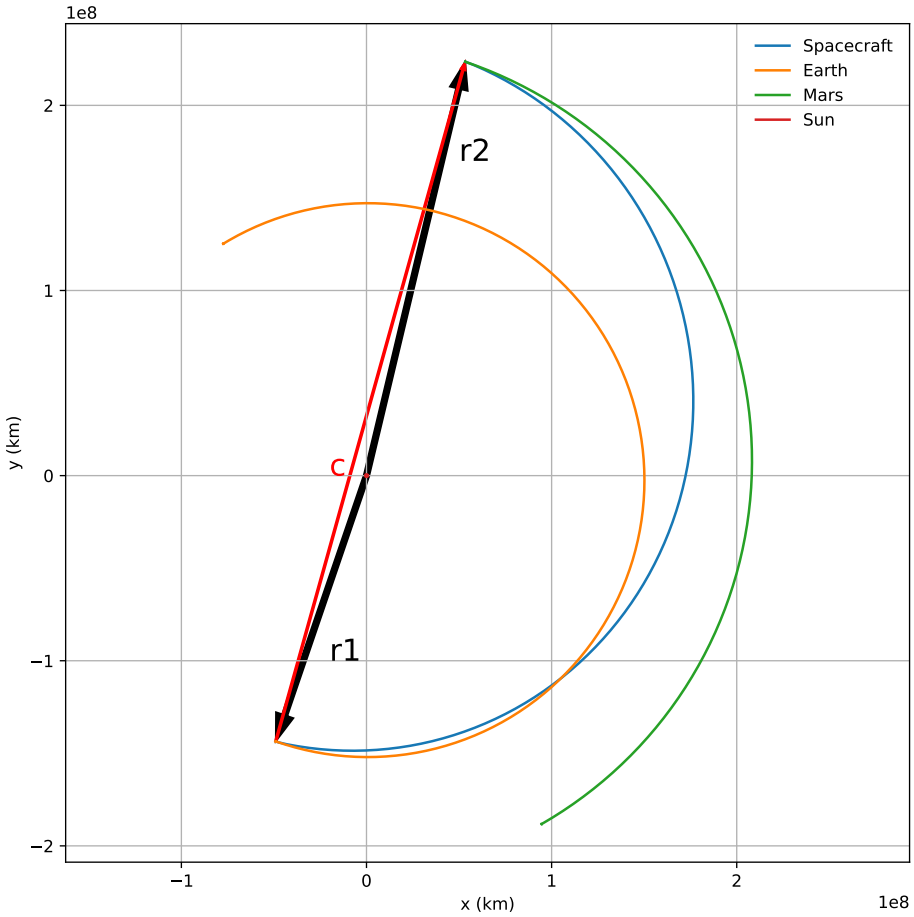


Figure 2.1: Representation of the Lambert's problem with the position state $r1$ and $r2$ and chord c

$$\mathbf{v} = \ddot{\mathbf{r}} = -\mu \frac{\mathbf{r}}{r^3} \tag{2.1}$$

where μ is the gravitational parameter, $\mathbf{v}$ and $\mathbf{r}$ are the position and velocity vector respectively, while r is the position's magnitude.

For any two body orbit we have:

$$\begin{aligned} \mathbf{r} &= F\mathbf{r_0} + G\mathbf{v_0} \\ \mathbf{v} &= \dot{F}\mathbf{r_0} + \dot{G}\mathbf{v_0} \end{aligned} \tag{2.2}$$

where $\mathbf{r_0}, \mathbf{v_0}$ are position and velocity vectors at time t_0 and $\mathbf{r}, \mathbf{v}$ are the vectors at time t.

The coefficients F, G are functions of $\Delta t = t - t_0$ called Lagrange functions, while $\dot{F}, \dot{G}$ are the time derivatives so that

$$F\dot{G} - \dot{F}G = 1 \tag{2.3}$$

The solutions to the Lambert's problem, vectors $\mathbf{v}$ and $\mathbf{v_0}$, can be used to plot the so called porkchop plots, which summarizes the required ΔV for an interplanetary trajectory having as inputs the departure time from the starting planet to the arrival time to the target planet, as well as the position of the planets.

In order to solve the problem, the algorithms exposed in [32] have been implemented in Python. The outputs are plotted in Figure 2.2. In the two figure the x and y axes represent the departure dates, between 01/05/2020 and 01/08/2020, and the arrival dates, between 01/12/2020 and 01/02/2021 respectively.

In Figure (a) the cyan lines represent constant travel times, every trajectory along these lines have the same travel time from Earth to Mars. The magenta lines represent the outgoing C3 from Earth, where C3 is the square of the excess velocity above the escape velocity from Earth, while the blue lines represent how much excess velocity the spacecraft has at Mars arrival.

In Figure (b), the cyan lines represent again the time of flight, while the blue-to-yellow

gradient lines represent contours of equal total ΔV for Earth to Mars trajectories.

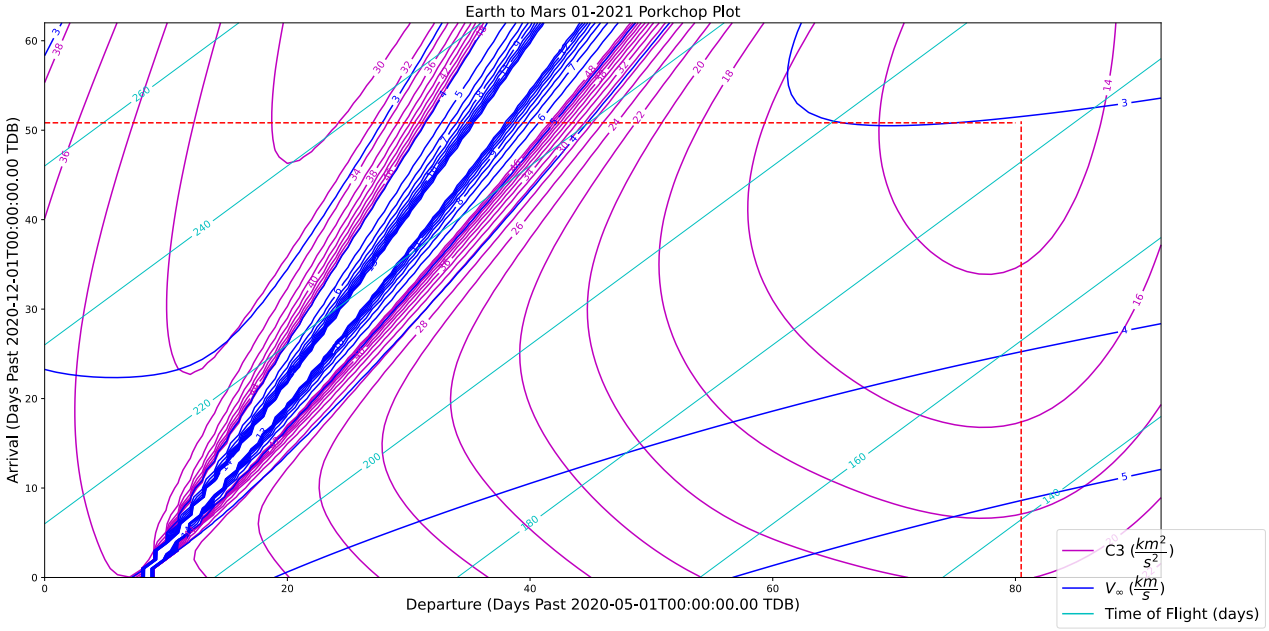
There is a defined gap in the plots. It separates trajectories that have a change in true anomaly of less than 180 degrees, on the right, to the ones with a change of more than 180 degrees on the left, respectively called "Type I" and "Type II" Mars transfer trajectories.

The red dotted line represent our initial guess. As we want our spacecraft to arrive at January 2021, from Figure(b) we can see that the best departure time for minimum ΔV is around 07/2020.

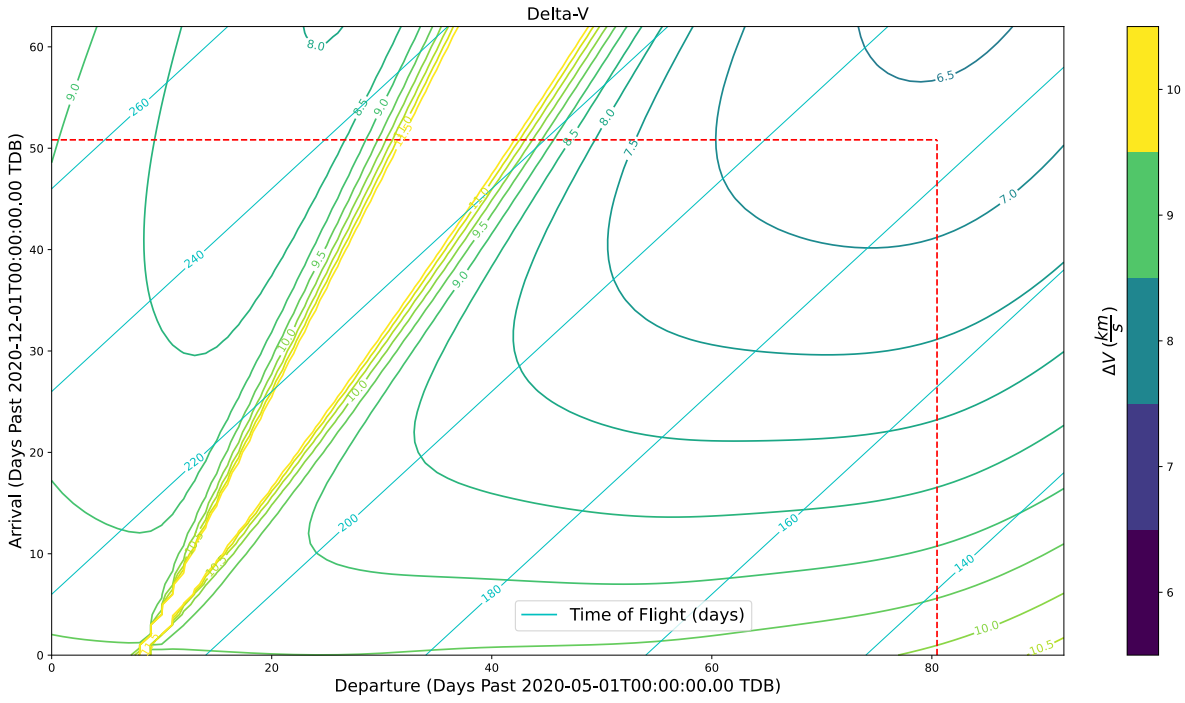
To sum up, our initial guesses are:

Departure date : 20/07/2020

Required ΔV : 7 km/s



(a) C3 and V_{inf} plot



(b) ΔV plot

Figure 2.2: Porkchop plots

2.1.2 Earth departure and B-plane targeting

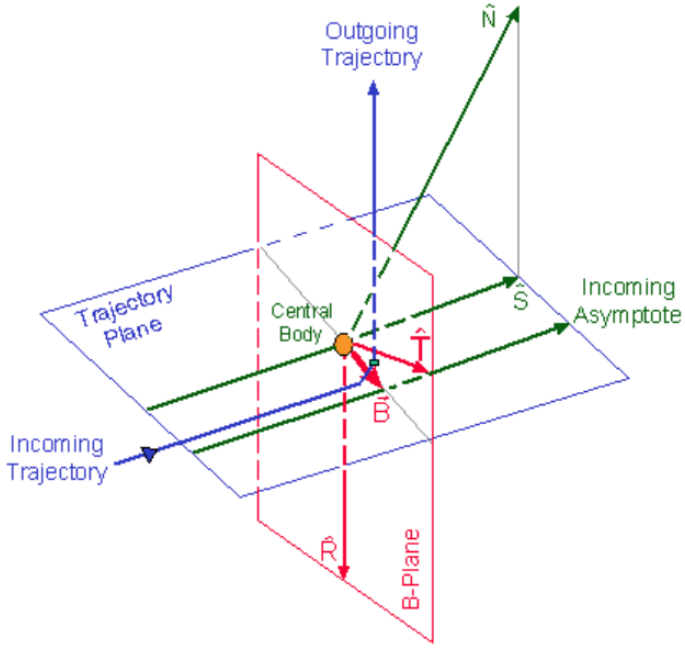
The first part of the trajectory analyzed is a Mars B-plane targeting from an orbit around Earth. The B-plane is a plane orthogonal to the plane of an hyperbolic trajectory and the initial hyperbolic excess velocity vector. We will use the notation presented in Figure 2.3. We will call the reference system on which the B-plane is defined MarsLocal. This frame consist of: x along position vector (radial) from Sun to Mars; y that completes right-hand-side system (pointing positive along the tangential direction) and z along orbital momentum vector

We aim to get an orbit around Mars with an $inc = 0\ deg$ and a $sma = 8000\ km$. This means that we are going to target the B-plane so that $\theta = 0\ deg$ and $B > 8000\ km$, where B is the vector of the intersection between the incoming asymptote and the B-plane. In particular, the last equation is required as the periapsis of the hyperbolic trajectory will be smaller than B. As a guess we are going to use $B = 11750\ km$. From the porkchop plot in Figure 2.2(a) we see that the excess velocity is $V_\infty \sim= 3km/s$, and this value will be used as initial guess for the velocity of the incoming trajectory.

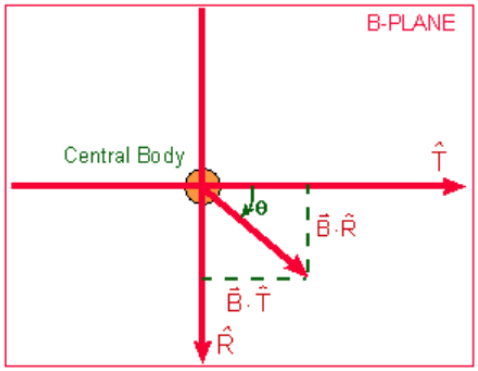
For the Earth departure the constraints have already been presented at the end of Section 2.1, while the initial guess for the departure has been explained on Section 2.1.1.

Two impulsive manoeuvres will be implemented, one to enter the escape trajectory from Earth to Mars and one to correct the trajectory during the cruise phase.

The frame used to define the first manoeuvre is the TCN orbital frame direction, where T is in the direction of the velocity vector (tangential), C is in the direction of the orbital momentum vector (pos x vel) while N completes the TCN system (T x C). We will define the manoeuvre with ΔV, right ascension ras and declination dec. We will use as guess value $\Delta V = 3650m/s$, again based on Figure 2.2(a). The guess values of ras, dec are chosen after propagating the trajectory with different burn directions until a reasonable guess has been found. The point in the orbit around Earth in which the burn happens will be found by the optimization algorithm using the true anomaly angle of the orbit as free parameter.



(a) Side view



(b) Front view

Figure 2.3: B-plane visualization [33]

The second manoeuvre will be expressed using the ICRF frame and as values the three component of the velocity vector dv_x, dv_y, dv_z.

The guess values are summarized here:

Earth orbit:

- sma: 6678 km
- ecc: 0
- inc: 18 deg
- ran: 0 deg
- aop: 115 deg
- tan: 112 deg

Earth escape manoeuvre:

- departure time: 2020-07-20T00:10:00 TDB
- dv: 3650 m/s
- ras: -12 deg
- dec: 3 deg

Cruise control manoeuvre:

- 2020-10-12T00:00:00 TDB
- dv_x: 0 m/s
- dv_y: 0 m/s

- dv_z: 0 m/s

Mars B-plane targeting:

- Arrival date: 2021-01-20T09:31:00.000 TDB
- V_{in} : 3 km/s
- $B * T$: 11750 km
- $B * R$: 0 km

In Figure 2.4 we can see that, beside the discontinuity due to the missing match, the guess is decent.

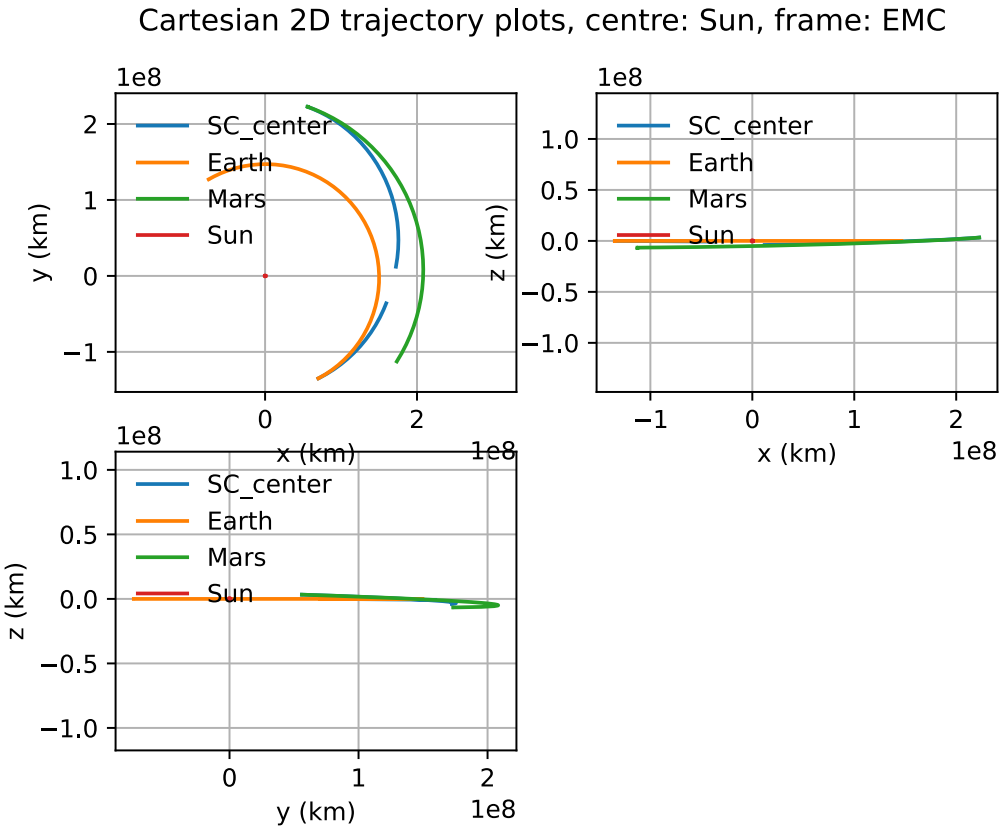


Figure 2.4: Earth to B-plane, propagation of the first guess

After optimizing via the SLSQP algorithm, the solution found is the following:

Earth orbit:

- sma: 6678 km
- ecc: 0
- inc: -6.864 deg
- ran: 0 deg
- aop: 115 deg
- tan: 132.663 deg

Earth escape manoeuvre:

- departure time: 2020-04-23T00:52:33.219 TDB
- dv: 4885.732 m/s
- ras: -9.10 deg
- dec: -0.07 deg

Cruise control manoeuvre:

- 2020-10-12T00:00:00 TDB
- dv_x: 63.583 m/s
- dv_y: 120.522 m/s
- dv_z: -277.802 m/s

Mars B-plane targeting:

- Arrival date: 2020-12-31T03:26:26.829 TDB
- V_{in} : 3 km/s
- $B * T$: 11750 km
- $B * R$: 0 km

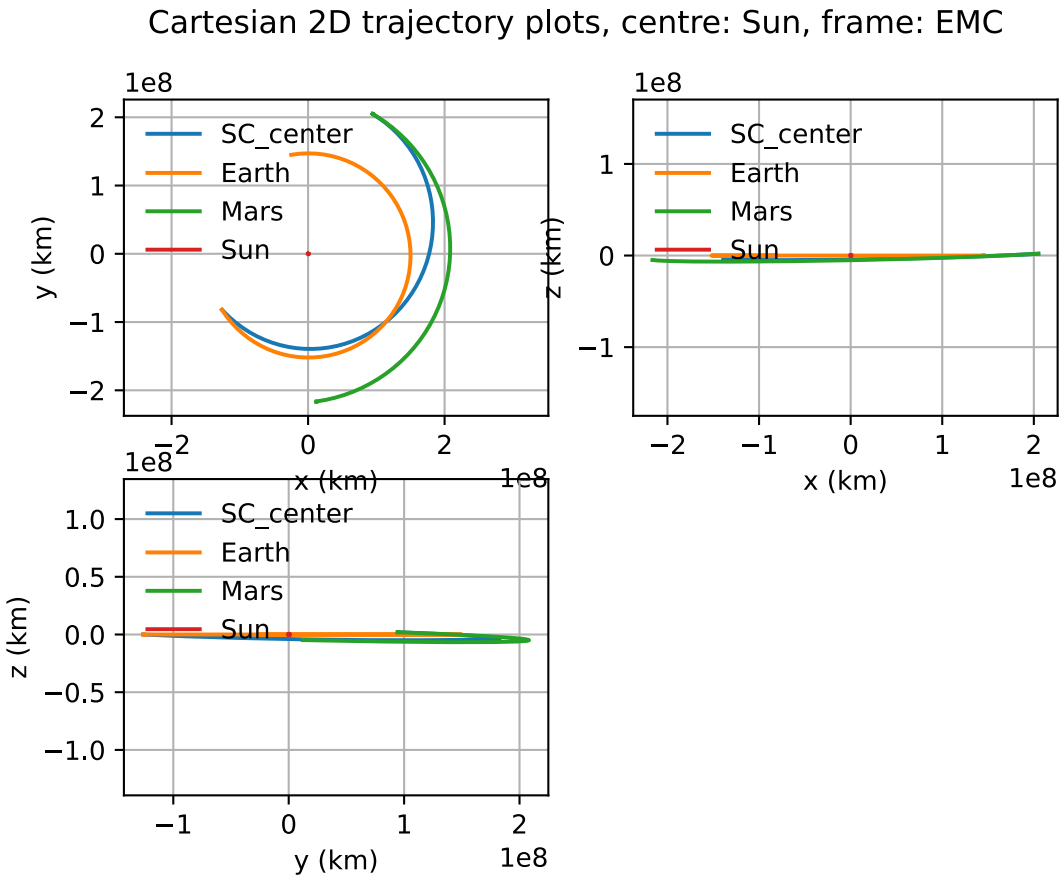


Figure 2.5: Earth to B-plane, propagation of the optimized trajectory

The final values for the mass of the spacecraft and the ΔV are the following:

- SC mass: 5729.925 kg

- SC ΔV : 5195.155 m/s

These evaluation will be the guess values for the trajectory of the whole mission that will be optimized.

2.1.3 Mars orbit capturing

The second part of the mission is the orbit circularization around Mars. It starts from hyperbolic trajectory to Mars then, once the periapsis is reached, a burn to circularize the orbit is performed to obtain the desired final orbit.

As starting data for the hyperbolic trajectory we will use the B-plane targeting data obtained by the previous optimization. We set $B * T$ length as the only value that can change for this part of the trajectory, and it will vary so that the hyperbolic tragectory's periapsis is equal to the objective final orbit's semi-major axis of 8000 km.

The manoeuvre is defined again with a TCN frame, this time with Mars as central body. The burn will happen when the periapsis is reached.

The final orbit parameter, apart from the semi-mayor axis $sma = 8000km$ and eccentricity $ecc = 0$, are free. The data are written in the MarsLocal frame.

The guess values are then summarized here:

Mars B-plane targeting:

- Arrival date: 2020-12-31T03:26:26.829 TDB
- V_{in} : 3 km/s
- $B * T$: 11750 km
- $B * R$: 0 km

Circularization burn:

- burn time: at periapsis
- dv: 2000 m/s
- ras: 180 deg
- dec: 0 deg

Mars orbit:

- Final date: 2021-01-02T03:26:26.829 TDB
- sma: 8000 km
- ecc: 0
- inc: 0 deg
- ran: 0 deg
- aop: 115 deg
- tan: 140 deg

In Figure 2.6 the propagation of the initial guess is shown. It is worth noticing how the circularized trajectory for the burn and the final one are fairly similar, factor that will enhance the probability of the optimization algorithm's convergence.

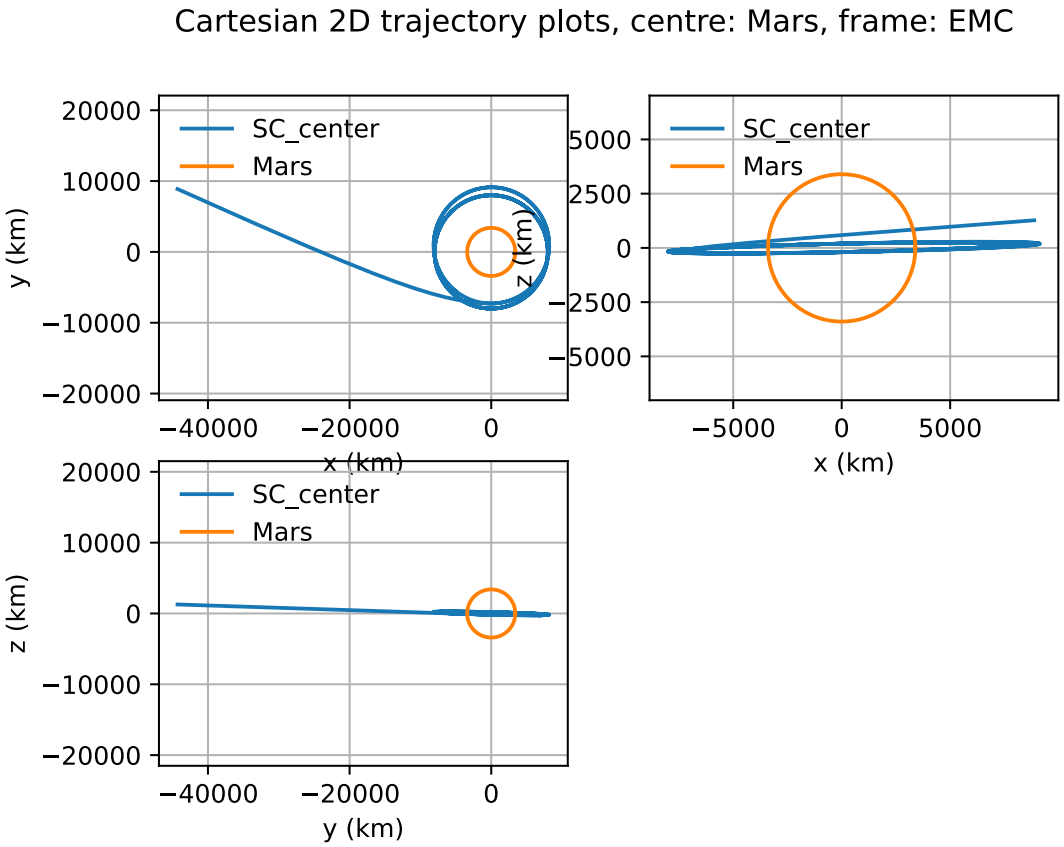


Figure 2.6: Hyperbolic trajectory to circularization, propagation of the first guess

After the optimization, we get the following outputs:

Mars B-plane targeting:

- Arrival date: 2020-12-31T03:26:26.829 TDB
- V_{in} : 3 km/s
- $B * T$: 11736 km
- $B * R$: 0 km

Circularization burn:

- burn time: at periapsis
- dv: 2125.495 m/s
- ras: 180 deg
- dec: 0 deg

Mars orbit:

- Final date: 2020-12-31T19:50:37.865 TDB
- sma: 8000 km
- ecc: 0
- inc: 0 deg
- ran: 0 deg
- aop: 84.559 deg
- tan: 116.873 deg

In Figure 2.7 we can see the final output, which has no discontinuities.

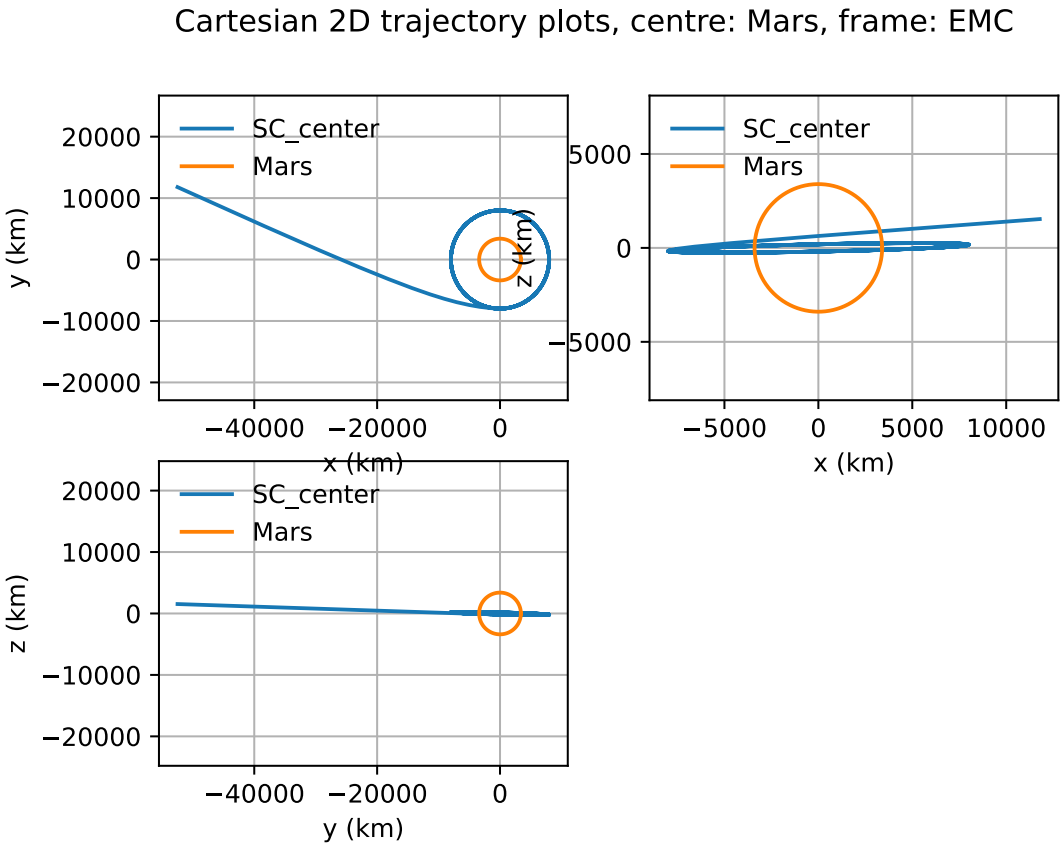


Figure 2.7: Hyperbolic trajectory to circularization, propagation of optimized trajectory

The final values for the mass of the spacecraft and the ΔV at the end are the following:

- SC mass: 2910.688 kg
- SC ΔV : 2125.494 m/s (only for this part of the mission)

The final required ΔV is then $\Delta V = 5195.155 m/s + 2125.494 m/s = 7320.649 m/s$.

2.1.4 Final optimization

There are two reason to perform a final optimization. The first is that there is a discontinuity between the trajectories as the value of $B * T$ for the B-plane targeting differs between the two

parts of the trajectory, thus a reevaluation needs to be performed. Secondly, now that we have an idea of the values of all the variables, we can perform an optimization with the whole mission in mind, which will increase the margin of optimization.

The guess values of the optimization are the outputs of the two previous ones. In Table 2.1 there are the final value of the orbit.

The final values for the spacecraft's mass and ΔV are the following:

- SC mass: 3960.005 kg
- SC ΔV : 6354.558 m/s

As we can see, there is a great saving in mass and ΔV, thus fuel, demonstrating the usefulness of this last step. It is important to note that the idea of a multiple stage rocket to launch the satellite haven't been analyzed in this test, as the tool mainly aims to analyze the ΔV required for the mission.

The plots of the trajectory are presented in Figure 2.8, 2.9, 2.10

Earth orbiting	
sma	6678 km
ecc	0
inc	2.74901603632635 deg
ran	0 deg
aop	118.464 deg
tan	136.127 deg
Earth escape manoeuvre	
epoch	2020-06-02T11:17:43.079 TDB
dv	4111.355
ras	-0.035 deg
dec	-3.019 deg
Cruise control manoeuvre	
epoch	2020-09-10T19:16:43.518 TDB
dv_x	35.474
dv_y	48.192
dv_z	424.460
Mars B-plane targeting	
epoch	2021-01-20T04:53:11.949 TDB
V_{in}	2.517 km/s
bt	13062.115 km
br	0 km
Circularization burn	
epoch	2021-01-20T08:19:46.785 TDB
dv	1814.545
ras	180 deg
dec	0.029 deg
Mars orbit	
epoch	2021-01-20T19:50:38.395 TDB
sma	8000 km
ecc	0
inc	0
ran	0
aop	81.327 deg
tan	113.642 deg

Table 2.1: Final value for the Earth to Mars mission

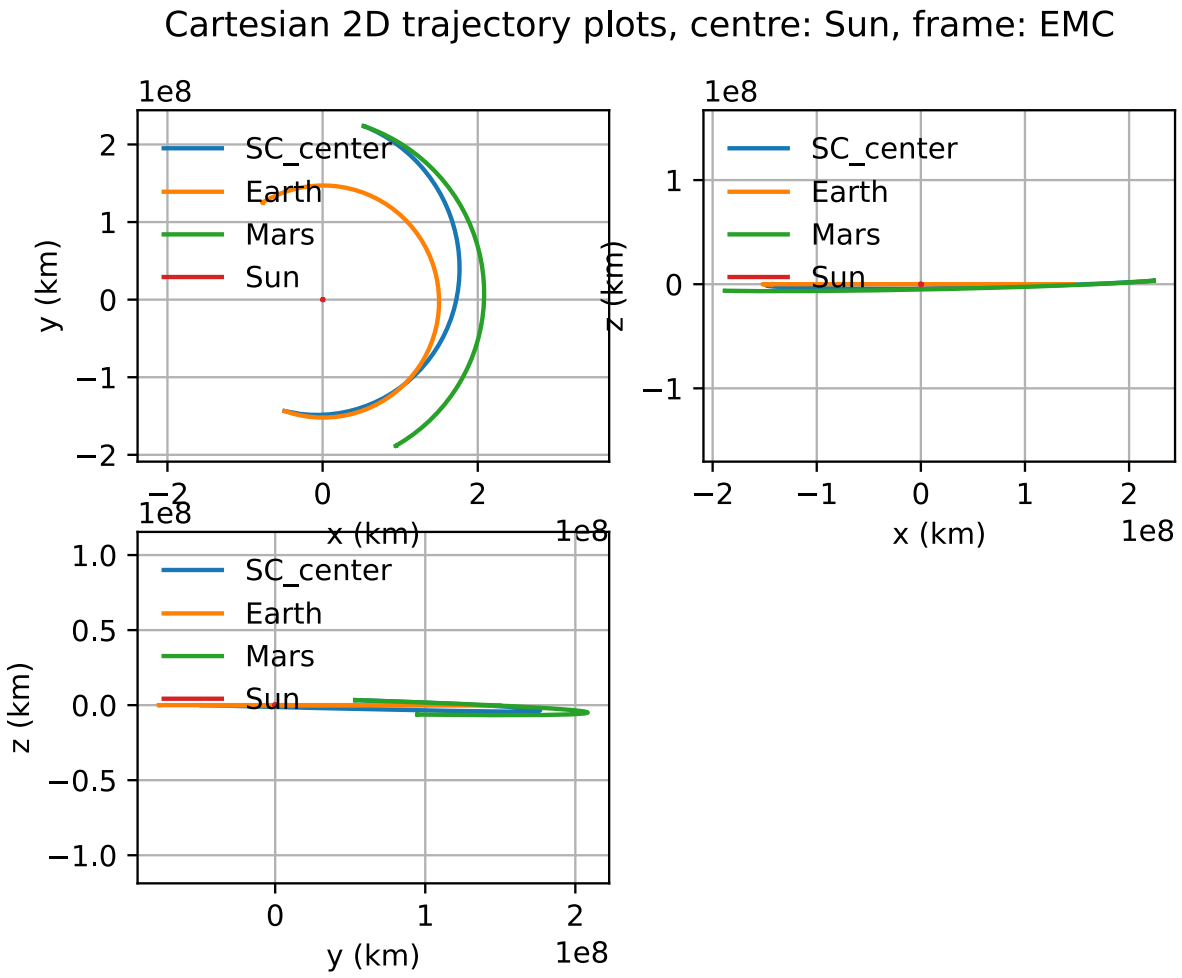


Figure 2.8: Optimized trajectory of the Earth to Mars mission

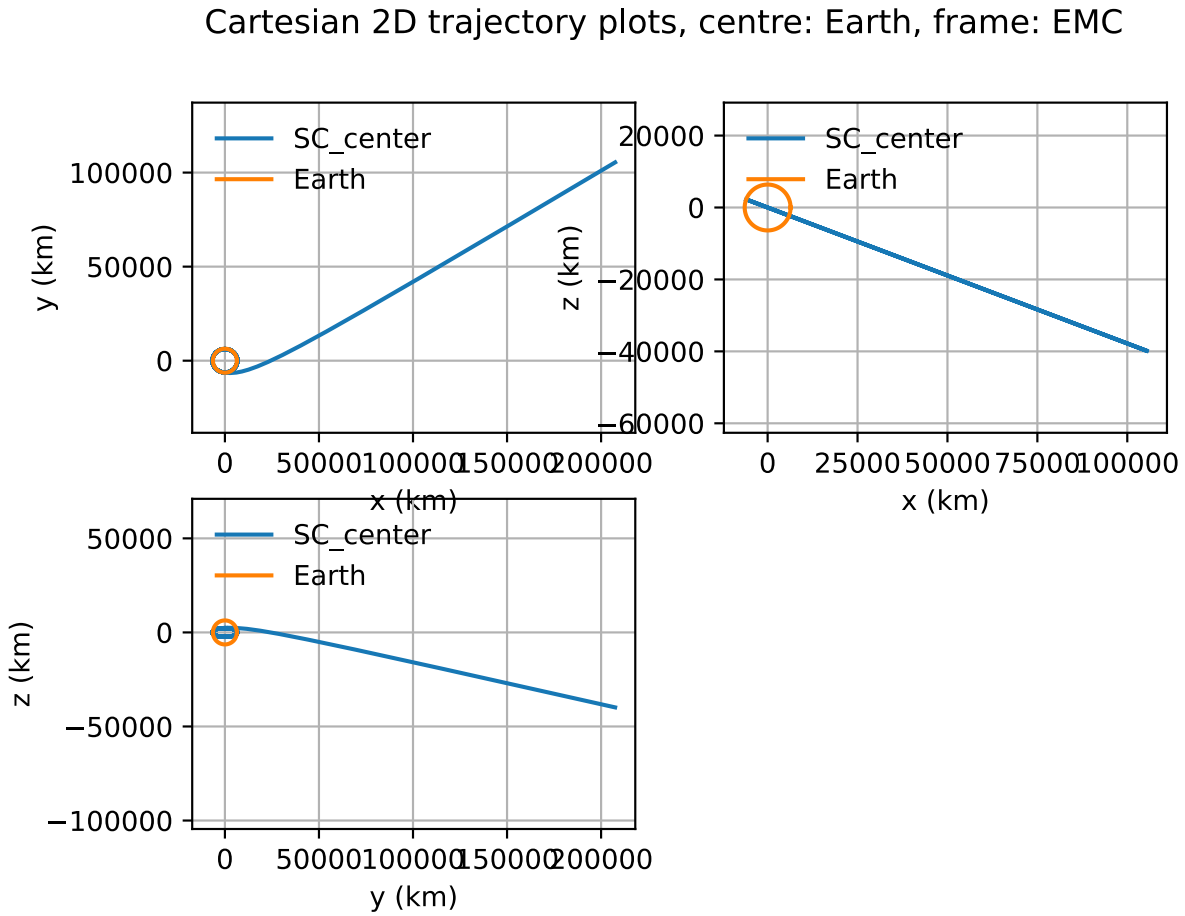


Figure 2.9: Optimized trajectory, departure from Earth

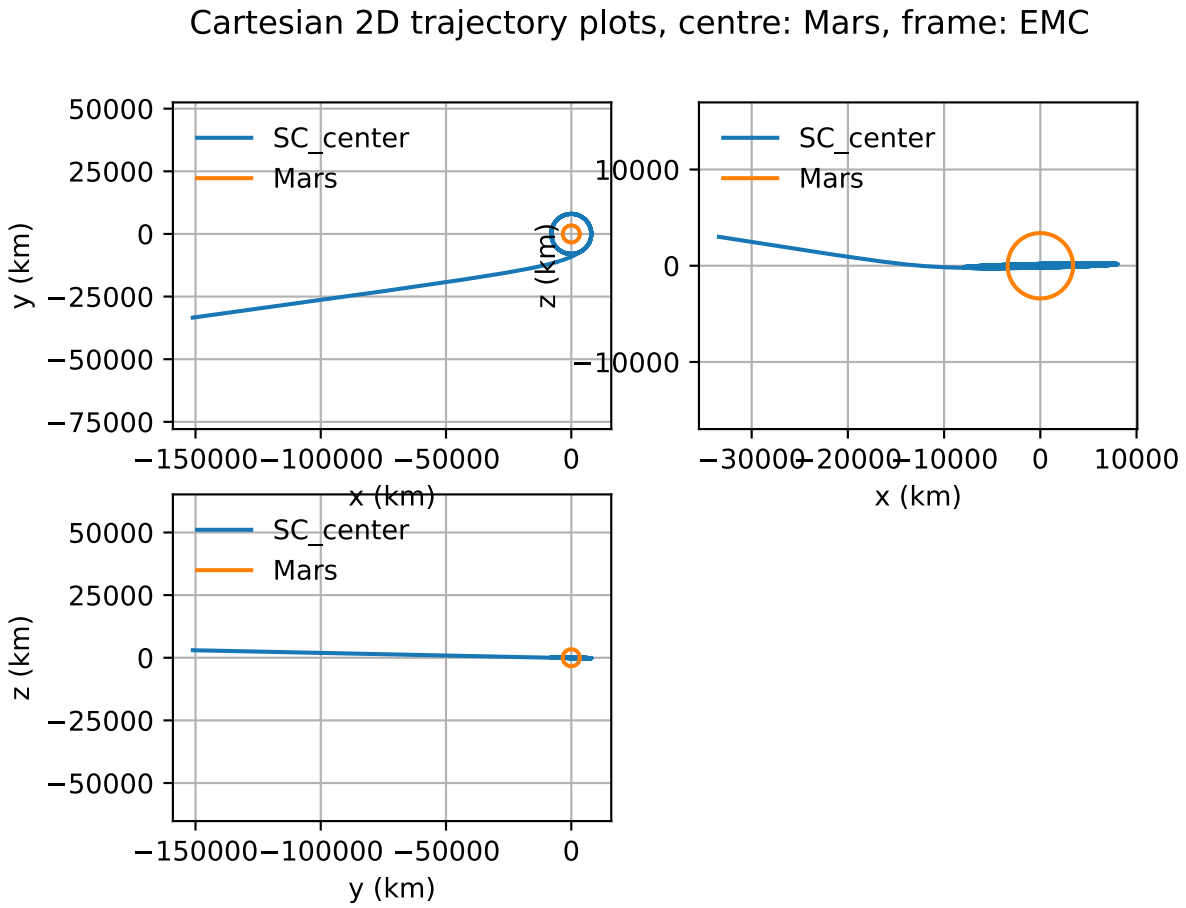


Figure 2.10: Optimized trajectory, Mars arrival and circularization

2.2 Moon free-return trajectory

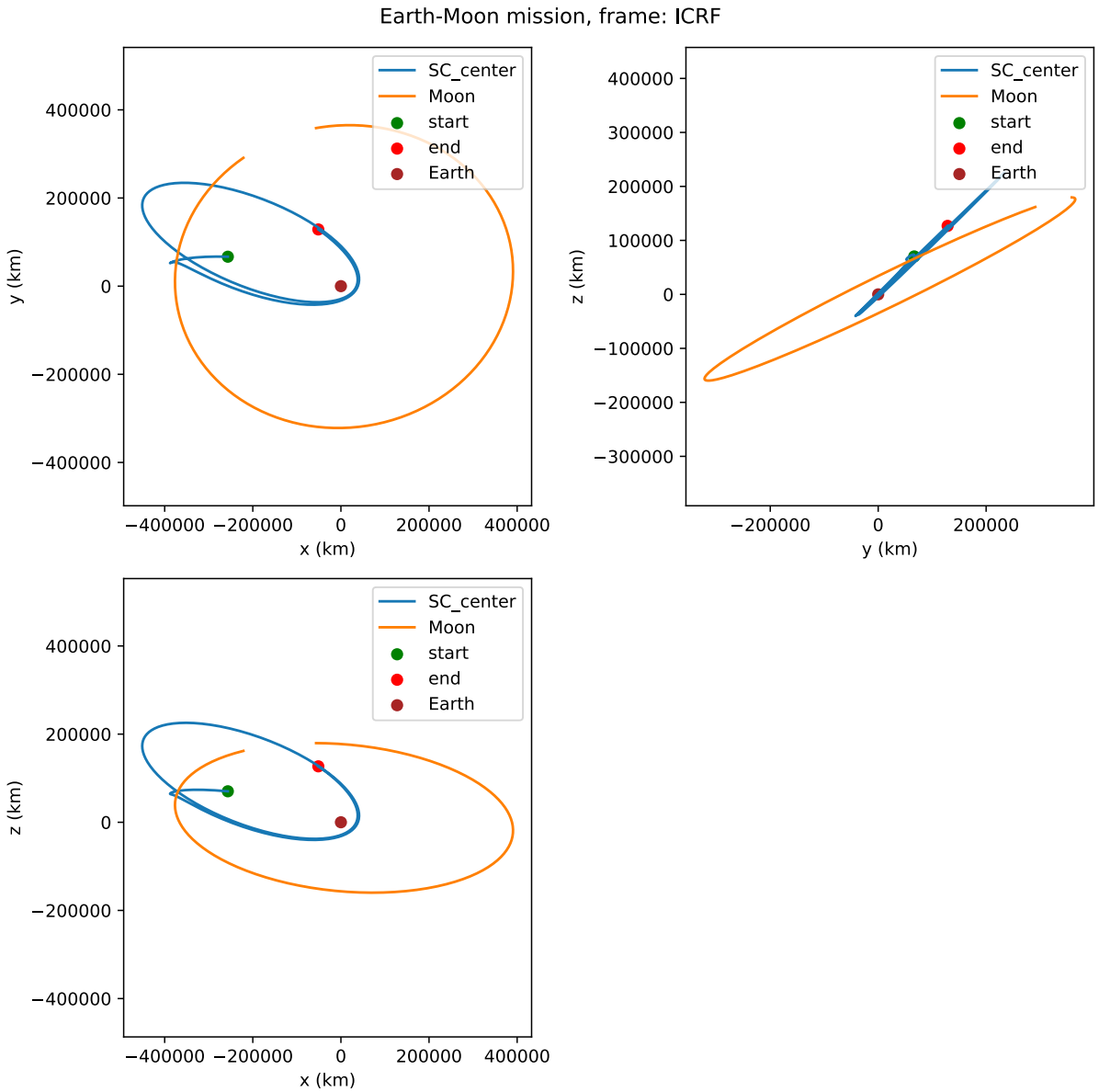


Figure 2.11: Free return trajectory from Earth to Moon in ICRF reference frame

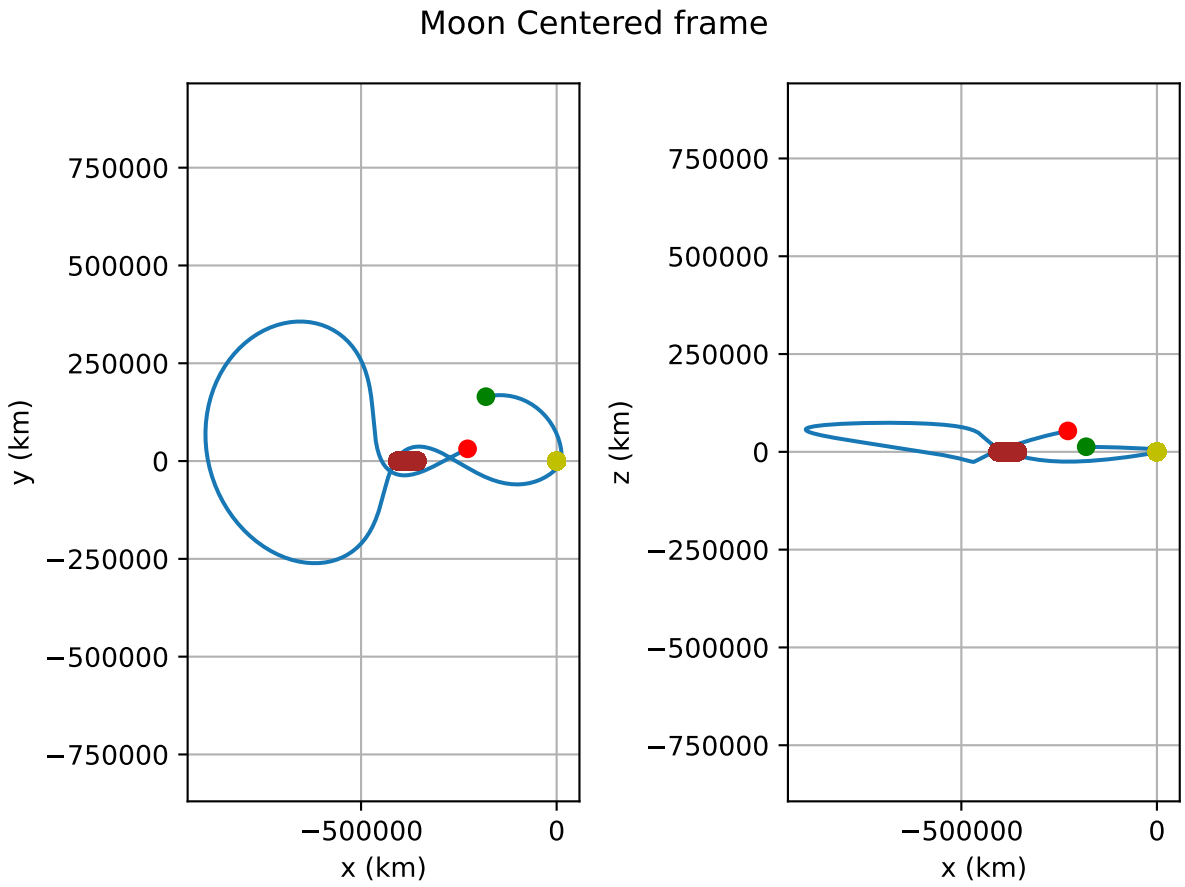


Figure 2.12: Free return trajectory from Earth to Moon in Moon centered reference frame

To further evaluate the software's capabilities, another mission has been evaluated. This time a free-return trajectory from Earth to Moon and back has been designed. For free-return we intend a trajectory on which a spacecraft departing from a body, in this case Earth, due to the gravity of a second body, Moon in this example, return to the first body with no propulsion applied.

In this test the values of the trajectory won't be exposed due to Non-Disclosure Agreement, but it is worth mentioning to demonstrate the capability of the software furthermore.

The trajectory is shown in Figure 2.11 in Earth centered ICRF, where the sudden turn of the

satellite at the mission's start is due to the Moon's gravity. The relative position between the satellite and the Moon is better illustrated in Figure 2.12, where the trajectory is presented in a Moon centered coordinate system. Here it is possible to observe how the both the Moon's gravity and the Earth's one drift the satellite and allow a free-return trajectory.

We also present the application's capability to perform a a Monte Carlo analysis, as shown in Figure 2.13. A Monte Carlo analysis consist in having different starting points, usually due to various uncertainties, and perform an analysis to have as many test cases as possible for when the real mission will be conducted. In this case, the uncertainty simulated is the position of the satellite after the release from a rocket.

In Figure 2.13 five orbits are presented, all with a different initial starting point. Although the perturbation, all the optimizations performed on the trajectories converged, demonstrating the robustness of the software.

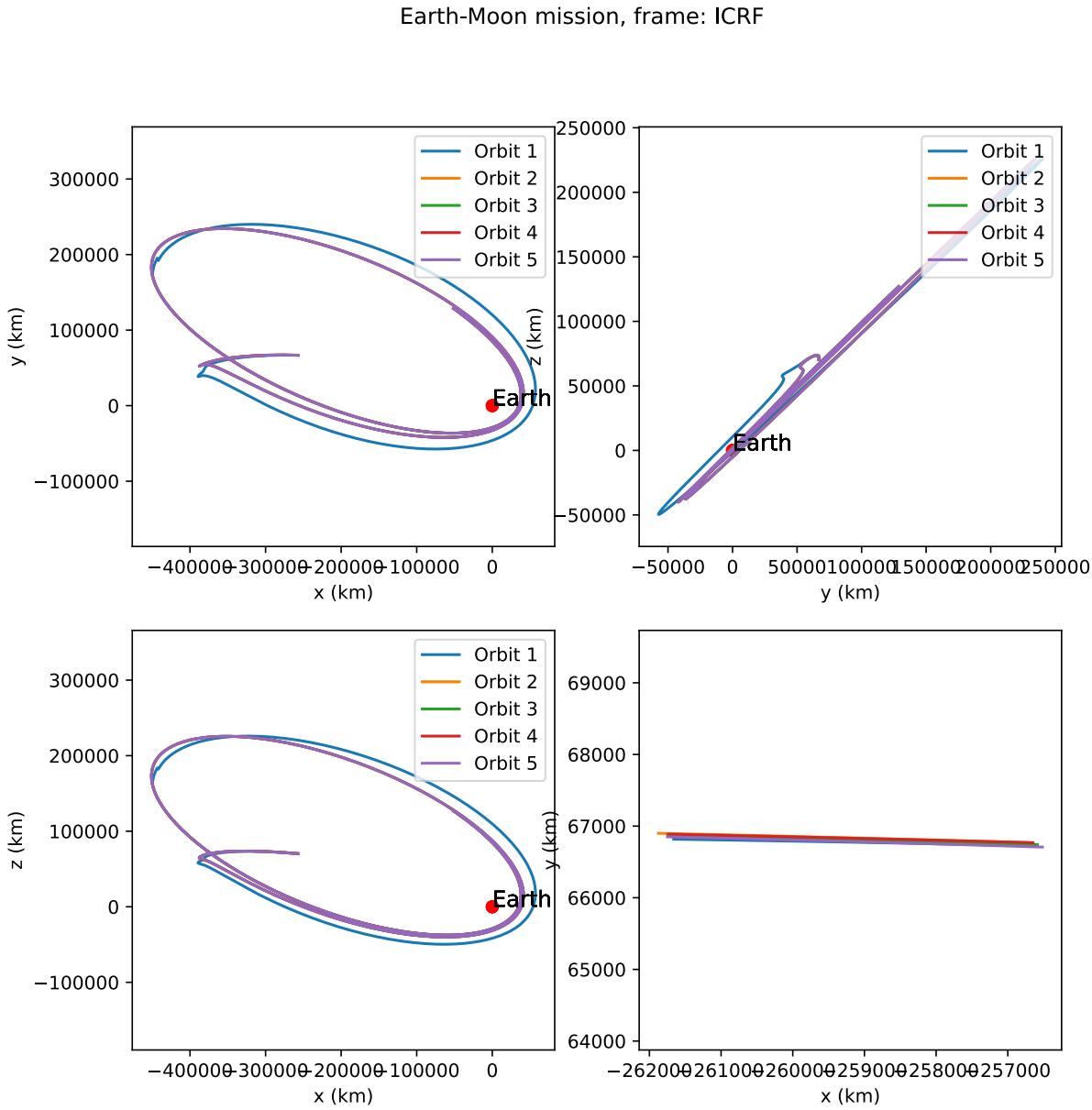


Figure 2.13: Monte Carlo analysis of the free return trajectory. The bottom right figure is a closeup to show the implementation of different starting points

Chapter 3

In-Plane Station keeping application

In planet observation missions, the satellite's ground track is chosen during the mission analysis, as it will define the observable terrain based on the Field of View of the satellite's payloads. It is then crucial during the mission to keep the real ground track as similar as possible to the ideal one to fulfill the payloads' requirements. To do so orbit correction manoeuvres, that take the name of station keeping manoeuvres are performed.

An application has been developed to define the required manoeuvres to perform an in-plane station keeping. In-plane manoeuvres are engine burns along the orbit's velocity vector that allow to control the semi-major axis of the orbit, to adjust in turn the time and position of the satellite's equator crossing. These are used to compensate the decay in semi-major axis due to air friction, thus are more frequent the lower is the satellite's altitude.

3.1 In-plane manoeuvre optimization

The developed application takes inspiration from the orbit control strategy of the Aeolus satellite [34]. The core idea is to evaluate the ground track deviation between the real and ideal trajectory at the equator crossings and perform In-plane manoeuvres to prevent the difference to be outside

a control-band, that for the Aeolus case is $\pm 25km$.

The data required to analyze the station-keeping manoeuvres are the longitudes of the ideal and real trajectory at the equator passes. While the ideal trajectory is defined by the mission, the real trajectory can be analyzed by propagating the satellite's trajectory via the GODOT library.

To do so, especially when working with LEO orbits, an accurate atmosphere model has to be used. The one used by the GODOT library is the NRLMSISE-00 empirical model [35]. In the application the solar and geomagnetic activity data required to work with the model are made by the software package SOLMAG [36]. This software is used for long term activity predictions, thus the results of the following analysis will have to be validated using short to medium term predictions, like the outputs of the PDFLAP software presented in [36], in a time period near the station-keeping manoeuvre date.

Once the longitudes have been calculated, it is possible to interpolate the ground track deviation at the equator passes to obtain a plot like the one in Figure 3.1. As expected, the ground-track deviation increases with time and gets closer to the East limit, as with the decrease in semi-major axis there is a decrease in orbital period.

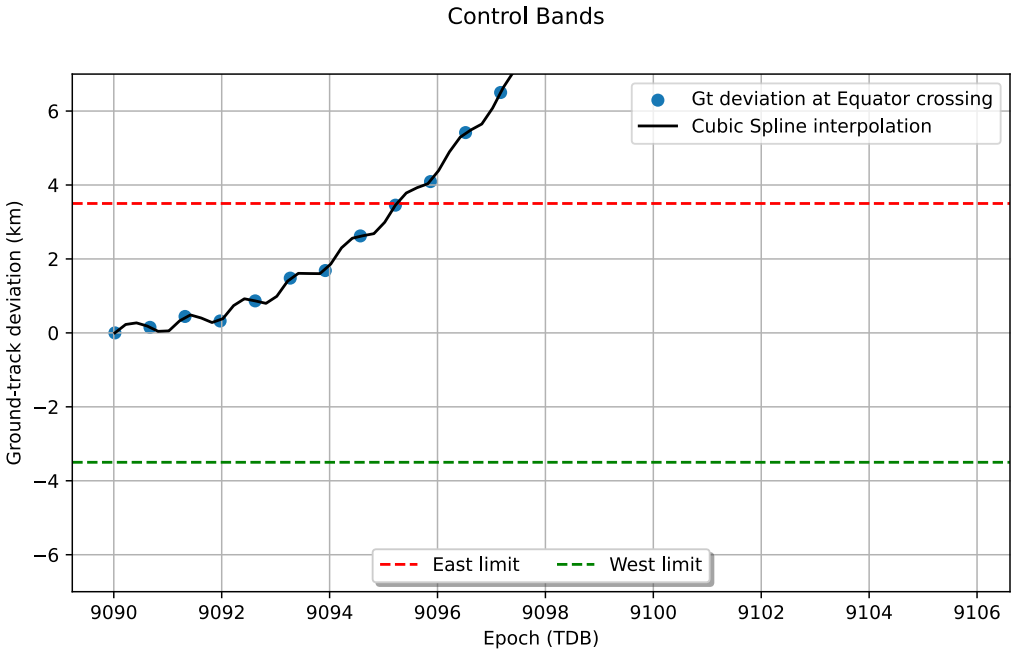


Figure 3.1: Ground-track deviation with control bands

Once the surpass of the control-bands has been detected, the analysis for the ΔV required to perform the In-plane station-keeping manoeuvre will be performed. As the Aeolus paper [34] suggests, the optimization target is smaller than the West limit to prevent its violation due to uncertainty of the solar activity or of performance errors. This consideration is demonstrated in Section 3.4, as Figure 3.10 (a) shows how, performance errors or the difference in solar activity heavily affect the evolution of the Ground-track deviation.

The optimization is performed by a bisection algorithm [37] by considering the ground-track distance as a function of ΔV and time

$$gt = f(\Delta V, t)$$

Let opt be the value in km of the optimization target. Then two margins, $opt_+ > opt$ and $opt_- < opt$ around opt are created to have a less strict objective. The aim is to to find the ΔV

so that the ground-track function gets close enough to opt.

In this case for a chosen ΔV the zero is found when $gt > opt_-$ for every t in a defined time interval and there is at least one t so that $gt < opt_+$, which means that the trajectory entered in the opt margin without getting too close to the West limit.

The bisection algorithm requires a lower and upper limit of the variable and the value of the variable so that the function is zero shall be inside these limits. The initial lower limit is $\Delta V = 0$, while for the upper limit the twice of a $\Delta V_{first-guess}$ chosen by the user is used. The guess interval is then

$$[0, \Delta V_{first-guess} * 2]$$

To ensure the presence of a zero inside the interval, as the intermediate-value theorem for continuous functions suggests, the app checks if there is at least one t in the defined time interval so that

$$gt > opt_-$$

If this is not the case, another $\Delta V_{first-guess}$ is added to the upper limit and the loop goes until a sufficiently high $\Delta V_{first-guess}$ is found. From here the typical bisection method is applied, where the function is evaluated for every t using as ΔV the value in the middle of the interval. If the resultant function never crosses the opt_+ value, the new lower value of the interval become the ΔV before obtained, while the upper value remains the same; if the function crosses the opt_- value, the upper value is replaced.

By iterating this algorithm a value of ΔV is eventually obtained so that the condition for the zero is fulfilled or the difference between the two limits of the interval of ΔV is smaller than a tolerance $toll$ chosen by the user.

To sum up, the pseudo-code of the algorithm is presented in Algorithm 1.

Algorithm 1: Bisection method for evaluating ΔV

Data: opt_+, opt_-, $\Delta V_{first-guess}$, $toll$, t_0 and t_f

Result: Evaluate the ΔV for an In-plane station-keeping manoeuvre

$timeInterval = [t_0, t_f]$;

$lower_{\Delta V} = 0$;

$upper_{\Delta V} = \Delta V_{first-guess} * 2$;

while $gt > opt_-$ *for every t in timeInterval* ***if*** $\Delta V = upper_{\Delta V}$ **do**

    $upper_{\Delta V} = upper_{\Delta V} + \Delta V_{first-guess}$;

while *Solution not found* **do**

    $\Delta V = (lower_{\Delta V} + upper_{\Delta V})/2$ orbit propagation with ΔV as input;

    a flag is used to determine if the function passes over opt;

    $Flag =$ "not passed over opt";

    foreach t *in* $timeInterval$ **do**

        if $gt = f(\Delta V, t) < opt_+$ **then**

            if $gt = f(\Delta V, t) > opt_-$ **then**

                Solution found;

            else

                $Flag =$ "passed over opt";

                break out the **for** statement;

    if $Flag =$ *"not passed over opt"* **then**

        $lower_{\Delta V} = \Delta V$;

    if $Flag =$ *"passed over opt"* **then**

        $upper_{\Delta V} = \Delta V$;

    if $upper_{\Delta V} - lower_{\Delta V} < toll$ **then**

        Solution found due to max toll reached;

An example output of this iteration can be seen in Figure 3.2.

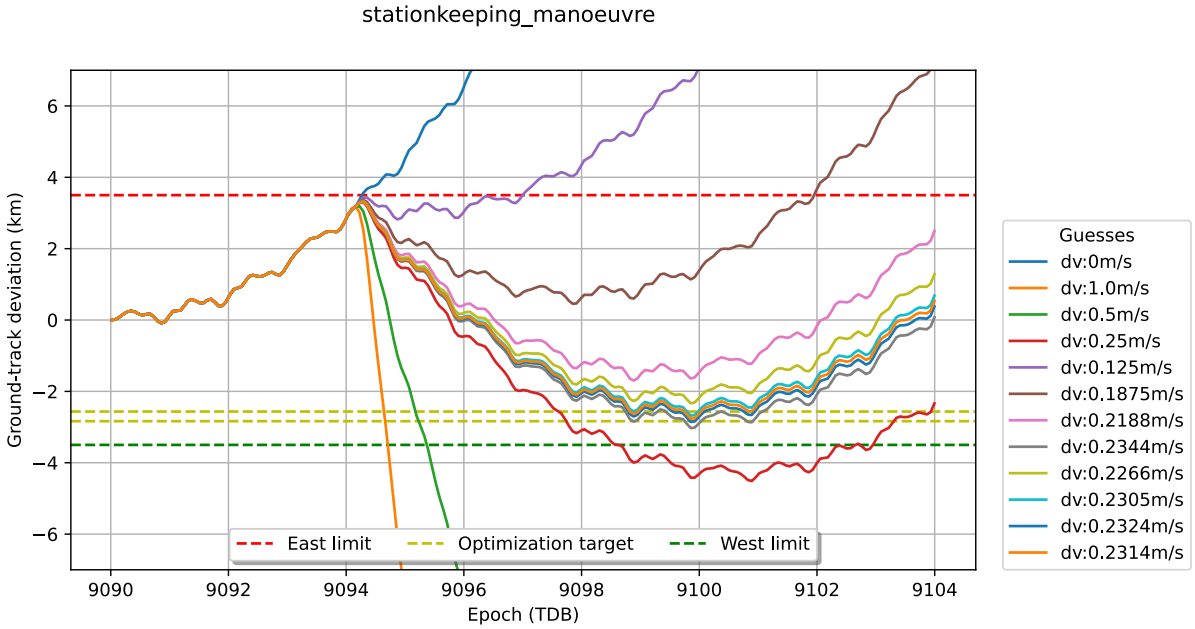
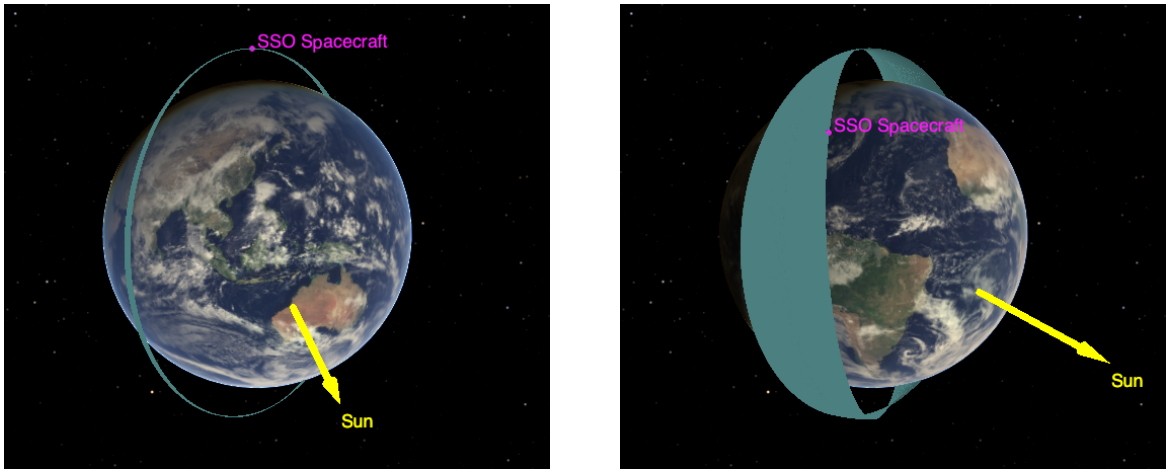


Figure 3.2: Representation of the iterations of the bisection algorithm

To validate the tool test analyses have been conducted regarding the so-called Sun-Synchronous Orbits .

3.2 Sun-Synchronous Orbits

A Sun-Synchronous orbit (abbreviated as SS-O later in this paper) is a near polar orbit around Earth with the property of maintaining a constant angle between the orbit's plane and the vector from the orbit's centre to the Sun, as we can see in Figure 3.3.



(a) SS-O **(b)** Same orbit 30 days later

Figure 3.3: Comparison between a SS-O (a) and the same orbit 30 days apart (b)

The SS-O works as such because, based on the defined altitude, the orbit's inclination is chosen so its precession rate, due to the J2 effect, is the same as the Earth's Mean Motion. This means that the orbital altitude and inclination for SS-Os are uniquely paired, as shown in Figure 3.4. The relation is presented:

$$\dot{\Omega} = -\frac{3}{2} J_2 (\frac{a_e}{p})^2 n * cos(i) \tag{3.1}$$

where $p = a(1 - e^2)$ is the semi-latus rectum, $n = \sqrt{\mu/a^3}$ is the mean motion, i is the inclination, a is the semi-mayor axis, a_e is Earth's equatorial radius, $\dot{\Omega}$ is the precession rate that in this case is equal to the Earth's mean orbital rate. It is possible to notice how the only two free parameters are the orbit's inclination and semi-mayor axis.

This property allows to have a ground track with the same sun lighting for the whole mission's duration, a desirable feature for analysing terrain areas in different periods of time. There are also advantages from a thermal analysis standpoint, due to a constant thermal environment and an orbit "dark side" useful to cool down the satellite [38].

Of particular interest are orbits with repeated ground tracks, which are orbits that retrace

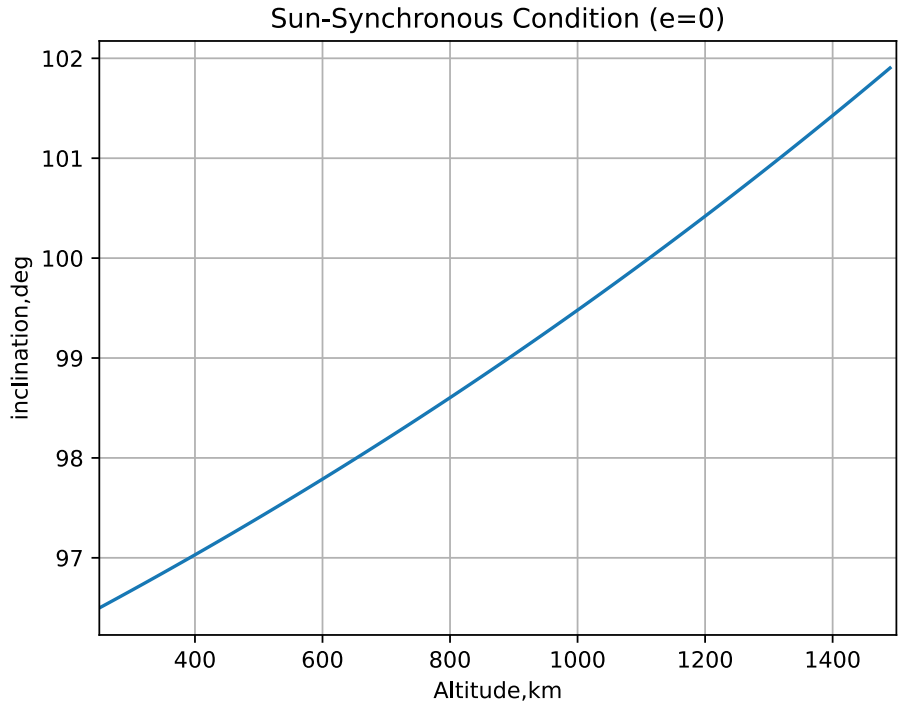


Figure 3.4: Inclination vs. Altitude for SSOs

themselves after an integer amount of revolutions and day.

As there is a correlation between the height of an orbit and its period

$$T = 2\pi\sqrt{\frac{a^3}{\mu}} \tag{3.2}$$

it is possible to evaluate the orbit's altitude based on the desired amount of revolutions and days. By considering as desirable orbits with altitudes in the range between 250 and 1680 km, a plot like the one in Figure 3.5 can be deduced. Not all the orbits are presented in the plot to avoid a crowded image.

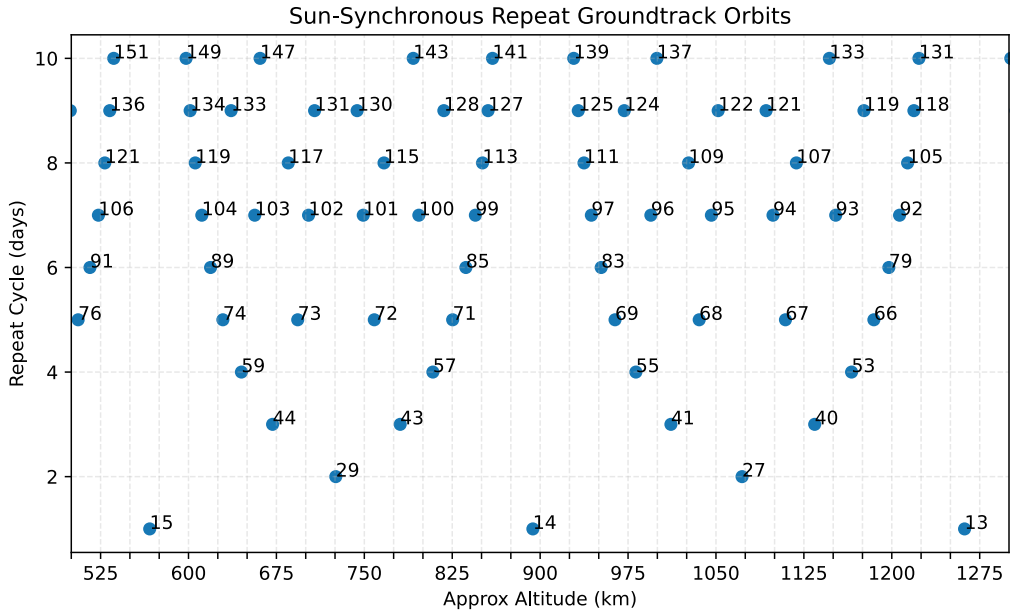


Figure 3.5: Revolutions and Days vs. Altitude for SS-Os. The values near the dots are the number of cycles

3.3 SS-O Test analysis

To evaluate the capability of the software tests have been conducted using a variety of SS-Os with different amount of revolutions and days. Table 3.1 shows the initial values of the orbits.

	1D16R	2D31R	3D44R	5D74R	7D111R	9D139R
sma	6652.56 km	6794.86 km	7049.86 km	7007.46 km	6692.45 km	6811.14 km
inc	94.83 deg	95.27 deg	96.11 deg	95.96 deg	94.95 deg	95.32 deg

Table 3.1: Semi-major axis and inclinations of the SS-Os used for the app validation (D: number of days, R: number of revolutions)

It is important to address why the inclination is different than the one from the Figure 3.4. The analysis in [38] to obtain the relation between inclination and altitude start from an idealization of the Earth's movement around the Sun, simplifying the motion by considering it

circular and considering the satellite influenced only by the J2 perturbation.

To reduce the approximation, thus increasing the accuracy, the inclination required to obtain a repeating ground-track has been again evaluated, empirically this time. In the simulation the Earth's orbit is no more considered circular, while the idealization of considering only the spherical harmonics up to J2 still holds, as the other perturbations only slightly affects the change in inclination and is easier to compare these results with the theoretical ones. The requirement to obtain a repeating ground-track is that the distance between two consecutive ascending nodes at the equator crossing is equal to a valued named "fundamental interval". This value is:

$$\Delta L = 360^\circ * D/R$$

so it is a function of D, number of days, and R, number of revolutions. The inclination required to archive this requirement has been computed using another bisection algorithm, which aims to find the right inc so that

$$\Delta L_{real} = \Delta L_{ideal}$$

where ΔL_{ideal} depends on the mission requirements.

The lower limit required by the algorithm is set as $lower_{inc} = 90deg$, as a SS-O requires an inclination higher than $90deg$, while the upper limit is chosen by the user. A first check is performed control if $\Delta L_{real} \geq \Delta L_{ideal}$, as this ensure that the upper inclination is higher than the required one. If the check fails, the inclination of the upper limit is increased by $10deg$ and the check is again performed until it is satisfied.

Then the usual iteration of a bisection algorithm is performed, where the candidate inc is evaluated as

$$inc = (lower_{inc} + upper_{inc})/2$$

If $\Delta L_{real} \geq \Delta L_{ideal}$, then the upper limit is replaced by inc, otherwise is the lower limit to be replaced.

If $\Delta L_{real} = \Delta L_{ideal}$ the algorithm found the solution. The algorithm end also if the difference between the limits, or the difference between ΔL_{real} and ΔL_{ideal} is lower than two user defined tolerances.

The pseudo-code is shown in Algorithm 2.

Algorithm 2: Bisection method for evaluating SS-O inclination

Data: ΔL, $firstGuess$, $deltaToll$, $incToll$
Result: Evaluate the inc to have a repeating ground-track
$lower_{inc} = 90°$;
$upper_{inc} = firstGuess$;
Evaluate real distance between the first two ascending nodes at equator crossing = ΔL_{real};
while $\Delta L_{real} < \Delta L$ **do**
    $upper_{inc} = upper_{inc} + 10°$;
    Propagate orbit; Evaluate new ΔL_{real}
while *Solution not found* **do**
    $inc = (lower_{inc} + upper_{inc})/2$;
    Propagate orbit;
    Evaluate new ΔL_{real}
    if $\Delta L_{real} > \Delta L$ **then**
        $upper_{inc} = inc$;
    if $\Delta L_{real} < \Delta L$ **then**
        $lower_{inc} = inc$;
    if $|upper_{inc} - lower_{inc}| < incToll$ **then**
        Solution found due to incToll reached;
    if $|\Delta L_{real} - \Delta L| < deltaToll$ **then**
        Solution found due to deltaToll reached;

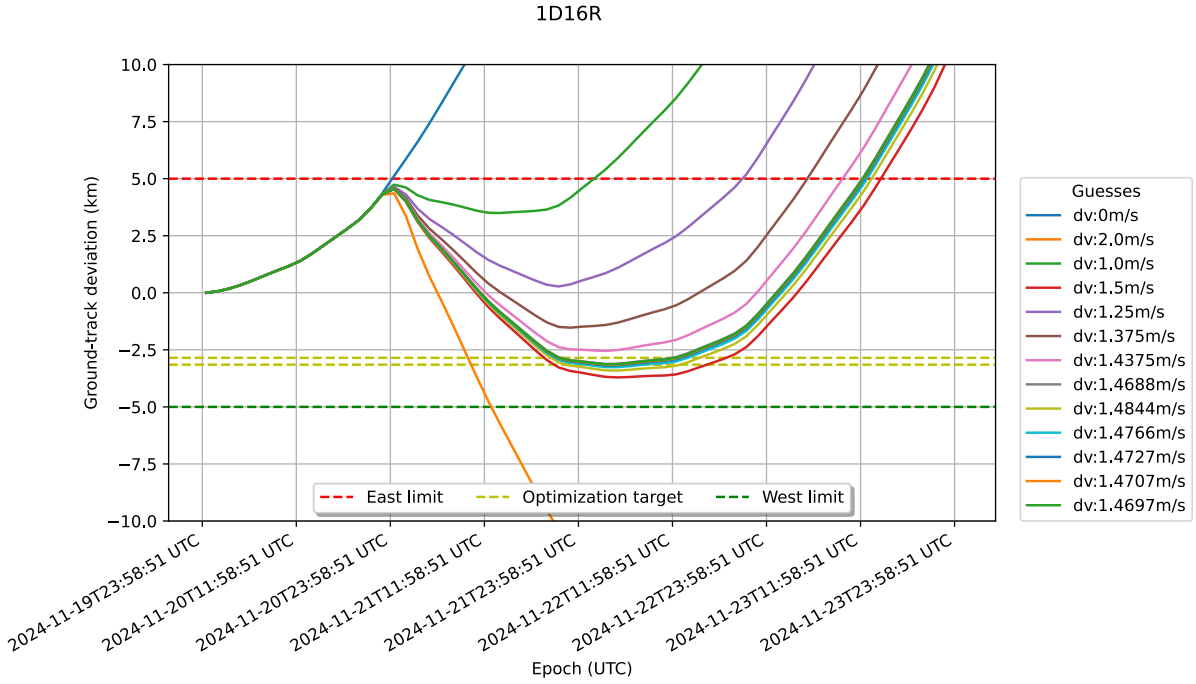
For a generic cubesat with $20\ kg$ mass, drag area of $0.08\ m^2$ and Cd=2.2, the values of ΔV required for the stationkeeping manoeuvres, considering control bands of $\pm 5km$ and with $opt = -3km$, obtained via the Algorithm 1, are summarized in Table 3.2. The optimization process can be seen in Figure 3.6,3.7,3.8. These Figures have different propagation times for a

better visualization, as the frequency to which the control-bands are violated heavily depends on the orbit's height, thus the different air drag experienced by the satellite.

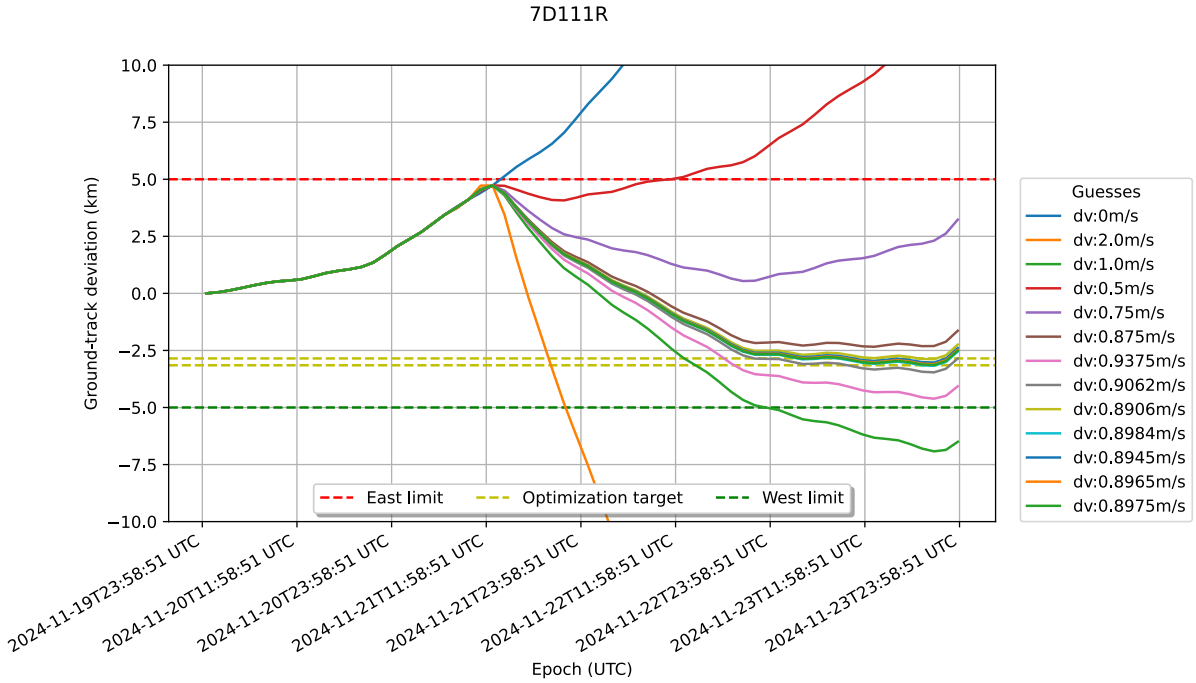
	1D16R	2D31R	3D44R	5D74R	7D111R	9D139R
sma	6652.56 km	6794.86 km	7049.86 km	7007.46 km	6692.45 km	6811.14 km
ΔV	1.470 m/s	0.311 m/s	0.046 m/s	0.052 m/s	0.897 m/s	0.2701 m/s

Table 3.2: ΔV required to perform the In-plane station-keeping. It is possible to note the inverse proportionality between the semi-mayor axis and the required ΔV

The app is robust and can optimize the orbits independently from number of days and revolutions. It can also optimize multiple manoeuvres to keep the satellite from violating the control-band, as shown in Figure 3.9

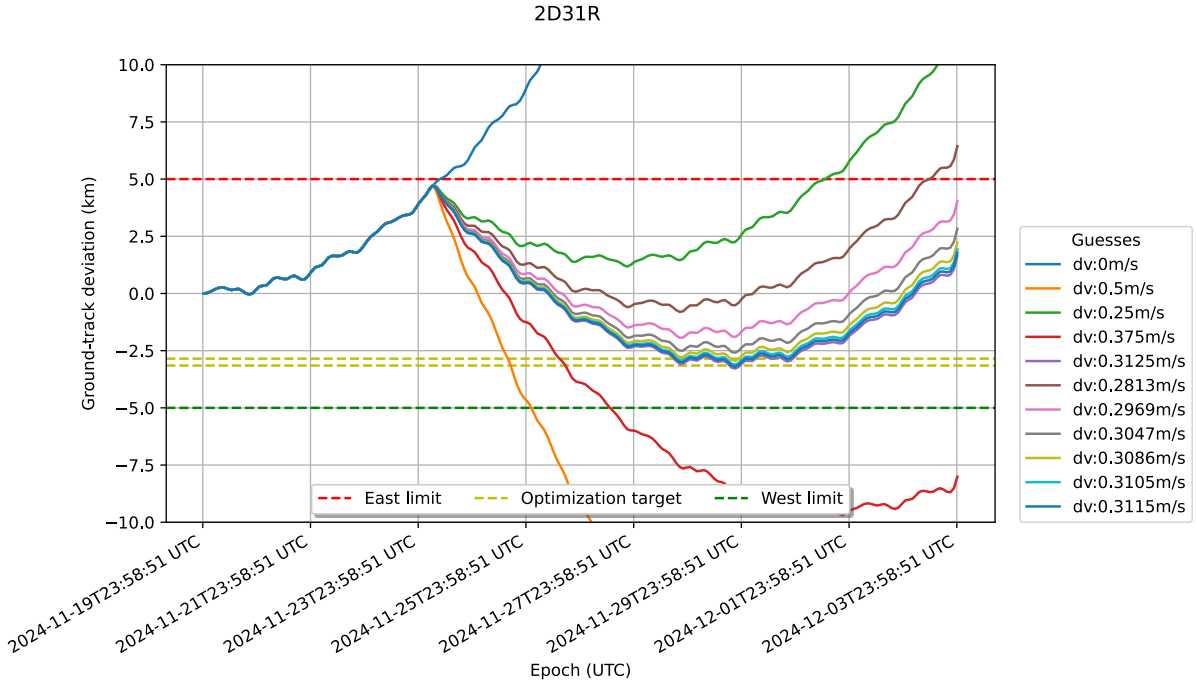


(a) 1 Day 16 Revolutions

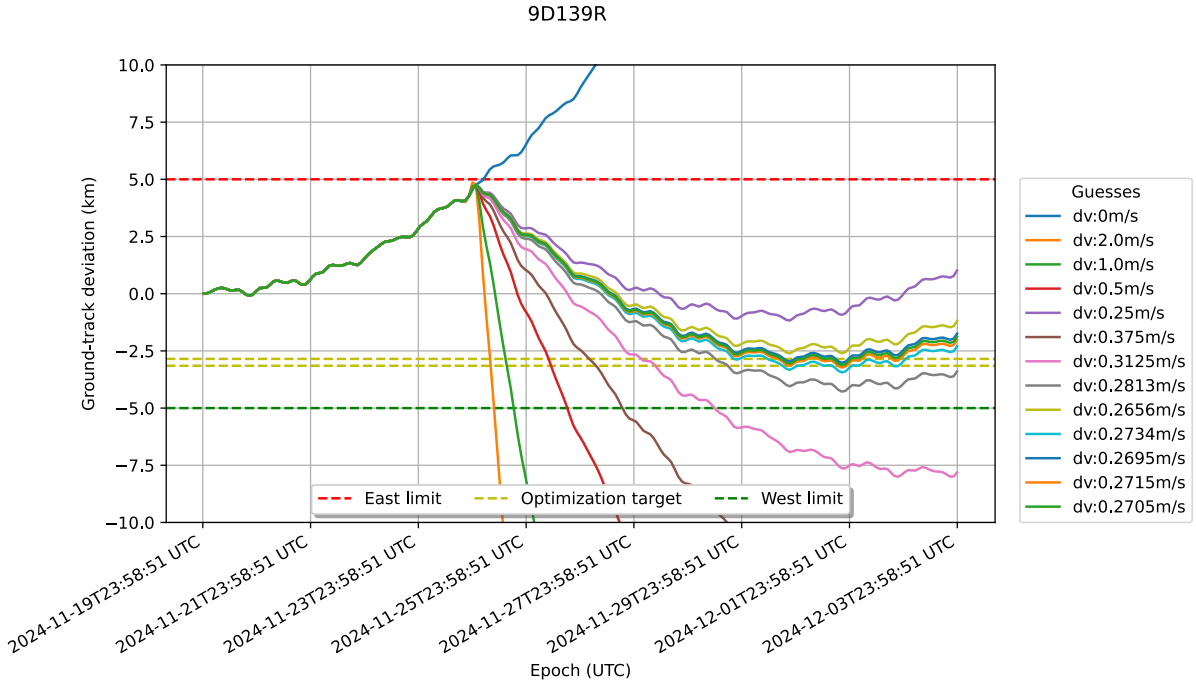


(b) 7 Days 111 Revolutions

Figure 3.6: Optimization process — 4 Days propagation and $sma \sim= 6700\ km$

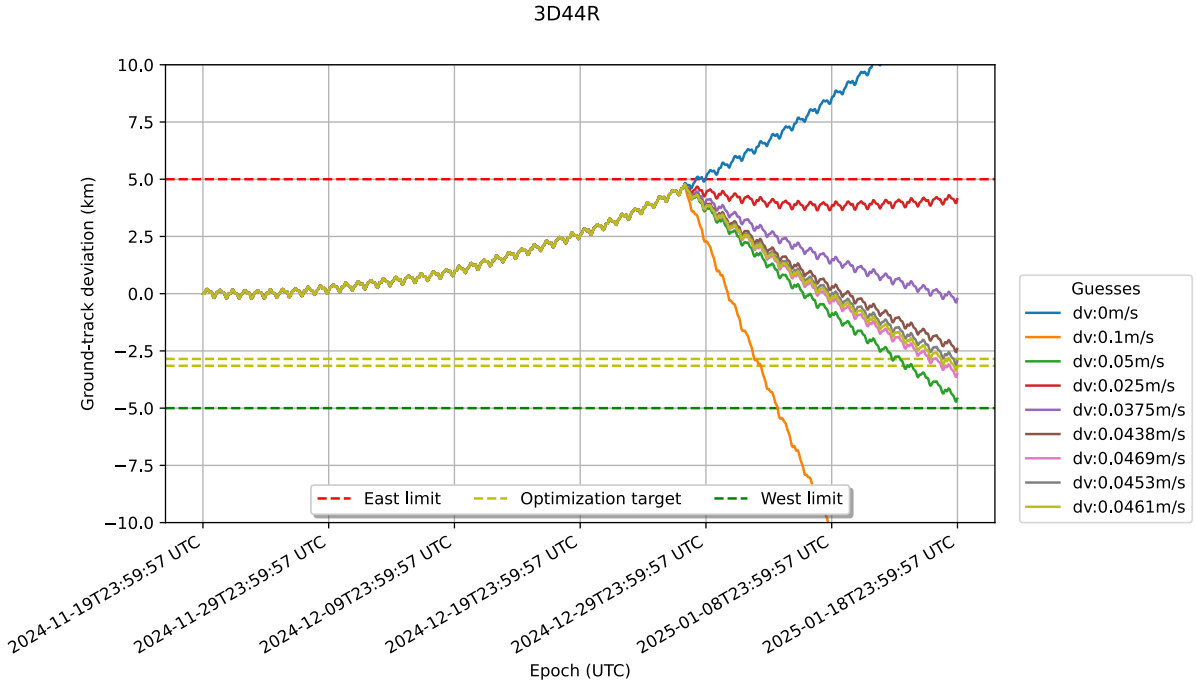


(a) 2 Days 31 Revolutions

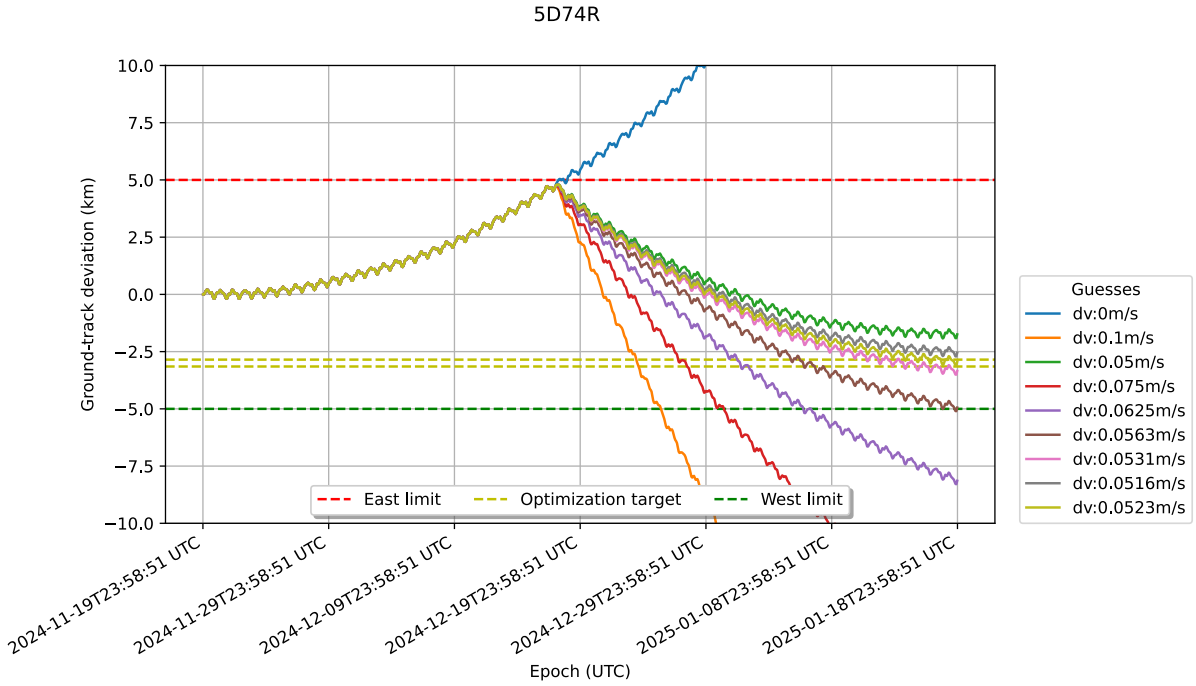


(b) 9 Days 139 Revolutions

Figure 3.7: Optimization process — 14 Days propagation and $sma \sim= 6800\ km$



(a) 3 Days 44 Revolutions



(b) 5 Days 74 Revolutions

Figure 3.8: Optimization process — 60 Days propagation and $sma \sim= 7000\ km$

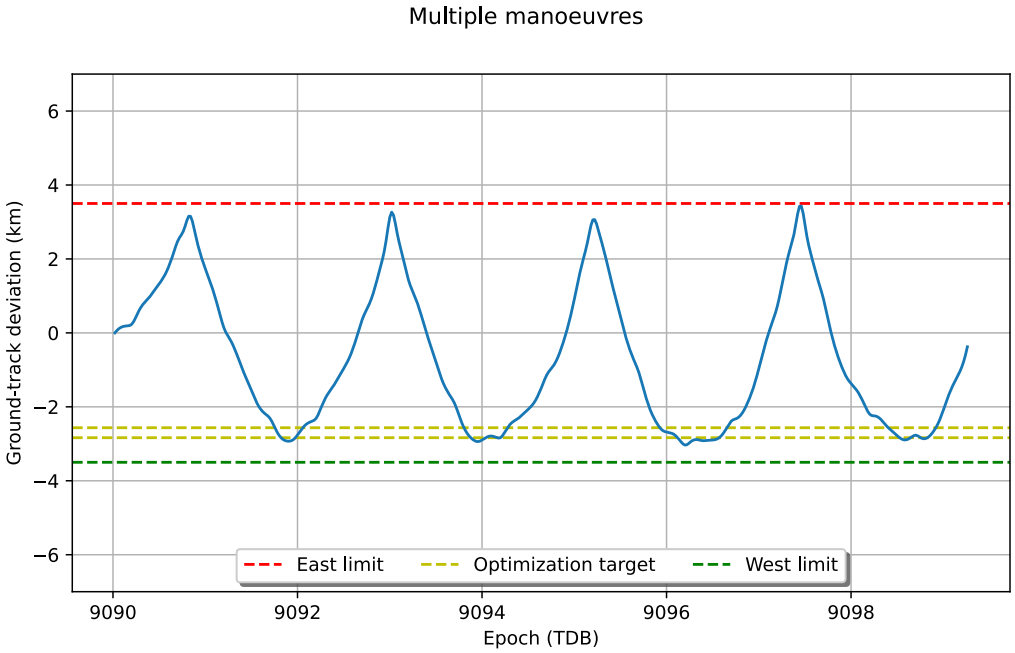


Figure 3.9: 1D16R orbit with four station-keeping manoeuvres

3.4 Comparison with Aeolus In-Plane station-keeping

To further validate the application, a comparison with between the data in [34] of the Aeolus mission and the application's output can be conducted. In Table 3.3 the data of the Aeolus spacecraft at the start of the orbit control phase are presented.

Dry mass	Propellant mass	Equivalent drag area	Cd	SRP area	SRP coefficient
1079 kg	262 kg	6,163 m^2	1.5	24.25 m^2	1.3

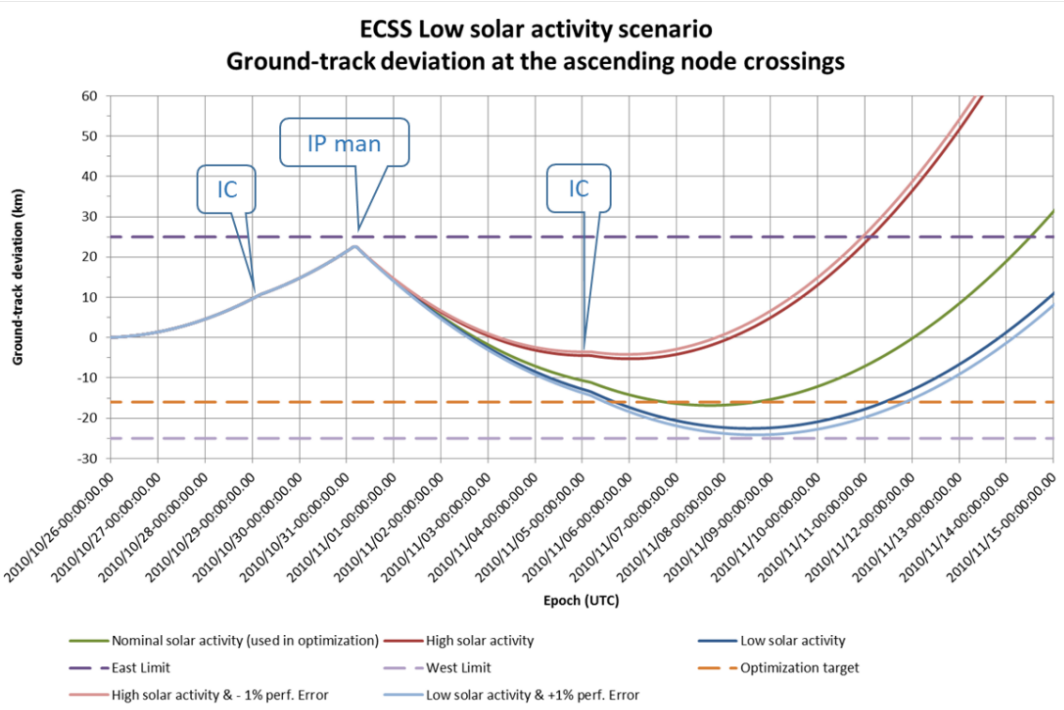
Table 3.3: Aeolus spacecraft parameters relevant to the orbit control analysis

These value have been given as input in the app. The other required input are the orbit's

inclination and semi-mayor axis, that for Aeolus are around 97 deg and 320 km respectively. These data correspond with a 6D95R SS-0, which as been used as input for the app.

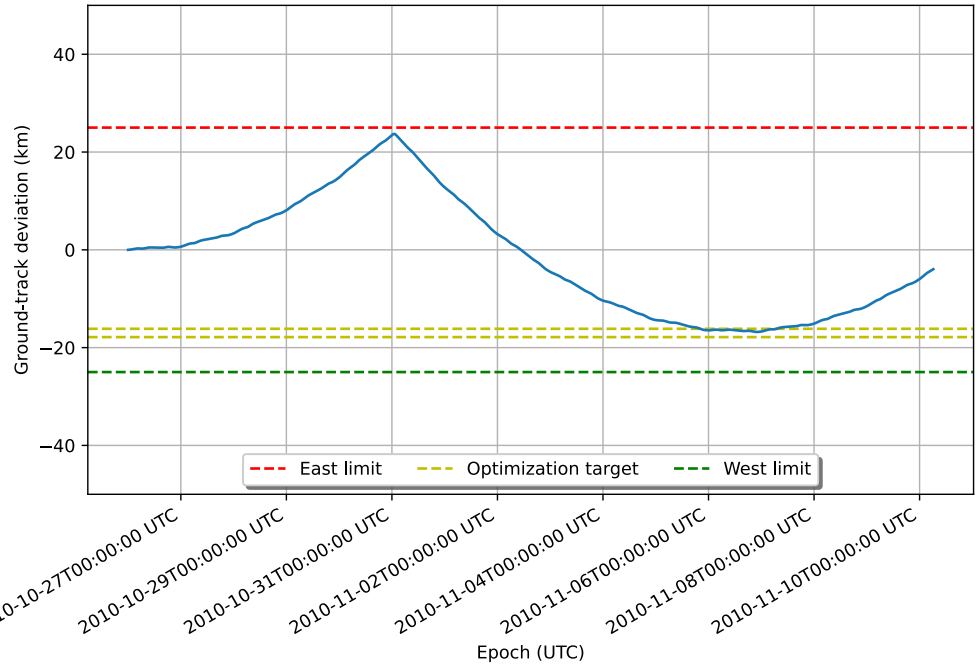
From Figure 3.10 we can confirm the similarity between the two analyses. The nominal activity curve in the Aeolus results show a similar trend respect to the output of the application, although there is a difference in the rate of increment of the Ground-track deviation which implies that different values of solar radiation pressure, thus air density and drag, have been used in the two analyses.

It is possible to further validate the results by comparing the ΔV required for the orbit control from the Aeolus paper, which is between $1.240 m/s$ and $2.780 m/s$ respectively for low and medium solar activity, with the one obtained as output from the app, equal to $1.408 m/s$, denoting the same order of magnitude of the solutions.



(a) Aeolus groud-track deviation analysis from the paper

Test with Aeolus spacecraft data



(b) Evaluation from the app with Aeolus data

Figure 3.10: Comparison between the Aeolus paper and the app's outputs

Chapter 4

Conclusions and future works

The present work showed the potentiality of GODOT software, especially in combination with optimization tools like PyGMO for the optimization of the ΔV required for a mission. The applications developed are capable to analyze a decent range of missions, reducing the work due to the user.

Three use cases have been presented to test the developed applications.

An interplanetary mission to Mars and a Moon free-return mission have been implemented, analyzed and optimized with the trajectory optimization app. Throughout the thesis the preliminary analysis for these type of missions have been explained, as the solution to the Lambert's problem, to obtain an initial guess of a trajectory. Then, the successful optimized trajectories outputted by the application have been presented, showing also the possibility to perform a Monte Carlo analysis.

An application to study the required In-plane station-keeping manoeuvres has been also developed, using Sun-Synchronous orbits as test cases. This application combines the possibility to propagate multiple trajectories with different manoeuvres via GODOT with a bisection algorithm, in order to find the manoeuvre with the best ΔV that fulfil the station-keeping re-

quirements.

A limitation of the applications derives from the type of manoeuvres allowed. Both the applications can analyze missions considering only instant or finite manoeuvres with constant thrust. This prevent the applications to study or optimize manoeuvres working on the engine's throttle, as it is not implemented in the version of GODOT used. Since during the majority of a mission a satellites have more or less constant thrust, the level of detail of the analyses is not much reduced.

Nonetheless, these tools remains valuable instruments to have a closer idea, compared to only a theoretical analysis, of the required ΔV and in turn allows a better sizing of the spacecraft's engines and fuel tanks. Moreover, the ability to present the trajectories with 3D animations via Cosmographia makes the trajectories way more human-readable.

Further studies shall be conducted on the In-plane station-keeping manoeuvre optimization app.

Firstly, the manoeuvres shall be optimized directly by combining PyGMO and GODOT rather than the empirical bisection algorithm, to minimise the ΔV required furthermore. To add on that, different In-plane station-keeping strategies may be studied. For example, a multiple-manoeuvres strategy once the east limit is reached can be implemented, which will allow to choose the arrival time at the west limit.

Moreover, an addendum at the current code can be developed to expand the functionality, like a control of the inclination and LTAN, thus perform out-of-plane manoeuvres.

Finally, to further validate the application, an extensive campaign of comparisons between the application's outputs and already performed missions, like Aeolus, shall be conducted.

Bibliography

[1] European Space Agency, "Godot documentation," 2021. https://godot.io.esa.int/docs/0.7.0/index.html.

[2] T. D. Moyer, "Mathematical formulation of the double precision orbit determination program (DPODP)," tech. rep., Jet Propulsion Laboratory, 1971.

[3] International Astronomical Union, "IAU 2000 resolution," 2000. https://www.iau.org/static/resolutions/IAU2000_French.pdf.

[4] C. Hohenkerk, "Sofa and the algorithms for transformations between time scales and between reference systems," in *Proceedings of the Journées*, pp. 21–24, 2011.

[5] J. Meeus, "Astronomical algorithms," *Richmond*, 1991.

[6] C. H. Acton Jr, "Ancillary data services of NASA's navigation and ancillary information facility," *Planetary and Space Science*, vol. 44, no. 1, pp. 65–70, 1996.

[7] W. M. Folkner, J. G. Williams, D. H. Boggs, R. S. Park, and P. Kuchynka, "The planetary and lunar ephemerides de430 and de431," *Interplanetary Network Progress Report*, vol. 196, no. 1, pp. 42–196, 2014.

[8] H. Curtis, *Orbital mechanics for engineering students*. Butterworth-Heinemann, 2013.

[9] E. Fomalont, "The international celestial reference system," *The Science of Calibration*, vol. 503, p. 177, 2016.

[10] B. A. Archinal, M. F. A'Hearn, E. Bowell, A. Conrad, G. J. Consolmagno, R. Courtin, T. Fukushima, D. Hestroffer, J. L. Hilton, G. A. Krasinsky, *et al.*, "Report of the IAU working group on cartographic coordinates and rotational elements: 2009," *Celestial Mechanics and Dynamical Astronomy*, vol. 109, no. 2, pp. 101–135, 2011.

[11] M. P. Kelly, "Transcription methods for trajectory optimization: a beginners tutorial," *arXiv preprint arXiv:1707.00284*, 2017.

[12] E. Fehlberg, "Classical fifth-, sixth-, seventh-, and eighth-order runge-kutta formulas with stepsize control," tech. rep., National Aeronautics and Space Administration, 1968.

[13] J. H. Verner, "Numerically optimal runge–kutta pairs with interpolants," *Numerical Algorithms*, vol. 53, no. 2, pp. 383–396, 2010.

[14] W. H. Enright, K. R. Jackson, S. P. Nørsett, and P. G. Thomsen, "Interpolants for runge-kutta formulas," *ACM Transactions on Mathematical Software (TOMS)*, vol. 12, no. 3, pp. 193–218, 1986.

[15] J. C. Butcher, "Coefficients for the study of runge-kutta integration processes," *Journal of the Australian Mathematical Society*, vol. 3, no. 2, pp. 185–201, 1963.

[16] E. Fehlberg, "New high-order runge-kutta formulas with step size control for systems of first-and second-order differential equations," *ZAMM-Journal of Applied Mathematics and Mechanics/Zeitschrift für Angewandte Mathematik und Mechanik*, vol. 44, no. S1 S1, pp. T17–T29, 1964.

[17] F. Biscani and D. Izzo, "A parallel global multiobjective framework for optimization: pagmo," *Journal of Open Source Software*, vol. 5, no. 53, p. 2338, 2020.

[18] T. Weise, "Global optimization algorithms-theory and application," *Self-Published Thomas Weise*, vol. 361, 2009.

[19] C. Darwin, *On the Origin of Species by Means of Natural Selection, or Preservation of Favoured Races in the Struggle for Life.* Murray, London, 1859.

[20] D. Kraft, "A software package for sequential quadratic programming," *Forschungsbericht-Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt*, 1988.

[21] D. Kraft, "Algorithm 733: Tomp–fortran modules for optimal control calculations," *ACM Transactions on Mathematical Software (TOMS)*, vol. 20, no. 3, pp. 262–281, 1994.

[22] S. G. Johnson, "The nlopt nonlinear-optimization package." `http://github.com/stevengj/nlopt`.

[23] B. Goodman, F. You, D. Yue, *et al.*, "Northwestern university process optimization open textbook." Chapter Sequential quadratic programming.

[24] A. Wächter and L. T. Biegler, "On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming," *Mathematical programming*, vol. 106, no. 1, pp. 25–57, 2006.

[25] A. Wächter and L. T. Biegler, "Line search filter methods for nonlinear programming: Motivation and global convergence," *SIAM Journal on Optimization*, vol. 16, no. 1, pp. 1–31, 2005.

[26] M. J. Powell, "A direct search optimization method that models the objective and constraint functions by linear interpolation," in *Advances in optimization and numerical analysis*, pp. 51–67, Springer, 1994.

[27] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The computer journal*, vol. 7, no. 4, pp. 308–313, 1965.

[28] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[29] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant, "Array programming with NumPy," *Nature*, vol. 585, pp. 357–362, Sept. 2020.

[30] P. Virtanen, R. Gommers, T. E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S. J. van der Walt, M. Brett, J. Wilson, K. J. Millman, N. Mayorov, A. R. J. Nelson, E. Jones, R. Kern, E. Larson, C. J. Carey, İ. Polat, Y. Feng, E. W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E. A. Quintero, C. R. Harris, A. M. Archibald, A. H. Ribeiro, F. Pedregosa, P. van Mulbregt, and SciPy 1.0 Contributors, "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python," *Nature Methods*, vol. 17, pp. 261–272, 2020.

[31] D. Izzo, "Revisiting Lambert's problem," *Celestial Mechanics and Dynamical Astronomy*, vol. 121, no. 1, pp. 1–15, 2015.

[32] M. Sharaf, A. Saad, and M. Nouh, "Lambert universal variable algorithm," *Arabian Journal for Science and Engineering*, vol. 28, no. 1, pp. 87–98, 2003.

[33] AGI, "B-plane targeting." `https://help.agi.com/stk/11.0.1/Content/gator/eq-bplane.html`.

[34] M. M. Serrano, F. Petrucciani, C. Dietze, and G. Ziegler, "Aeolus orbit control strategy: Analysis and final implementation," in *AIAC18: 18th Australian International Aerospace Congress (2019): HUMS-11th Defence Science and Technology (DST) International Conference on Health and Usage Monitoring (HUMS 2019): ISSFD-27th International Sym-*

posium on Space Flight Dynamics (ISSFD), pp. 1532–1549, Engineers Australia, Royal Aeronautical Society. Melbourne, 2019.

[35] J. Picone, A. Hedin, D. P. Drob, and A. Aikin, "Nrlmsise-00 empirical model of the atmosphere: Statistical comparisons and scientific issues," *Journal of Geophysical Research: Space Physics*, vol. 107, no. A12, pp. SIA–15, 2002.

[36] R. Mugellesi-Dow, D. Kerridge, T. Clark, and A. Thomson, "Solmag: an operational system for prediction of solar and geomagnetic indices.," *Space Debris*, pp. 373–376, 1993.

[37] S. D. Conte and C. deBoor, *Elementary Numerical Analysis: An Algorithmic Approach*. McGraw-Hill, New York, 1972.

[38] R. J. Boain, "A-B-Cs of sun-synchronous orbit mission design," in *14th AAS/AIAA Space Flight Mechanics Conference*, Pasadena, CA : Jet Propulsion Laboratory, National Aeronautics and Space Administration, 02 2004.