POLITECNICO
MILANO 1863

Geoinformatics Engineering
Geoinformatics Project

SimuLEO: a Low Earth Orbit Satellites Simulator

Angelica Iseni
Emma Lodetti

Students: Angelica Iseni (10668862) - Emma Lodetti (10619244)
Supervisor: Prof. Ludovico G. A. Biagi
Academic Year: 2023-2024

Abstract

SimuLEO is an innovative tool designed to simulate Low Earth Orbit (LEO) satellite orbits, offering a robust foundation for future positioning studies.
Traditionally, positioning research has always focused on studying GNSS satellite orbits. In contrast, SimuLEO introduces a new approach, focusing on Low Earth Orbit satellites to create fresh research opportunities in LEO-based positioning.
The tool is composed of a user-friendly dashboard developed in PYTHON (Jupyter Lab), which allows users to design satellite constellations by specifying the number of satellite orbits, the number of satellites per orbit, and the inclination of orbital planes. The tool graphically represents the ground tracks of the desired constellations.
The core computational work is handled by a group of functions written in OCTAVE, responsible for accurately calculating satellite positions in the International Terrestrial Reference Frame (ITRF).
Additionally, SimuLEO includes functionality for satellite visibility analysis. This feature transforms satellite coordinates into local coordinates, enabling users to determine the visibility periods of specific satellites from any given location on Earth's surface throughout the day.
By combining these features, SimuLEO offers a complete platform for simulating and analyzing LEO satellite constellations and orbits, supporting and advancing research in satellite-based positioning systems. This tool could play a crucial role in developing new methods and applications within the field of satellite navigation and positioning.

The project is available at the following repository: **SimuLEO repository**

Contents

1 | Introduction

SimuLEO is a project developed by Emma Lodetti and Angelica Iseni, two final-year students from the Geoinformatics Engineering program. This project was supervised by Professor Ludovico G. A. Biagi in the field of Positioning and Location Based Services.

Low Earth Orbit (LEO) satellites, typically situated at altitudes ranging from 200 to 2,000 kilometers above the Earth's surface, are characterized by their rapid orbit times, often completing a full orbit in approximately 130 minutes. These satellites are increasingly gaining attention due to their potential to enhance global communication networks and positioning systems.

Traditionally, positioning research has depended on simulation tools for GNSS (Global Navigation Satellite System) satellites, which orbit at much higher altitudes (approximately 20,000 kilometers for GPS satellites). While GNSS satellites offer extensive coverage and have long been the backbone of global positioning systems, the closer proximity of LEO satellites to the Earth results in reduced signal latency and stronger signal strength, enhancing the accuracy and reliability of positioning data. Moreover, the rapid movement of LEO satellites allows for more frequent updates of positional data, which is particularly beneficial for applications requiring real-time tracking and high precision.

SimuLEO aims to leverage these advantages by providing a comprehensive tool for simulating and visualizing LEO satellite orbits, creating a strong foundation for future positioning studies.

This document reports the background knowledge on which the tool is developed, its various functions, and how they work, through illustration of the main portions of code. Finally, some results are reported, such as the graphical representation of ground tracks and visibility tables produced by the tool.

2 | Background

2.1. Theoretical Background

Low Earth Orbit (LEO) satellites operate at altitudes ranging from 200 to 2,000 km above the Earth's surface. These satellites are pivotal for various applications, including Earth observation, communications, and weather monitoring. Due to their proximity to Earth, LEO satellites complete an orbit with a period of 128 minutes or less, offering advantages like lower communication latency compared to satellites in higher orbits.

2.1.1. Specifications of LEO Satellites

LEO satellites typically have circular orbits, meaning their semi-major and semi-minor axes are equal, and their eccentricity is nearly zero. They orbit at a velocity of around 7.8 km/sec at an altitude of 500 km. The specific parameters used to determine their positions, known as ephemerides, are crucial for maintaining accurate tracking and control.

The main parameters to consider when describing the orbit of a satellite are the following:

- Orbit inclination i: angle between the orbital plane and the reference equatorial plane.
- Right ascension of the ascending node Ω: angle, on the reference equatorial plane, between the equinox node and the ascending node.
- Perigee argument ω: angle between the intersection of the orbital plane with the equatorial plane and the orbit perigee direction.
- Mean anomaly M_0: angle between the satellite mean motion and the perigee at epoch zero.

2.1.2. Satellite Position and Orbits

The motion of a LEO satellite is governed by the fundamental principles of orbital mechanics. The position of a satellite can be described by the equation (2.1) of motion:

$$\ddot{x}_P(t) = \frac{1}{m_P} \sum_i f_i(x_P, \dot{x}_P, t) \tag{2.1}$$

In this general case, the various forces f_i acting on the satellite, including gravitational and perturbative forces.

2.1.3. Orbital Mechanics for LEO Satellites

Given the circular nature of LEO orbits (eccentricity $e \approx 0$), the calculation of the orbital period T and mean angular velocity n is simplified:

$$T^2 = \frac{4\pi^2 a^3}{GM_E} \tag{2.2}$$

$$n = \frac{2\pi}{T} = \sqrt{\frac{GM_E}{a^3}} \tag{2.3}$$

where a is the semi-major axis that in this case coincides with the radius of the orbit. For a circular orbit, the satellite's position can be described with the following equations:

$$x(t) = r(t)\cos(\psi(t)) \tag{2.4}$$

$$y(t) = r(t)\sin(\psi(t)) \tag{2.5}$$

$$r(t) = a \tag{2.6}$$

The true anomaly $\psi(t)$, which represents the angle between the perigee and the satellite's position, varies linearly with time.

2.1.4. Reference Frame Transformations

To accurately determine a satellite's position, it is necessary to transform its coordinates from the Orbital Reference System (ORS) to the Earth-fixed reference frame (ITRF):

$$\begin{pmatrix} X_S(t) \\ Y_S(t) \\ Z_S(t) \end{pmatrix}_{\text{ITRF}} = R_3(-\Omega)R_1(-i)R_3(-\omega)\begin{pmatrix} x_S(t) \\ y_S(t) \\ 0 \end{pmatrix}_{\text{ORS}} \tag{2.7}$$

In this case, with eccentricity $e = 0$, the orbit is perfectly round and the distance to the central body is the same at all points. Consequently, there is no distinct perigee (nearest point) or apogee (farthest point). As a result, the Perigee Argument is often set to 0 by convention.

The right ascension of the ascending node Ω is calculated as:

$$\Omega(t) = \Omega_0 + \dot{\Omega}(t - t_0) \tag{2.8}$$

where Ω_0 is calculated as follows:

$$\Omega_0 = (i\_orbit - 1) \times \frac{180}{num\_orbits} \tag{2.9}$$

Here, i_orbit is the orbit identification number and num_orbits is the total number of orbits.

The Orbit Inclination (i) in this case is defined by the user and it is constant for every satellite of the considered constellation.

The Perigee Argument (ω) is often set to 0 by convention in the case of circular orbits. The eccentricity $e = 0$ so the orbit is perfectly round and the distance to the central body is the same at all points. Consequently, there is no distinction between perigee and apogee.

Using the ITRF coordinates allows for the accurate representation of a satellite's ground track. The ground track is the projection of the satellite's orbit onto the Earth's surface, showing the path it follows as it moves around the planet. By converting ORS coordinates to ITRF coordinates, one can accurately determine the satellite's position relative to the Earth's surface, enabling the computation of this ground track.

2.1.5. Perturbations and Orbital Adjustments

Low Earth Orbit (LEO) satellites are generally subject to various perturbative forces, including gravitational influences from the Moon and the Sun, solar radiation pressure, and atmospheric drag. These forces can cause deviations from a perfect Keplerian orbit, typically necessitating periodic adjustments to maintain precise orbital parameters. However, our analysis assumes that these perturbative forces are negligible. Consequently, for our calculations and predictions, we rely on idealized Keplerian models without accounting for the minor perturbations that would otherwise require regular adjustments.

2.1.6. PDOP and Satellite Visibility Chart

The PDOP is an important metric that indicates the quality of satellite positioning based on the geometry of the satellite constellation. The visibility of each satellite, determined by examining its local coordinates, plays an important role in this. A satellite is considered visible if its altitude (the third component of the local coordinates) is non-negative. The number and geometric distribution of these visible satellites directly affects the PDOP value: a greater number of well-distributed visible satellites results in a lower PDOP, indicating higher positioning accuracy, whereas fewer or clustered satellites increase the PDOP, reducing accuracy.

To accurately determine the position and visibility of satellites, it is necessary to compute their local coordinates relative to an observer on the Earth's surface. This process involves several key steps:

Geodetic to Cartesian Conversion: The observer's location is specified by its latitude (ϕ_0) and longitude (λ_0). These geodetic coordinates are converted into Cartesian coordinates (x_0, y_0, z_0) to facilitate further calculations. The same conversion is applied to the satellite's position, provided in Cartesian coordinates (x_s, y_s, z_s).

Rotation Matrix: A rotation matrix R is constructed to transform global Cartesian coordinates into a local frame centered at the observer's position. This matrix accounts for the rotation induced by the observer's latitude and longitude, aligning the coordinate system with the observer's local horizon.

Computing Local Coordinates: The difference vector $\mathbf{dx}$ between the satellite's Cartesian coordinates and the observer's coordinates is computed for each satellite. This difference vector is then rotated using the matrix R to obtain the local coordinates of the satellite relative to the observer. Mathematically, the local coordinates **loc_coords** are given by:

$$\mathbf{loc\_coords} = R \cdot \mathbf{dx}$$

where $\mathbf{dx} = [x_s - x_0, y_s - y_0, z_s - z_0]^T$.

Local Coordinates Normalization: For each visible satellite (non-negative altitude), the local coordinates are normalized to construct a matrix A. This normalization ensures that each satellite's contribution to the PDOP calculation is appropriately scaled.

PDOP calculation: The PDOP value is derived from the matrix A, which as stated before contains the normalized local coordinates of all visible satellites. The PDOP is a measure of the geometric strength of the satellite configuration and is calculated as the square root of the trace of the inverse of the $A^T A$ matrix:

$$\text{PDOP} = \sqrt{Tr\left((A^T A)^{-1}\right)}$$

2.2. Preexisting Projects

The integration of Low Earth Orbit (LEO) satellite systems into various applications has become a transformative trend in the field of satellite technology, particularly in Positioning, Navigation, and Timing (PNT) services. Recent advancements have underscored the potential of LEO constellations to enhance the capabilities of traditional satellite systems in multiple domains.

A comprehensive overview of the evolution and current state of LEO satellite systems is provided in [2]. Over the past decade, significant investments have shifted towards LEO-based constellations, moving away from the traditional Medium Earth Orbit (MEO) and Geostationary Earth Orbit (GEO) satellites typically used for PNT services.

The document highlights the successful deployment of several LEO systems, such as Iridium, OneWeb, and Starlink, which offer enhanced communication and Earth observation capabilities. These systems have proven instrumental in delivering high-speed internet and IoT services globally. However, their impact extends beyond communication, offering promising improvements in PNT applications, which traditionally rely on systems like GPS, Galileo, GLONASS, and Beidou. LEO satellites bring several advantages, including lower latency, higher signal power, and more frequent revisit times, which are particularly beneficial for rapid and precise positioning. These attributes make LEO satellites an attractive addition to existing PNT frameworks, potentially revolutionizing navigation and timing accuracy across various industries.

The paper [1], delves into the technical aspects of integrating LEO satellites with Global Navigation Satellite Systems (GNSS) to improve Precise Point Positioning (PPP) and Real-Time Kinematic (RTK) performance. The study simulates the inclusion of 180 LEO satellites to assess their impact on PPP convergence times and positioning accuracy.

Key findings indicate that the addition of LEO satellites significantly enhances the performance of single GNSS systems (e.g., GPS or BeiDou) and combined systems (e.g., GPS/Galileo/GLONASS). The convergence speed of PPP solutions improves a lot with substantial reductions in Time-To-First-Fix (TTFF) for ambiguity resolution (AR) solutions. This integration also boosts the success fix rate and positioning performance in various environmental conditions, including urban canyons and indoor settings. LEO satellites contribute additional observations that enhance satellite geometric distribution and provide more redundant data, mitigating the limitations posed by atmospheric delays and hardware-related errors. These enhancements are crucial for applications requiring high precision and rapid positioning, such as seismic monitoring, precise orbit determination, and real-time navigation.

The SimuLEO project can comprehensively simulate the coverage areas of various satellite constellations. By adjusting critical parameters such as the number of satellites, their respective orbital planes, and the spacing between them, users can design constellations that ensure optimal global coverage. This capability allows for the minimization of coverage gaps, thereby enhancing the overall reliability of service provided by the constellation. Detailed coverage analysis with SimuLEO facilitates the creation of robust satellite networks that cater to diverse geographical and service requirements. Additionally, accurate positioning and navigation largely depend on the geometric distribution of satellites within a constellation. A well-distributed constellation ensures that signals from multiple satellites can be received from different angles, leading to more precise triangulation and reduced positional errors.

One of the significant advantages of integrating LEO satellites into PNT services is the potential for rapid convergence times in Precise Point Positioning (PPP) solutions. SimuLEO can allow this improvement by incorporating additional observations from LEO satellites into the PPP process. This rapid convergence is particularly beneficial for applications requiring quick and precise location fixes, such as autonomous vehicle navigation and emergency response operations.

The role of LEO satellites in providing enhanced time synchronization capabilities is another area where SimuLEO can demonstrate significant improvements. Due to their lower latency and higher signal strength, LEO satellites offer more precise timing information. SimuLEO can model these improvements, showing how LEO constellations can deliver highly accurate time synchronization. This is particularly crucial for applications requiring high synchronization accuracy.

3 | Project developement

3.1. Software requirements

- **Operating System:** the tool is compatible with any operating system. In this case, Windows has been used.
- **Programming Language:** the tool is developed using Python, version `3.11.5`, and Octave, version `9.1.0`. The installation of both Python and GNU Octave is required.
- **Libraries:** the core of the tasks are carried out through the following Python packages:

IPYTHON	version `8.20.0`	OCT2PY	version `5.6.1`
IPYWIDGETS	version `8.1.2`	PANDAS	version `2.2.1`
MATPLOTLIB	version `3.8.4`	PLOTLY	version `5.19.0`
NUMPY	version `1.26.4`	TK	version `8.6.12`

Table 3.1: Packages needed for the project.

  In our case, these libraries have been installed in a proper *Anaconda* `24.5.0` virtual environment.

- **Platforms**: The software is developed in *Jupyter Lab* and GNU Octave, and executed in *Jupyter Lab* for better visualization of results.
- **Hardware Requirements**: The software is able to run on a standard computer system with 8 GB of RAM and a 64-bit processor.
- **Development Environment**: The software is developed using an appropriate Integrated Development Environment (IDE) such as VS Code or Anaconda.

3.2. Dashboard design

The SimuLEO dashboard is split into several sections, each offering different functions for analyzing various aspects of the designed constellation.

1. **Constellation creation:** the user designs its constellation of satellites, by entering the number of orbital planes of the constellation, the number of satellites per orbital plane, and the inclination of the orbital planes with respect to the reference equatorial plane.

2. **Ground tracks plot:** the user can plot the constellation groundntracks both in 2D and 3D plots.

3. **PDOP computation:** the user can compute the PDOP index with respect to a chosen position and time of the day.

4. **Satellites Visibility Chart:** the user can analyze the designed constellation's coverage through the visibility table's usage.

These different tasks can be accomplished in sequence, or independently. Indeed, in each step, the user will be prompted to select a constellation from the saved designs to perform the chosen task.

In the following sections, the various functions of the tool will be described through the main parts of the code.

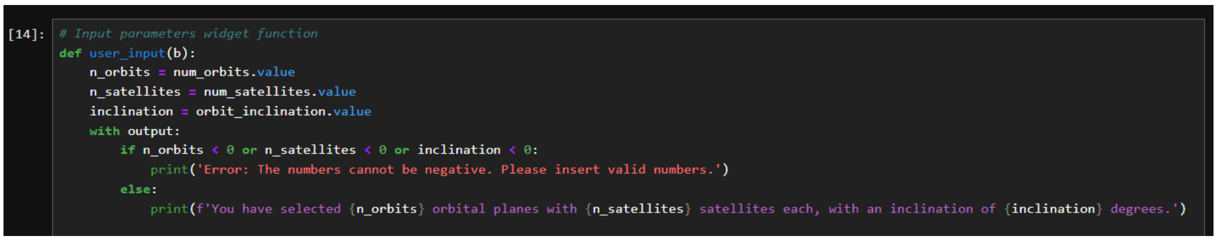
3.2.1. Constellation creation

The user designs the constellation by inputting three different parameters, in the dedicated widgets, as shown in Figure 3.2:

- The number of orbital planes of the constellation
- The number of satellites per orbital plane
- The inclination of the orbital planes with respect to the reference equatorial plane

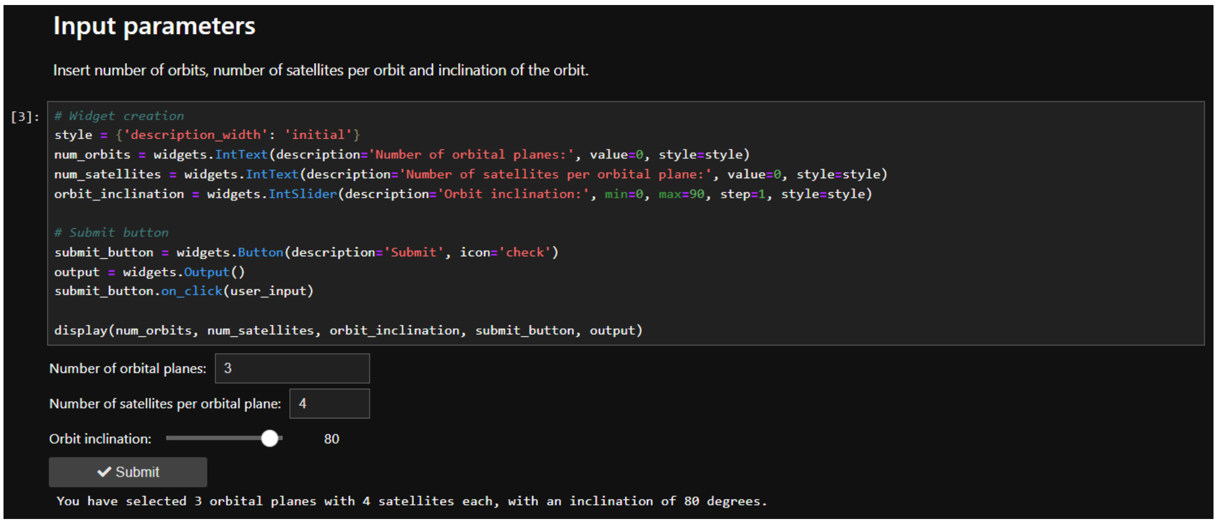
The submission button works thanks to the function in Figure 3.1.

The inputted data are used to compute all the parameters of the constellation: orbit radius (always equal to 7180 km), orbit inclination i, mean anomaly M_0, and right ascension of the ascending node Ω_0. Since M_0 and Ω_0 are changing for each satellite of the constellation, a different *txt* file for each satellite containing these parameters is created. Every *txt* is named after the satellite: **LEOXXYY**, where XX is the orbit number and YY is the satellite number. An example is shown in Figure 3.3. These files are created with the function in Figure 3.4.



```
[14]: # Input parameters widget function
def user_input(b):
    n_orbits = num_orbits.value
    n_satellites = num_satellites.value
    inclination = orbit_inclination.value
    with output:
        if n_orbits < 0 or n_satellites < 0 or inclination < 0:
            print('Error: The numbers cannot be negative. Please insert valid numbers.')
        else:
            print(f'You have selected {n_orbits} orbital planes with {n_satellites} satellites each, with an inclination of {inclination} degrees.')
```

Figure 3.1: Function for parameters submission in the widgets.



Input parameters

Insert number of orbits, number of satellites per orbit and inclination of the orbit.

```
[3]: # Widget creation
style = {'description_width': 'initial'}
num_orbits = widgets.IntText(description='Number of orbital planes:', value=0, style=style)
num_satellites = widgets.IntText(description='Number of satellites per orbital plane:', value=0, style=style)
orbit_inclination = widgets.IntSlider(description='Orbit inclination:', min=0, max=90, step=1, style=style)

# Submit button
submit_button = widgets.Button(description='Submit', icon='check')
output = widgets.Output()
submit_button.on_click(user_input)

display(num_orbits, num_satellites, orbit_inclination, submit_button, output)
```

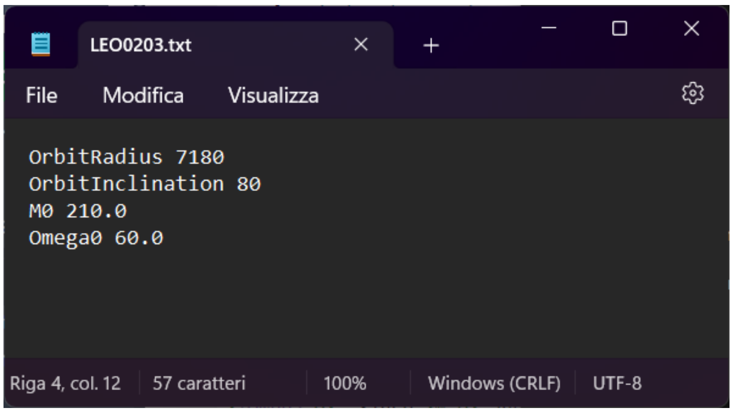
Number of orbital planes: 3

Number of satellites per orbital plane: 4

Orbit inclination: 80

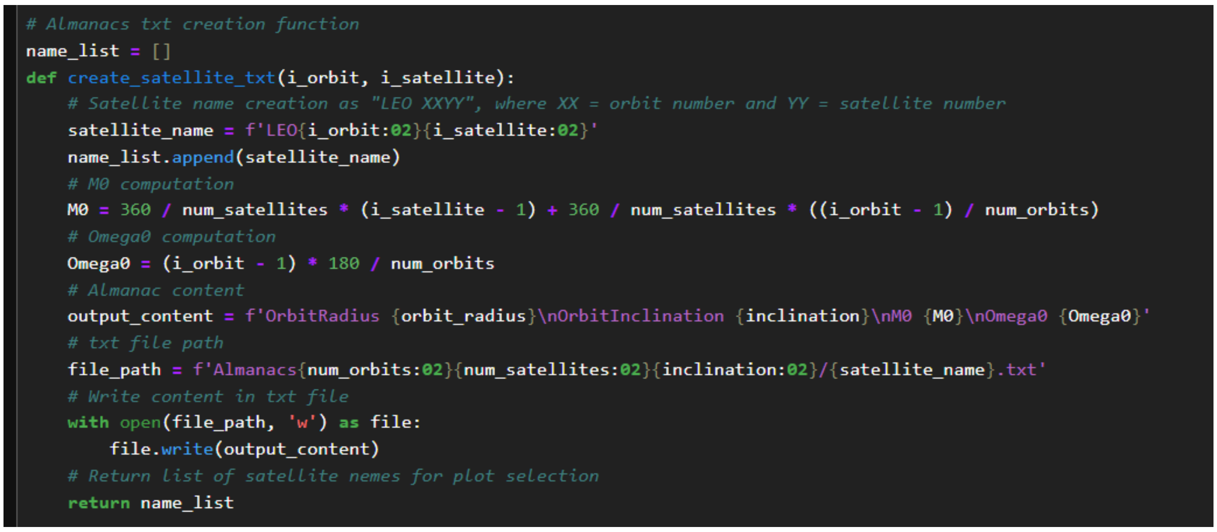
```
You have selected 3 orbital planes with 4 satellites each, with an inclination of 80 degrees.
```

Figure 3.2: Widget for number of orbital planes, number of satellites, and orbital plane inclination insertion. For this example, we are considering a constellation of 4 orbital planes, with 3 satellites each, and an inclination of 80 degrees.



```
OrbitRadius 7180
OrbitInclination 80
M0 210.0
Omega0 60.0
```

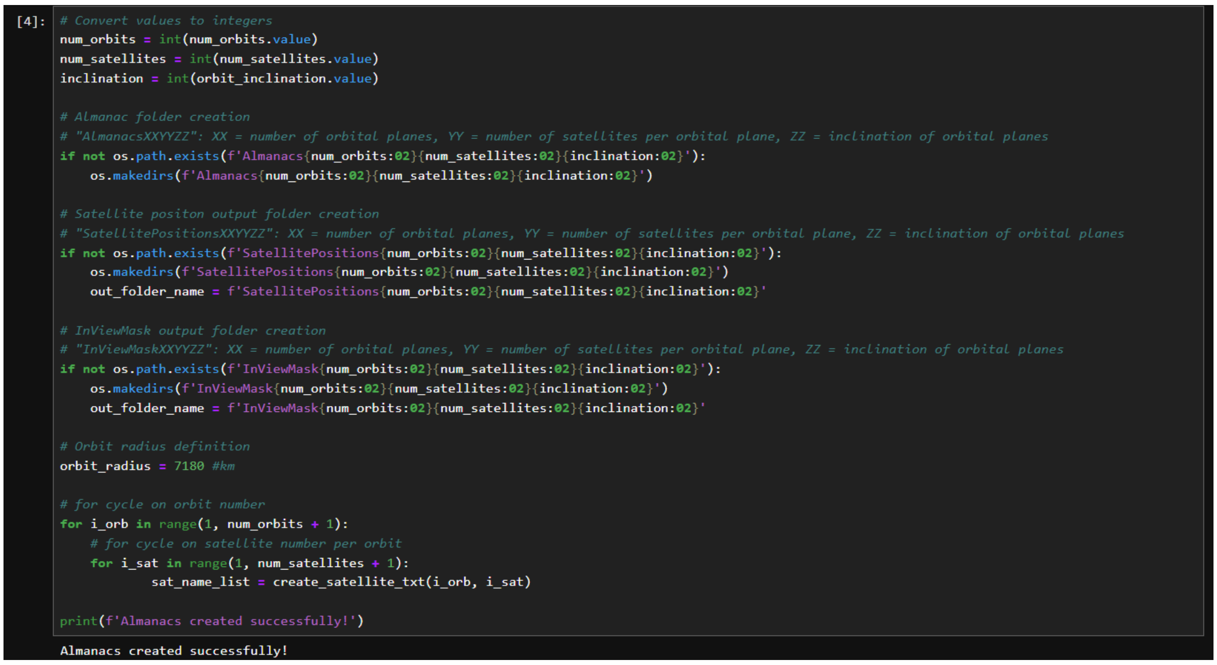
Figure 3.3: Almanac of the satellite LEO0203, the third satellite of the second orbit of the constellation.



```python
# Almanacs txt creation function
name_list = []
def create_satellite_txt(i_orbit, i_satellite):
    # Satellite name creation as "LEO XXYY", where XX = orbit number and YY = satellite number
    satellite_name = f'LEO{i_orbit:02}{i_satellite:02}'
    name_list.append(satellite_name)
    # M0 computation
    M0 = 360 / num_satellites * (i_satellite - 1) + 360 / num_satellites * ((i_orbit - 1) / num_orbits)
    # Omega0 computation
    Omega0 = (i_orbit - 1) * 180 / num_orbits
    # Almanac content
    output_content = f'OrbitRadius {orbit_radius}\nOrbitInclination {inclination}\nM0 {M0}\nOmega0 {Omega0}'
    # txt file path
    file_path = f'Almanacs{num_orbits:02}{num_satellites:02}{inclination:02}/{satellite_name}.txt'
    # Write content in txt file
    with open(file_path, 'w') as file:
        file.write(output_content)
    # Return list of satellite nemes for plot selection
    return name_list
```

Figure 3.4: Function for the *txt* creation (Almanacs). Here all the needed parameters for the constellation are computed.

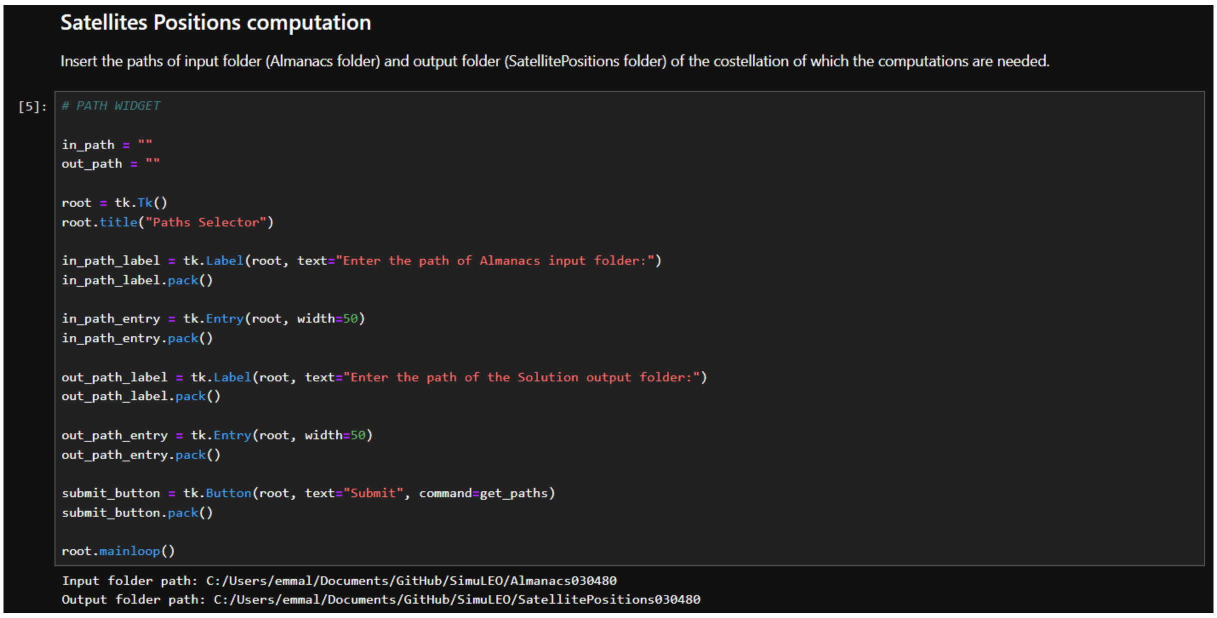
Almanacs *txt* files are contained in a folder called **AlmanacsXXYYZZ** (XX is the number of orbital planes, YY is the number of satellites per orbital plane, ZZ is the inclination of orbital planes). **AlmanacsXXYYZZ** folder and the other folders (*SatellitePositionsXXYYZZ* and *InViewMaskXXYYZZ*) needed for saving useful information for the tool are created by the code snippet in Figure 3.5.



```python
[4]: # Convert values to integers
     num_orbits = int(num_orbits.value)
     num_satellites = int(num_satellites.value)
     inclination = int(orbit_inclination.value)

     # Almanac folder creation
     # "AlmanacsXXYYZZ": XX = number of orbital planes, YY = number of satellites per orbital plane, ZZ = inclination of orbital planes
     if not os.path.exists(f'Almanacs{num_orbits:02}{num_satellites:02}{inclination:02}'):
         os.makedirs(f'Almanacs{num_orbits:02}{num_satellites:02}{inclination:02}')

     # Satellite positon output folder creation
     # "SatellitePositionsXXYYZZ": XX = number of orbital planes, YY = number of satellites per orbital plane, ZZ = inclination of orbital planes
     if not os.path.exists(f'SatellitePositions{num_orbits:02}{num_satellites:02}{inclination:02}'):
         os.makedirs(f'SatellitePositions{num_orbits:02}{num_satellites:02}{inclination:02}')
         out_folder_name = f'SatellitePositions{num_orbits:02}{num_satellites:02}{inclination:02}'

     # InViewMask output folder creation
     # "InViewMaskXXYYZZ": XX = number of orbital planes, YY = number of satellites per orbital plane, ZZ = inclination of orbital planes
     if not os.path.exists(f'InViewMask{num_orbits:02}{num_satellites:02}{inclination:02}'):
         os.makedirs(f'InViewMask{num_orbits:02}{num_satellites:02}{inclination:02}')
         out_folder_name = f'InViewMask{num_orbits:02}{num_satellites:02}{inclination:02}'

     # Orbit radius definition
     orbit_radius = 7180 #km

     # for cycle on orbit number
     for i_orb in range(1, num_orbits + 1):
         # for cycle on satellite number per orbit
         for i_sat in range(1, num_satellites + 1):
                 sat_name_list = create_satellite_txt(i_orb, i_sat)

     print(f'Almanacs created successfully!')

     Almanacs created successfully!
```

Figure 3.5: Function for the creation of the folders *AlmanacsXXYYZZ*, *SatellitePositionsXXYYZZ*, and *InViewMaskXXYYZZ*.

In order to compute and plot the ground tracks of the constellation designed, the position of each satellite in each second of the day is needed.
GNU OCTAVE, a programming language for scientific computing, has been used to make a precise computation. Firstly, the user will be asked to choose which constellation among the designed and saved ones to compute the positions. To do so, the TKINTER (tk) library is used. With the code in Figure 3.6 the widget for path insertion in Figure 3.7 appears. The user can copy and paste the paths of the folders (created in Figure 3.5) of the constellation he/she wants to compute the positions.



Satellites Positions computation

Insert the paths of input folder (Almanacs folder) and output folder (SatellitePositions folder) of the costellation of which the computations are needed.

```
[5]: # PATH WIDGET

in_path = ""
out_path = ""

root = tk.Tk()
root.title("Paths Selector")

in_path_label = tk.Label(root, text="Enter the path of Almanacs input folder:")
in_path_label.pack()

in_path_entry = tk.Entry(root, width=50)
in_path_entry.pack()

out_path_label = tk.Label(root, text="Enter the path of the Solution output folder:")
out_path_label.pack()

out_path_entry = tk.Entry(root, width=50)
out_path_entry.pack()

submit_button = tk.Button(root, text="Submit", command=get_paths)
submit_button.pack()

root.mainloop()

Input folder path: C:/Users/emmal/Documents/GitHub/SimuLEO/Almanacs030480
Output folder path: C:/Users/emmal/Documents/GitHub/SimuLEO/SatellitePositions030480
```

Figure 3.6: Path selector widget creation.

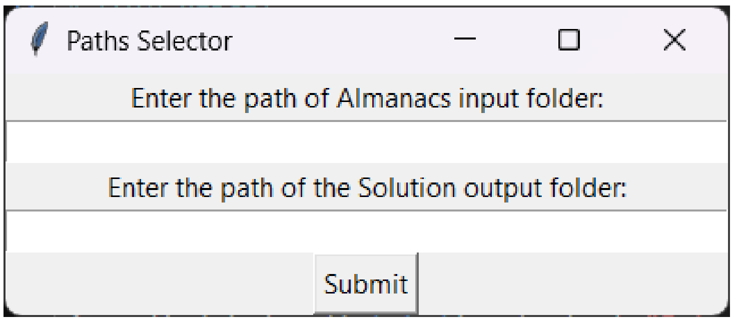
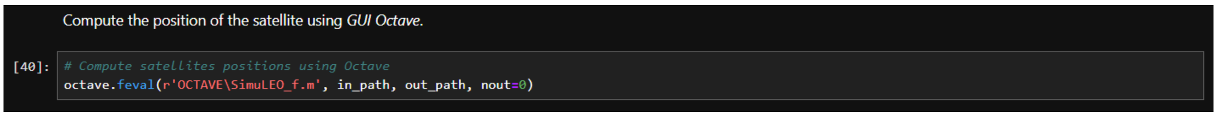

Paths Selector
Enter the path of Almanacs input folder:
Enter the path of the Solution output folder:
Submit

Figure 3.7: Path selector widget.

The actual calculation of the position is performed with the use of the library OCT2PY. This library allows to call m-files and OCTAVE functions from PYTHON. The connection between the tool and the external OCTAVE function is performed by a simple line of code (Figure 3.8).



Compute the position of the satellite using *GUI Octave*.

```
[40]: # Compute satellites positions using Octave
      octave.feval(r'OCTAVE\SimuLEO_f.m', in_path, out_path, nout=0)
```

Figure 3.8: Call Octave function from Python tool.

Octave function for positions computation

The main function that computes the position of each satellite of the chosen constellation for each second of the day is presented in Listing 1. `SimuLEO_f` takes as input the path inserted by the user in the box shown in Figure 3.7.

```
function [] = SimuLEO_f(InputFolderPath,OutputFolderPath)

    % Read data from txt files and compute position for each second in a day for each
       satellite

    % List all files in the folder
    files = dir(fullfile(InputFolderPath, '*.txt'));

    % Initialize vector of epochs
    t_0 = 0; %[sec]
    t_end = 24*3600; %[sec]
    D_t = 1; %[sec]
    t = t_0:D_t:t_end;

    % Loop through each file in the folder
    for i = 1:length(files)
        % Get the file name
        InputFileName = files(i).name;

        % Create the full file path
        InputFilePath = fullfile(InputFolderPath, InputFileName);

        % Read data
        [OrbitRadius,OrbitInclination,M0,Omega0] = ReadData(InputFilePath);

        % Compute ITRF positions each second of the day
        [ITRF_geod] =
           ITRF_positions(t,t_0,t_end,D_t,OrbitRadius,OrbitInclination,M0,Omega0);

        % Save position matrix in a txt file in the output folder
        SavePositions(ITRF_geod, InputFileName, OutputFolderPath);

    end

end
```

Listing 1: Main function: takes as input the paths of the folder inputted in the dashboard.

```
function [OrbitRadius,OrbitInclination,M0,Omega0] = ReadData(files_path)

    fid = fopen(files_path, 'r');

    % Read data from file
    data = fscanf(fid, '%*s %f');

    % Close file
    fclose(fid);

    % Variable extraction
    OrbitRadius = data(1);
    OrbitInclination = data(2);
    M0 = data(3);
    Omega0 = data(4);

end
```

Listing 2: Read data function: reads data from *txt* files.

```
function [ITRF_geod] = ITRF_positions(t,t_0,t_end,D_t,r,o_i,M0,Omega0)

    % Inizialize constants
    OmegaEdot = 7.2921151467e-05; %(rad/s)
    GMe = 3.986005e+14; %(m3/s2)

    % Compute positions in ITRF, X, Y, Z

    % Initialize vector of positions
    x_t = zeros(1,length(t));
    y_t = zeros(1,length(t));
    W = zeros(1,length(t));

    % Convert radius in meters
    r = r*1000; %(m)

     % Compute the mean angular velocity
    n = sqrt(GMe/r^3);

    % Fill the previous vectors
    i = 1;
    for Dt = t_0 : D_t : (t_end - t_0)      % Dt is the counter
        % compute M_t
           M_t = M0 + n*Dt;
        % compute W(t)
           W(i)= Omega0 - OmegaEdot*Dt;
        % compute x(t)
           x_t(i) = r*cos(M_t);
        % compute y(t)
           y_t(i) = r*sin(M_t);
        i = i+1;
    end
    ORS = [x_t' y_t' zeros(length(t), 1)]; % axis z is 0 because the orbital plane
                                           % lays on the x,y plane

    % Fill the three rotation matrices and rotate from ORS to ITRF

    ITRF = zeros(length(t), 3);
    ITRF_geod = zeros(length(t), 3);

    for i = 1:length(t)

        R1 = [1 0 0 ; 0 cos(o_i*pi/180) -sin(o_i*pi/180); 0 sin(o_i*pi/180)
           cos(o_i*pi/180)];    % o_i = OrbitInclination
        R31 = [cos(W(i)) -sin(W(i)) 0; sin(W(i)) cos(W(i)) 0; 0 0 1];
        R32 = [1 0 0; 0 1 0; 0 0 1];
        R = R31*R1*R32;

        X_ORS = ORS(i, :);
        X_ITRF = R*X_ORS';
        ITRF(i,:) = X_ITRF;

        % From global Cartesian to Geodetic
        [lat, lon, h] = Cart2Geod(X_ITRF(1),X_ITRF(2),X_ITRF(3));

        % From radiants to degrees
        lat = lat*180/pi;
        lon = lon*180/pi;

        % Rephase latitude in [-90;90] interval
        if lat > 90
           lat = lat - 180;
        end

        if lat < -90
           lat = lat + 180;
        end

        ITRF_geod(i, :) = [lat, lon, h];

    end

end
```

Listing 3: This function computes the positions of each satellite for each second of the day in ITRF coordinates.

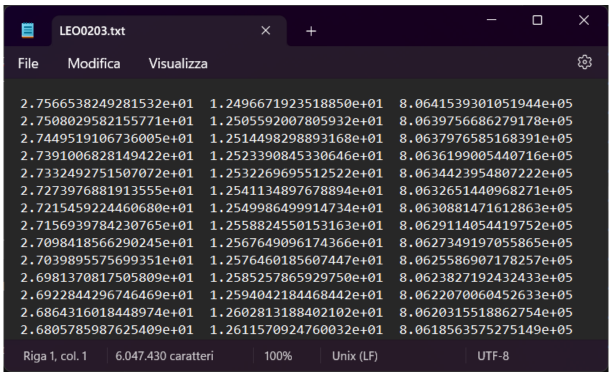
Function in Listing 1 reads data from the Almanacs (Figure 3.3), through ReadData function (Listing 2), computes the position in each second of the day (Listing 3) and saves (Listing 5) the positions during the day for each satellite in a *txt* file in the folder **SatellitePositionsXXYYZZ**. An example of *txt* file is pictured in Figure 3.9.

```
function [Phi,Lambda,h] = Cart2Geod(X,Y,Z)

    % Computation of the auxiliary quantities
    r = sqrt(X^2 + Y^2);
    a = 6378137.0; % major semiaxis
    f = 1/298.257222100882711243; % flattening: a-b/a
    b = a - a*f; % minor semiaxis
    e2 = 2*f - f^2;
    eb2 = (a^2-b^2)/(b^2);
    e = sqrt(e2); % eccentricity
    psi = atan2 (Z , (r * sqrt(1 - e2)));

    % Computation of geodetic coordinates [rad]
    Lambda = atan2 (Y,X);
    Phi = atan2 ( ( Z + (eb2 * b * (sin(psi))^3)) , (r - (e2 * a * (cos(psi))^3)));
    Rn = a / sqrt (1- e2 * (sin(Phi))^2);
    h = r / cos(Phi) - Rn; %[m]

end
```

Listing 4: This function converts Cartesian coordinates into Geodetic coordinates.

```
function [] = SavePositions(PositionMatrix, file_name, OutputFolderPath)

    % Define the file name for the output text file
    FileName = file_name;

    % Create the full file path
    FilePath = fullfile(OutputFolderPath, FileName);

    % Save the position matrix to a text file
    save(FilePath, 'PositionMatrix', "-ascii","-double","-tabs");

end
```

Listing 5: This function saves the positions in *txt* files in the folder SatellitePositionsXXYYZZ.



```
2.7566538249281532e+01  1.2496671923518850e+01  8.0641539301051944e+05
```

Figure 3.9: Positions of the satellite LEO0203. The three columns correspond respectively to latitude, longitude, and height.

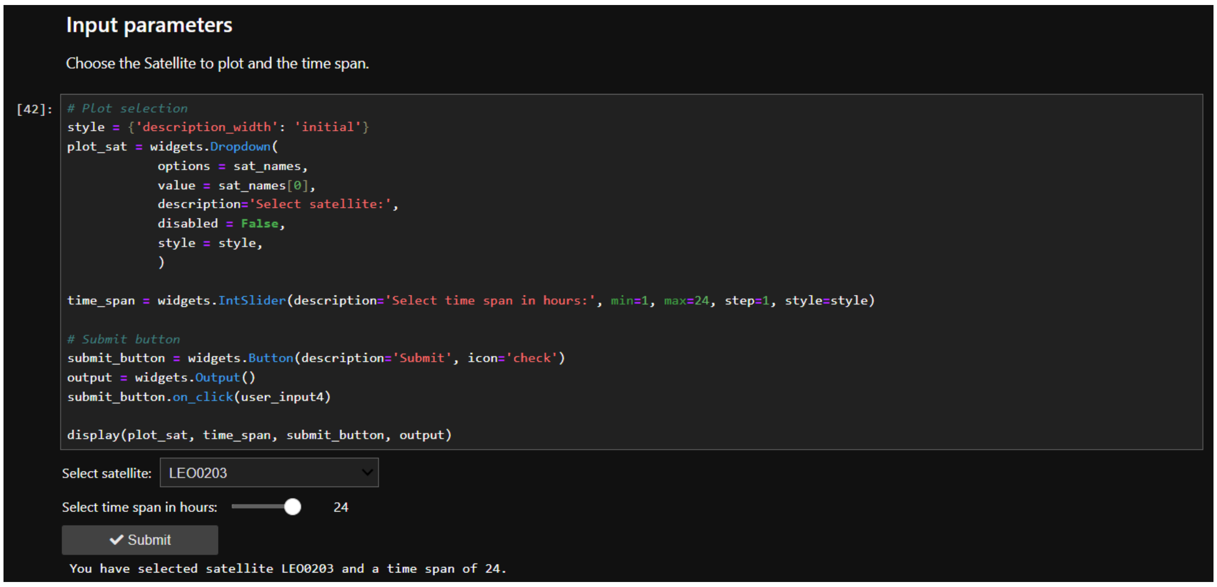
3.2.2. Ground tracks plot

After the computation of the position of each satellite of the desired constellation, it is possible to plot the ground tracks of one or more satellites of the constellation on the Earth's surface.

First, users select the constellation for which they want to plot the ground tracks using the path selector shown Figure 3.7. In this case, the selection of the folder **SatellitePositionsXXYYZZ** only is needed. The positions of that constellation must already have been calculated following the procedure in subsection 3.2.1.

The tool implements two different types of plots, using PLOTLY library:

- **Miller projection** ground tracks plot: the user selects the satellite and the period for which to plot the ground tracks, through the widget in Figure 3.10. The lines are plotted on a 2D map with Miller projection.
- **Globe** ground tracks plot: the ground tracks are plotted on a 3D map of the Earth.
  - **Single satellite:** the ground tracks of the same satellite introduced in the widget in Figure 3.10 are plotted in the same period.
  - **Multiple satellites**: the ground tracks of all the satellites of the chosen constellation are plotted within a two-hour time span.



Input parameters

Choose the Satellite to plot and the time span.

```
[42]: # Plot selection
style = {'description_width': 'initial'}
plot_sat = widgets.Dropdown(
            options = sat_names,
            value = sat_names[0],
            description='Select satellite:',
            disabled = False,
            style = style,
            )

time_span = widgets.IntSlider(description='Select time span in hours:', min=1, max=24, step=1, style=style)

# Submit button
submit_button = widgets.Button(description='Submit', icon='check')
output = widgets.Output()
submit_button.on_click(user_input4)

display(plot_sat, time_span, submit_button, output)
```

Select satellite: LEO0203

Select time span in hours: 24

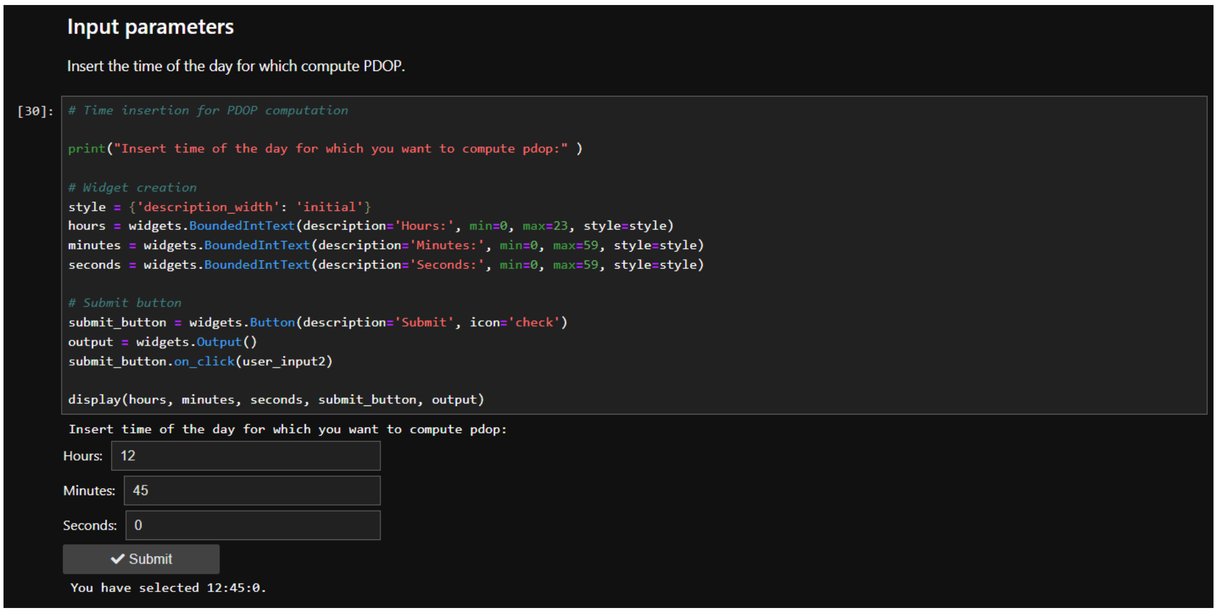
```
You have selected satellite LEO0203 and a time span of 24.
```

Figure 3.10: Widget for satellite and period choice (in hours).

The resulting maps are shown in chapter 4, in Figure 4.1, Figure 4.2 and Figure 4.3.

3.2.3. PDOP computation

In order to calculate the Position Dilution of Precision (PDOP) index, the user needs to enter the time and location to be considered in the dedicated widgets, respectively in Figure 3.11 and Figure 3.12.



Input parameters

Insert the time of the day for which compute PDOP.

```
[30]: # Time insertion for PDOP computation

print("Insert time of the day for which you want to compute pdop:" )

# Widget creation
style = {'description_width': 'initial'}
hours = widgets.BoundedIntText(description='Hours:', min=0, max=23, style=style)
minutes = widgets.BoundedIntText(description='Minutes:', min=0, max=59, style=style)
seconds = widgets.BoundedIntText(description='Seconds:', min=0, max=59, style=style)

# Submit button
submit_button = widgets.Button(description='Submit', icon='check')
output = widgets.Output()
submit_button.on_click(user_input2)

display(hours, minutes, seconds, submit_button, output)
```

```
Insert time of the day for which you want to compute pdop:
```

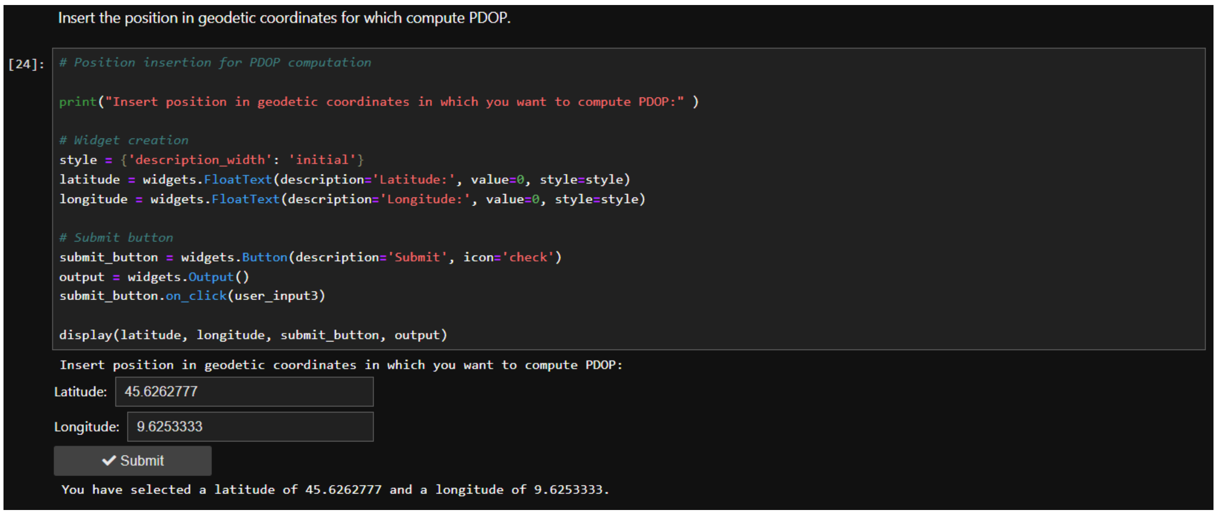
Hours: 12

Minutes: 45

Seconds: 0

```
You have selected 12:45:0.
```

Figure 3.11: Widget for time insertion for PDOP calculation.



Insert the position in geodetic coordinates for which compute PDOP.

```
[24]: # Position insertion for PDOP computation

print("Insert position in geodetic coordinates in which you want to compute PDOP:" )

# Widget creation
style = {'description_width': 'initial'}
latitude = widgets.FloatText(description='Latitude:', value=0, style=style)
longitude = widgets.FloatText(description='Longitude:', value=0, style=style)

# Submit button
submit_button = widgets.Button(description='Submit', icon='check')
output = widgets.Output()
submit_button.on_click(user_input3)

display(latitude, longitude, submit_button, output)
```

```
Insert position in geodetic coordinates in which you want to compute PDOP:
```

Latitude: 45.6262777

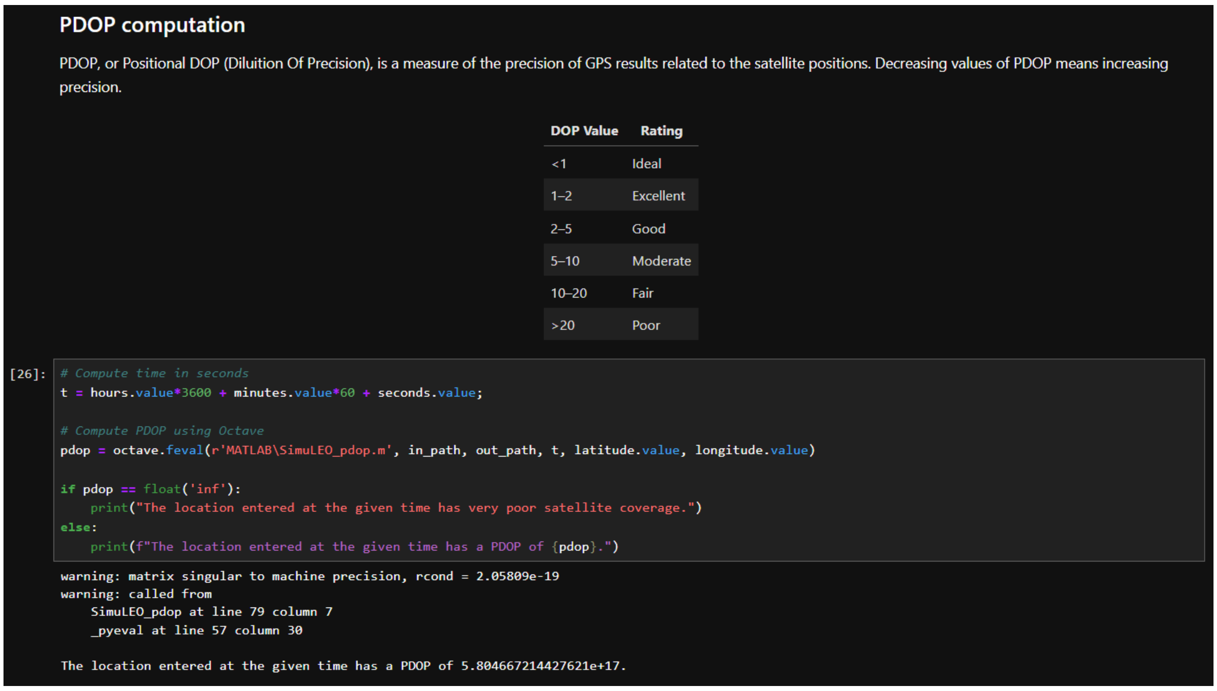
Longitude: 9.6253333

```
You have selected a latitude of 45.6262777 and a longitude of 9.6253333.
```

Figure 3.12: Widget for position insertion for PDOP calculation.

The user must select the constellation for which to perform the calculations using the path selector widget in Figure 3.7. In this case, the path of the folders **SatellitePositionsXXYYZZ** and **InViewMaskXXYYZZ** are needed.

The computation of the PDOP is pictured in Figure 3.13.
As in subsection 3.2.1, similarly here, a connection between the tool and an external OCTAVE function is implemented using (Figure 3.8).



PDOP computation

PDOP, or Positional DOP (Diluition Of Precision), is a measure of the precision of GPS results related to the satellite positions. Decreasing values of PDOP means increasing precision.

DOP Value	Rating
<1	Ideal
1–2	Excellent
2–5	Good
5–10	Moderate
10–20	Fair
>20	Poor

```
[26]: # Compute time in seconds
      t = hours.value*3600 + minutes.value*60 + seconds.value;

      # Compute PDOP using Octave
      pdop = octave.feval(r'MATLAB\SimuLEO_pdop.m', in_path, out_path, t, latitude.value, longitude.value)

      if pdop == float('inf'):
          print("The location entered at the given time has very poor satellite coverage.")
      else:
          print(f"The location entered at the given time has a PDOP of {pdop}.")
```

```
warning: matrix singular to machine precision, rcond = 2.05809e-19
warning: called from
    SimuLEO_pdop at line 79 column 7
    _pyeval at line 57 column 30

The location entered at the given time has a PDOP of 5.804667214427621e+17.
```

Figure 3.13: In this code, the PDOP index is computed by connecting the *Juypyter* dashboard to the GNU OCTAVE code, with `octave.feval` function.

OCTAVE function for PDOP computation

The function `SimuLEO_pdop`, in Listing 6, takes as input the paths inserted by the user and the time and the position for which to compute the index.
This code (function in Listing 10) also fills the folder **InViewMaskXXYYZZ** with a series of *txt* files, one for each satellite of the constellation, containing a binary vector of 0 and 1, with length equal to the number of seconds per day: **1**, if the satellite is **in view** in the current second with respect to the inputted time and position; **0**, if it is **not in view**. These files will be useful in the next task described in subsection 3.2.4.

Function in Listing 7 reads the positions of each satellite of the chosen constellation from the *txt* in the *SatellitePositionsXXYYZZ* folder. Then, the local coordinates of the satellite with respect to the coordinates of the point in time are computed with `local_coordinates` function (Listing 9). IN-VIEW/NOT-IN-VIEW masks are saved in the **InViewMaskXXYYZZ** for each satellite of the constellation. Finally, PDOP is computed and passed as the output of the function.

```
function [pdop] = SimuLEO_pdop(InputFolderPath,OutputFolderPath,t,phi0,lambda0)

    % Convert phi and lambda in radians
    phi_0 = phi0 * pi/180;
    lambda_0 = lambda0 * pi/180;

    % List all files in the folder
    files = dir(fullfile(InputFolderPath, '*.txt')); % SatellitePosition folder

    % Initialize matrices and vectors
    A = [];
    LC_normalized = [];

    % Loop through each file in the folder
    for i = 1:length(files)

        % Get the file name
        InputFileName = files(i).name;

        % Create the full file path
        InputFilePath = fullfile(InputFolderPath, InputFileName);

        % Read positions of satellite i
        [phi_s,lambda_s,h_s] = ReadPositions(InputFilePath);

        % Convert from geodetic coordinates to cartesian coordinates
        [x_s,y_s,z_s] = Geod2Cart(phi_s,lambda_s,h_s);

        % Compute local coordinates of the satellite with respect to the coordinates
           of the point in time
        [loc_coords] = local_coordinates(phi_0,lambda_0,x_s,y_s,z_s);

        % Save mask SATELLITE IN VIEW / SATELLITE NOT IN VIEW
        SaveMask(loc_coords, InputFileName, OutputFolderPath);

        % Check if satellite i is or isn't in view at given time t and normalize its
           coordinates
        if (loc_coords(t,3)) >= 0
            norm_LC = norm(loc_coords(t,:)); % L2 norm
            LC_normalized = - (loc_coords(t,:) / norm_LC);
            A = [A ; LC_normalized];
        end

    end

    % Compute pdop
    A = [A ones(size(A,1),1)];
    N = A'*A;
    K = inv(N);
    pdop = trace(K);

end
```

Listing 6: Main function: takes as input the paths of the folders, the time and the position inputted in the dashboard.

```
function [phi_s,lambda_s,h_s] = ReadPositions(files_path)

    fid = fopen(files_path, 'r');

    % Read data from file
    data = fscanf(fid, '%f %f %f', [3 Inf]);

    % Close file
    fclose(fid);

    % Transpose matrix
    data = data';

    % Coordinates extraction
    phi_s = data(:,1);
    lambda_s = data(:,2);
    h_s = data(:,3);

end
```

Listing 7: Read satellite positions function: reads data from txt files contained in *SatellitePositionsXXYYZZ*.

```
function [X,Y,Z] = Geod2Cart(phi,lambda,h)

    a = 6378137; %major semiaxis
    e = 0.0818191908426215; %eccentricy
    b = a*sqrt(1-e^2); %minor semiaxis

    % Convert from degree to radiants
    phi = phi*pi/180;
    lambda = lambda*pi/180;

    % Initialize coordinates vectors
    X = zeros(length(phi), 1);
    Y = zeros(length(phi), 1);
    Z = zeros(length(phi), 1);

    for i = 1:length(phi)

        Rn = a/sqrt(1-(e^2*(sin(phi(i)))^2));

        % Compute coordinates
        X(i) = (Rn+h(i))*cos(phi(i))*cos(lambda(i));
        Y(i) = (Rn+h(i))*cos(phi(i))*sin(lambda(i));
        Z(i) = (Rn*(1-e^2)+h(i))*sin(phi(i));

    end

end
```

Listing 8: This function converts Geodetic coordinates into Cartesian coordinates.

```
function [loc_coords] = local_coordinates(phi_0,lambda_0,x_s,y_s,z_s)

    % Convert point coordinates from geodetic to cartesian
    [x_0,y_0,z_0] = Geod2Cart(phi_0,lambda_0,0);

    R = [-sin(lambda_0) cos(lambda_0) 0; -sin(phi_0)*cos(lambda_0)
       -sin(phi_0)*sin(lambda_0) cos(phi_0); cos(phi_0)*cos(lambda_0)
       cos(phi_0)*sin(lambda_0) sin(phi_0)];

    % Initialize vector of positions
    loc_coords = zeros(length(x_s),3);

    for i = 1 : length(x_s)
        dx = [x_s(i)-x_0; y_s(i)-y_0; z_s(i)-z_0];
        loc_coords_vec = R*dx;
        loc_coords(i,:) = loc_coords_vec';
    end

end
```

Listing 9: This function computes the local coordinates of the satellite with respect to the position of the point inserted by the user.

```
function [] = SaveMask(local_coordinates, file_name, OutputFolderPath)

    % Define the file name for the output text file
    FileName = file_name;

    % Create the full file path
    FilePath = fullfile(OutputFolderPath, FileName);

    % Initialize mask matrix
    mask = zeros(length(local_coordinates(:,3)),1);

    % Create mask SATELLITE IN VIEW / SATELLITE NOT IN VIEW
    for i = 1 : length(local_coordinates(:,3))
        if local_coordinates(i,3) > 0
            mask(i) = 1;
        else
            mask(i) = 0;
        end
    end

    spy(mask);
    % Save the position matrix to a text file
    save(FilePath, 'mask', "-ascii", "-tabs");

end
```

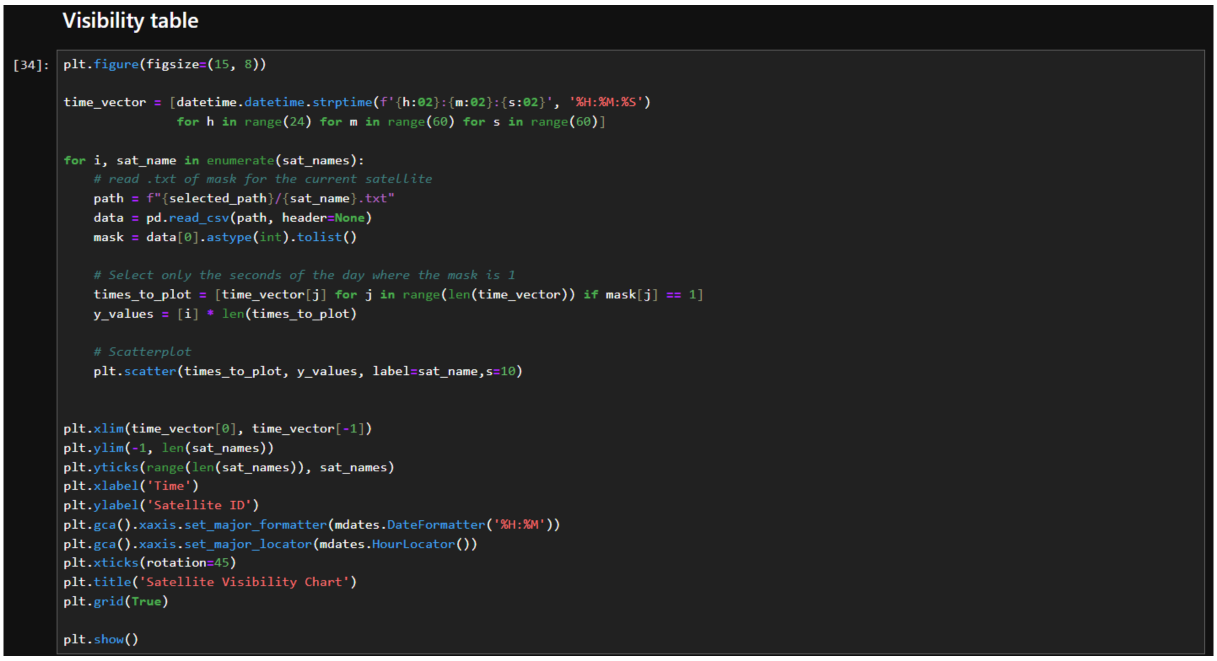
Listing 10: This function saves the IN-VIEW/NOT-IN-VIEW masks in the folder *InViewMaskXXYYZZ* for each satellite.

3.2.4. Satellite Visibility Chart

SimuLEO tool allows to visualize how the visibility of the satellites of the constellation changes during the day, through a **Visibility Table**.

As in the previous tasks, the first step is the constellation selection through path insertion (path selector in Figure 3.7). In this case, only the path of the **InViewMaskXXYYZZ** folder is needed.

The Visibility Chart is computed with the code snippet in Figure 3.14.
The resulting table are shown in chapter 4, Figure 4.4



Visibility table

```
[34]: plt.figure(figsize=(15, 8))

      time_vector = [datetime.datetime.strptime(f'{h:02}:{m:02}:{s:02}', '%H:%M:%S')
                     for h in range(24) for m in range(60) for s in range(60)]

      for i, sat_name in enumerate(sat_names):
          # read .txt of mask for the current satellite
          path = f"{selected_path}/{sat_name}.txt"
          data = pd.read_csv(path, header=None)
          mask = data[0].astype(int).tolist()

          # Select only the seconds of the day where the mask is 1
          times_to_plot = [time_vector[j] for j in range(len(time_vector)) if mask[j] == 1]
          y_values = [i] * len(times_to_plot)

          # Scatterplot
          plt.scatter(times_to_plot, y_values, label=sat_name,s=10)


      plt.xlim(time_vector[0], time_vector[-1])
      plt.ylim(-1, len(sat_names))
      plt.yticks(range(len(sat_names)), sat_names)
      plt.xlabel('Time')
      plt.ylabel('Satellite ID')
      plt.gca().xaxis.set_major_formatter(mdates.DateFormatter('%H:%M'))
      plt.gca().xaxis.set_major_locator(mdates.HourLocator())
      plt.xticks(rotation=45)
      plt.title('Satellite Visibility Chart')
      plt.grid(True)

      plt.show()
```

Figure 3.14: Widget for time insertion for PDOP calculation.

4 | Results

The SimuLEO tool enables the generation of plots that collectively offer a robust way to analyze and interpret satellite behavior, providing valuable insights into satellite trajectories, coverage patterns, and overall constellation performance. By visualizing this data, users can make informed decisions regarding satellite operations, optimize coverage, and enhance the effectiveness of satellite-based applications.

4.1. Individual satellite trajectory over time

The first and the second plots allow users to select a specific satellite and a time span (in hours) to visualize its trajectory respectively in 2D (Figure 4.1) and 3D (Figure 4.2). The plots display the satellite's path over the selected period, highlighting its positions at each second of the day. This visualization aids users in comprehending the movement and positioning of a single satellite over a defined period. This is valuable for analyzing the satellite's behavior and predicting its future locations.

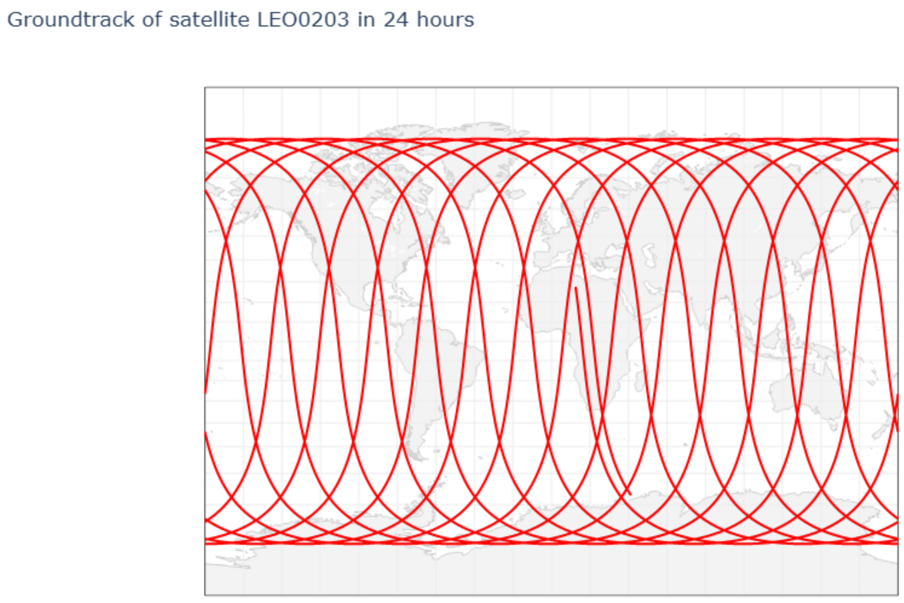


Figure 4.1: Ground track of satellite LEO0203 in 24 hours in a 2D visualization considering a Miller Projection

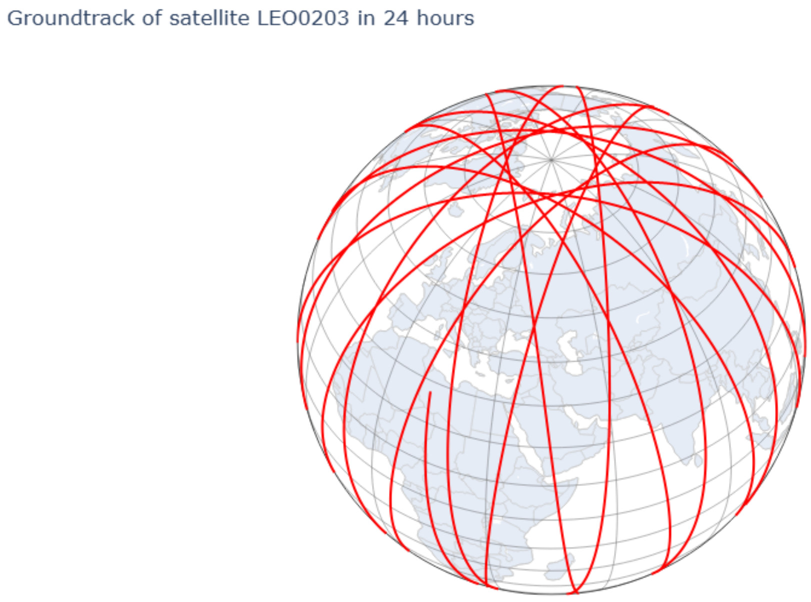


Figure 4.2: Ground track of satellite LEO0203 in 24 hours in a 3D visualization

4.2. Ground tracks of all satellites in a constellation

The third plot (Figure 4.3) provides a view of the ground tracks of all satellites within a constellation over **2 hours**. Each satellite is represented by a different color, making it easy to distinguish between their trajectories. This plot is particularly useful for observing the overall coverage and coordination of the satellite constellation, which can aid in optimizing satellite deployment and ensuring consistent coverage over desired areas.

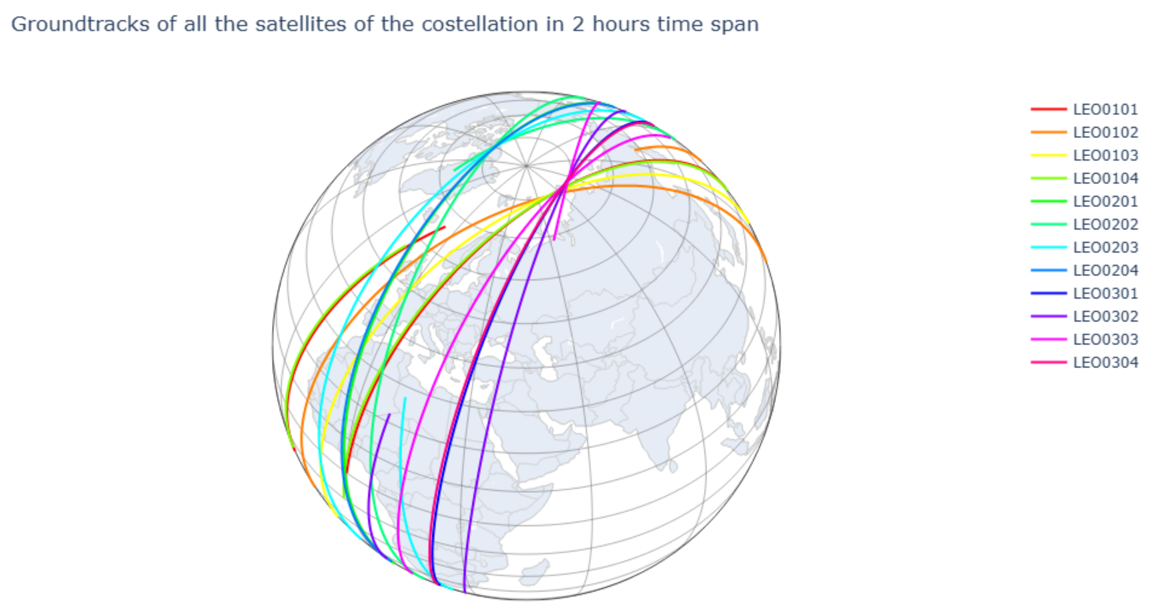


Figure 4.3: Ground tracks of all the satellites of the constellation with 3 orbital planes, 4 satellites for each orbital plane and an inclination of 80°in 2 hours time span

4.3. Satellite Visibility Chart

The Satellite Visibility Chart (Figure 4.4) provided by SimuLEO offers a comprehensive visualization of the visibility of each satellite in the constellation with respect to the user's specified position. This plot details which satellites are in view at each second of the day. The x-axis represents the 24-hour time span, while the y-axis lists the satellite IDs. Each data point on the chart indicates a second during which a particular satellite is visible from the user's location. This detailed temporal resolution allows users to precisely determine periods of satellite visibility, facilitating optimal scheduling for data collection and communication tasks. Additionally, this chart helps identify coverage gaps and overlaps within the constellation, enabling performance analysis and strategic planning for constellation management. The Satellite Visibility Chart is a valuable tool for users needing detailed insights into satellite coverage and visibility patterns throughout the day.

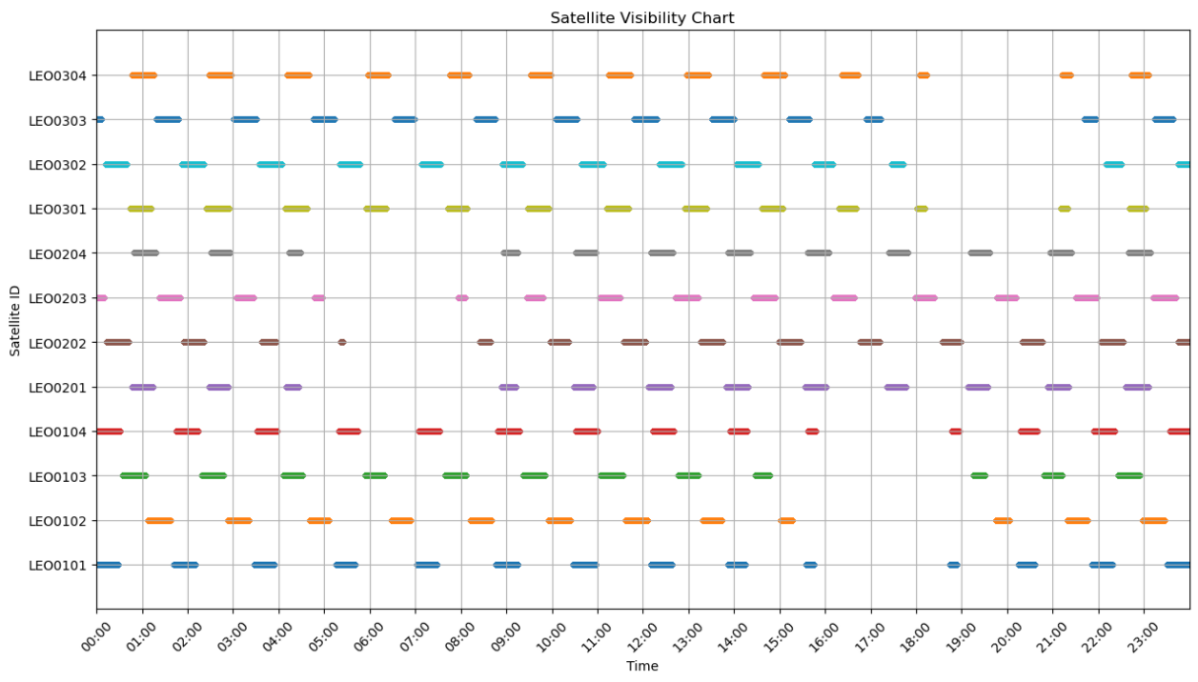


Figure 4.4: Visibility Chart of all the satellites of the constellation with 3 orbital planes, 4 satellites for each orbital plane and an inclination of 80°during the entire day

Moreover, for a given location input by the user, the PDOP index is computed. PDOP, or Positional DOP (Dilution Of Precision), is a measure of the precision of GPS results related to the satellite positions. Decreasing values of PDOP indicate increasing precision.

The PDOP values can be interpreted as follows:

DOP Value	**Rating**
< 1	Ideal
1–2	Excellent
2–5	Good
5–10	Moderate
10–20	Fair
> 20	Poor

Table 4.1: PDOP Value Ratings

5 | Conclusions and future developments

The development and implementation of SimuLEO have successfully demonstrated the potential of simulating and visualizing Low Earth Orbit (LEO) satellite constellations. This project has highlighted the advantages of LEO satellites in providing enhanced precision and reliability in positioning data, which is crucial for various applications requiring real-time tracking and high precision. The project's outcomes include a user-friendly dashboard that allows for the creation, visualization, and analysis of satellite constellations, offering valuable insights into satellite trajectories, coverage patterns, and overall constellation performance.

Despite the significant progress made, there are several areas for future development to further enhance the capabilities and utility of SimuLEO. Firstly, expanding the tool to support more complex orbital dynamics and perturbations could provide more accurate simulations. Additionally, integrating real-time data from existing LEO satellites would allow for more dynamic and responsive modeling, offering users an even more robust platform for planning and analysis.

Furthermore, expanding the tool's applicability to include other types of satellites, such as Medium Earth Orbit (MEO) and Geostationary Earth Orbit (GEO) satellites, could provide a comprehensive solution for satellite constellation management. This would allow users to compare and integrate different satellite systems, optimizing their overall space-based infrastructure.

Bibliography

[1] Hong, J., Tu, R., Zhang, P., Zhang, R., Han, J., Fan, L., Wang, S., and Lu, X. (2023). Gnss rapid precise point positioning enhanced by low earth orbit satellites. *Satellite Navigation*, 4.

[2] Prol, F. S., Ferre, R. M., Saleem, Z., Välisuo, P., Pinell, C., Lohan, E. S., Elsanhoury, M., Elmusrati, M., Islam, S., Çelikbilek, K., Selvan, K., Yliaho, J., Rutledge, K., Ojala, A., Ferranti, L., Praks, J., Bhuiyan, M. Z. H., Kaasalainen, S., and Kuusniemi, H. (2022). Position, navigation, and timing (pnt) through low earth orbit (leo) satellites: A survey on current status, challenges, and opportunities. *IEEE Access*, 10:83971–84002.