Satellite Orbit Simulator

MAE – Matlab i les seves aplicacions en enginyeria

David Cerezo Trashorras

Índice

1. Introduction

This project aims to simulate satellite constellation scenarios—such as LEO or GEO constellations—using MATLAB, with a focus on orbit visualization, ground coverage analysis, and communication link budget estimation. Users will interact with a GUI-driven tool to simulate real-world constellations like GPS, GALILEO, or GLONASS, or input individual satellites using Two-Line Element (TLE) data. The system will allow custom placement of ground stations and analyze visibility, data coverage, and communication performance between satellites and ground-based receivers.

2. Theoretical Background

This project integrates several theoretical and computational domains to model, simulate, and analyze satellite constellations and their communication links with ground stations. The core topics include orbital mechanics, satellite communication theory, and signal propagation, all implemented using MATLAB's numerical and visualization capabilities.

2.1. Orbital mechanics and Satellite Propagation

The motion of satellites is governed by Newton's laws of motion and the law of universal gravitation. For Earth-orbiting satellites, Keplerian orbital elements provide a useful framework to describe their trajectories. These elements can be extracted from Two-Line Element (TLE) sets, which are standardized formats used to convey orbital parameters for satellites.

Two-Line Element Sets (TLEs) are a standard data format issued by organizations such as NORAD and Celestrak that contain the necessary orbital parameters for tracking satellites. Each set includes information like epoch time, inclination, and mean motion, and is updated frequently to account for orbital perturbations.

- **TLE Propagation:** The project uses TLEs with propagation models like SGP4 (Simplified General Perturbations Model 4) to determine satellite positions and velocities over time. MATLAB's Aerospace Toolbox provides built-in functions for parsing TLEs and propagating orbits accurately. In this project we use the satellite scenario viewer included in the Aerospace toolbox which is based on the SGP4 model.

- **Orbit Visualization:** Once propagated, the satellite trajectories can be projected in both Earth-Centered Inertial (ECI) and Earth-Centered Earth-Fixed (ECEF) coordinate systems, enabling realistic 2D and 3D visualizations of orbits and ground tracks.

2.2. Ground Station Geometry and Coverage

A key part of satellite communications is the visibility of the satellite from a given ground location.

- **Elevation and Visibility:** Using the satellite's propagated position and the location of the ground station, elevation angles are computed. A satellite is considered "in view" if it is above the horizon (elevation > 0°), though practical links often require higher elevation thresholds (e.g., 10° or 20°) to ensure reliable communication.

- **Azimuth, Elevation, and Range (AER)**: The position of a satellite relative to a ground station is described using three angles: azimuth (horizontal direction), elevation (angle above the horizon, explained in the prior bullet point), and slant range (distance). These are time-dependent and must be calculated dynamically using satellite ephemerides and ground station coordinates.

- **Footprint and Coverage:** The satellite's coverage footprint on Earth is modeled as a cone defined by the satellite's altitude and antenna beamwidth. This helps determine which ground locations are within the satellite's communication range at any given time.

2.3. Link Budget Analysis

A link budget accounts for all the gains and losses in a communication system to determine the signal quality at the receiver. This is central to evaluating the feasibility and performance of satellite-ground links.
The basic link budget formula is:

$$SNR\ (dB) = Ptx + Gtx + Grx - FSPL - Ls$$

Where:

- Ptx : Transmit power (dBW)
- Gtx : Transmit antenna gain (dBi)
- Grx: Receive antenna gain (dBi)
- FSPL : Free-space path loss (dB)
- Ls : System losses (dB)

$$FSPL = 20log10(d) + 20log10(f) + 92.45$$

Where d is the distance (km), and f is the frequency (GHz).

Link Margin

In the current project we will work with the Margin definition which is the difference between the received power and the required minimum SNR to stablish communication. In this scenario the following assumptions have been made:

- Atmospheric and rain losses are not explicitly modeled (hardcoded to 2 dB).
- Noise power and data rate are not used to derive SNR; required SNR is a fixed input.
- Doppler, BER, modulation effects are not included.

$$Margin = Prx - SNR\ required$$

This value indicates the feasibility of the communication link. A positive margin means that the link is expected to operate reliably under the given parameters, while a negative margin suggests insufficient signal strength. In practical systems, a link margin of 3–10 dB is typically maintained to accommodate fading, interference, and other impairments.

3. Implementation

The implementation of this satellite constellation simulation and analysis tool is structured around modular, maintainable MATLAB functions, with an intention to distribute computational load outside of the main GUI. The application is driven through a graphical user interface (GUI) built in App Designer, allowing users to load TLE data, select constellations, visualize orbits, compute access times, and evaluate link budgets.

The GUI serves as the main entry point for the application. To launch the simulation, the user simply adds the project folder to the MATLAB path and runs the main application class. The GUI components coordinate a set of helper functions that handle orbit propagation, visibility computation, access plotting, and link budget analysis.

Aside from the main functions the application also needs a folder of TLE files where we will store the TLE pairs of the pre-defined constellations like GPS, GLONASS or GALILEO.

3.1. Key modules:

3.1.1. Orbit propagator:

The orbitPropagator function is the computational core of the satellite constellation simulator. It encapsulates the full process of scenario construction, satellite instantiation, coverage analysis, and result visualization. It leverages the **Aerospace Toolbox** from MATLAB, which includes TLE parsing, SGP4 orbit propagation, access computations, and 3D rendering.

Inputs:

- **StartTime**: datetime – the beginning of the simulation.
- **SimulationTime**: double – duration of the simulation in seconds.
- **SampleTime**: double – time step between state evaluations.
- **tleFilename**: char – path to a TLE file containing one or more satellites.
- **gslat**: double – ground station latitude in degrees.
- **gslon**: double – ground station longitude in degrees.
- **axesHandle** *(optional)*: for plotting access data inside a GUI.

Key Functionality:

- **Scenario Setup**
  A satelliteScenario object is created with the specified temporal parameters. This object serves as the master environment where all entities (satellites, sensors, ground stations) are placed and propagated.

- **TLE Parsing and Satellite creation**
  The function reads a .tle file using MATLAB's satellite constructor, which supports multi-satellite input from TLE format. It automatically applies the **SGP4 model** for orbit propagation based on the orbital elements.

- **Sensor Attachment**

  A conicalSensor is instantiated and attached to each satellite. This sensor defines the beamwidth or footprint of the satellite and is later used for access analysis and visualization.

- **Ground Station Definition**

  A ground station is placed at the specified latitude and longitude. A minimum elevation mask (e.g., 30°) ensures realistic access constraints and avoids low-elevation noise in communication analysis.

- **Access Computation**

  The function computes **access intervals** (i.e., times when a satellite has line-of-sight to the ground station) by creating access objects between all satellite sensors and the ground station. These intervals can be queried for timeline analysis or coverage visualization.

Orbit and Field of View Visualization

- 3D satellite trajectories and ground tracks are shown using the *satelliteScenarioViewer.*
- Optionally, the first satellite's *fieldOfView* cone is visualized.
- A system-wide access timeline is plotted, indicating whether **any** satellite in the constellation is in view at a given moment.
- If an axes handle is provided (e.g., in a GUI), the access timeline is plotted directly in the user interface.

Coverage Metrics

The percentage of time the constellation provides access to the ground station is computed. This metric is essential for quantifying system effectiveness and supporting design trade-offs.

Outputs:

While the function does not return data directly, it populates GUI fields, draws plots, and stores objects (e.g., satellites and scenarios) via *setappdata* for downstream use in link budget calculations.

Dependencies:

- Aerospace Toolbox (TLE parsing, SGP4 propagation, satellite/GS modeling)
- MATLAB GUI (optional, if plotting inside App Designer)
- A .tle file containing at least one valid 3-line entry (name + 2 lines)

Here is the matlab code with more detailed comments:

```matlab
function [sat, sc] = orbitPropagator(StartTime, SimulationTime, SampleTime, 
tleFilename, gslat, gslon, axesHandle)
% orbitPropagator propagates satellite orbits and evaluates access from a 
ground station.
```

```matlab
%
% INPUTS:
%   StartTime       - datetime object for simulation start time
%   SimulationTime  - duration in hours (scalar)
%   SampleTime      - sample time in seconds (scalar)
%   tleFilename     - string, filename of the TLE file
%   gslat, gslon    - ground station latitude and longitude (in degrees)

    %% Create Satellite Scenario
    stopTime = StartTime + hours(SimulationTime);
    sc = satelliteScenario(StartTime, stopTime, SampleTime);

    %% Load Satellite Constellation from TLE
    if ~isfile(tleFilename)
        error("TLE file not found: %s", tleFilename);
    end

    sat = satellite(sc, tleFilename);
    numSats = numel(sat);
    fprintf("Loaded %d satellites from %s\n", numSats, tleFilename);
    %% Ground track plotting
    groundTrack(sat, "LeadTime", 3600, "TrailTime", 1800);  % 1 hour forward, 
30 min back

    %% Add Conical Sensors to Each Satellite
    coneAngle = 20;  % degrees (adjustable)
    sensorNames = sat.Name + " Sensor";
    cam = conicalSensor(sat, "Name", sensorNames, "MaxViewAngle", coneAngle);

    %% Add Ground Station
    gs = groundStation(sc, ...
        "Name", "User Ground Station", ...
        "Latitude", gslat, ...
        "Longitude", gslon, ...
        "MinElevationAngle", 30);

    %% Antenna pattern visualization
    antElem = phased.CosineAntennaElement( ...
    'FrequencyRange', [1e9 2e9], ...      % 1–2 GHz
    'CosinePower', 2);

    freq = 1.5754e9; %L1 frequency for GPS use
    beam = conicalSensor(sat, ...
    "Name", sat.Name + " Antenna Beam", ...
    "MaxViewAngle", 20);  % Adjust to match -3dB beamwidth from pattern


    %% Set Up Access Analysis
    ac = access(cam, gs);

    %% Make satellites track the ground site
    pointAt(sat, gs);

    %% Open Viewer
    v = satelliteScenarioViewer(sc, "ShowDetails", false);
    show(sat(1));  % Focus on first satellite
    sat(1).ShowLabel = true;
    gs.ShowLabel = true;
    %% Visualize FOV for first satellite
    fieldOfView(cam(1));

    %% Compute and Plot System-Wide Access
    for idx = 1:numel(ac)
        [s, timeVec] = accessStatus(ac(idx));
        if idx == 1
            systemWideAccessStatus = s;
        else
            systemWideAccessStatus = or(systemWideAccessStatus, s);
        end
    end

    % Plot access timeline
    plot(axesHandle, timeVec, systemWideAccessStatus, 'LineWidth', 2);
    grid(axesHandle, 'on');
    xlabel(axesHandle, "Time");
    ylabel(axesHandle, "System-Wide Access Status");
    title(axesHandle, "System-Wide Access Timeline");

    % Compute access percentage
    n = nnz(systemWideAccessStatus);
    systemWideAccessDuration = n * sc.SampleTime; % seconds
    scenarioDuration = seconds(sc.StopTime - sc.StartTime);
    systemWideAccessPercentage = (systemWideAccessDuration /
scenarioDuration) * 100;

    fprintf("System-Wide Access Time: %.2f minutes (%.2f%% of total)\n", ...
        systemWideAccessDuration / 60, systemWideAccessPercentage);

    %% Play Simulation
    play(sc);
end
```

3.1.2. Link Budget functions

The link budget analysis estimates the performance and reliability of a communication link between a satellite and a ground station. In this project, the link budget is calculated for every time step of the simulation, enabling the visualization of dynamic signal quality metrics such as SNR margin over time.

This calculation is implemented in a dedicated function called computeLinkBudget which performs a simple static calculation of the link budget in a determined moment. This first function served as the basis to develop the computeSNRTimeSeries function which performed the calculations dynamically allowing us to plot the results and have a much clearer understanding of the behavior of the SNR. It relies on physical modeling of free-space path loss and basic RF link components.

The matlab code is as follows:

computeLinkBudget:

```matlab
function result = computeLinkBudget(sat, sc, lat, lon, ...
    freqGHz, txPower, txGain, rxGain, reqSNR)

    losses = 2;
    % Use scenario time (start or current)
    time = sc.StartTime;

    % Create temporary ground station
    gs = groundStation(sc, "Latitude", lat, "Longitude", lon);

    % Get range to first satellite
    [~, ~, range] = aer(gs, sat(1), time);  % meters
    rangeKm = range / 1000;

    % FSPL calculation
    fspl = 20*log10(rangeKm) + 20*log10(freqGHz) + 92.45;

    % Received power
    prx = txPower + txGain + rxGain - fspl - losses;

    % Link margin
    margin = prx - reqSNR;

    % Bundle results
    result = struct();
    result.RangeKm = rangeKm;
    result.FSPL = fspl;
    result.ReceivedPower = prx;
    result.Margin = margin;
end
```

computeSNRTimeSeries

```
function [timeVec, marginVec, azVec, elVec, rangeVec] = computeSNRTimeSeries(
...
    sat, sc, lat, lon, freqGHz, txPower, txGain, rxGain, reqSNR)

    % Constants
    losses = 2;  % dB
    timeVec = sc.StartTime:seconds(sc.SampleTime):sc.StopTime;
    N = numel(timeVec);

    % Output arrays
    marginVec = zeros(1, N);
    azVec     = zeros(1, N);
    elVec     = zeros(1, N);
    rangeVec  = zeros(1, N);

    % Ground station
    gs = groundStation(sc, "Latitude", lat, "Longitude", lon);

    % Loop over time
    for k = 1:N
        [az, el, range] = aer(gs, sat(1), timeVec(k));  % range in meters
        rangeKm = range / 1000;

        % Store azimuth, elevation, distance
        azVec(k)    = az;
        elVec(k)    = el;
        rangeVec(k) = rangeKm;

        % FSPL and received power
        fspl = 20*log10(rangeKm) + 20*log10(freqGHz) + 92.45;
        prx = txPower + txGain + rxGain - fspl - losses;

        % Margin
        marginVec(k) = prx - reqSNR;
    end
end
```

3.1.3. User interface GUI

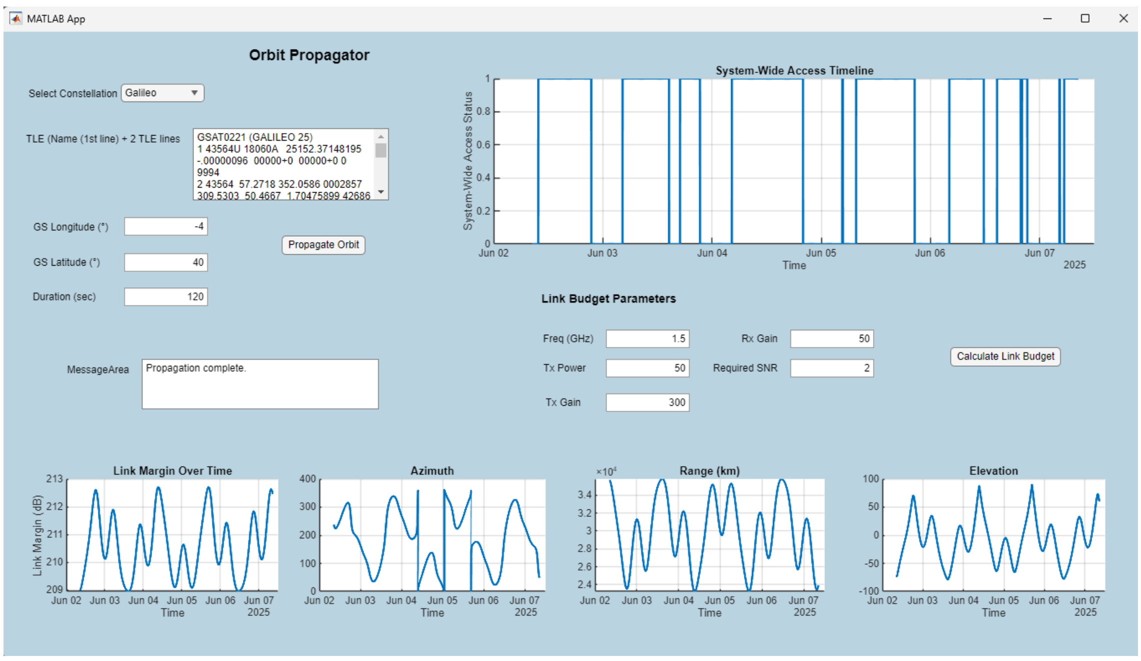
To improve accessibility and usability, the entire simulator is wrapped in a graphical user interface (GUI) developed using MATLAB App Designer. This interface enables users to simulate satellite constellations, visualize orbital dynamics, and analyze communication performance without needing to interact directly with the code.

3.2. Layout

The GUI consists of the orbit propagator fields on the left side where the user can input the desired TLE or select a pre-defined constellation from the dropdown menu. On the right side we have the fields regarding the calculation of SNR and its respective buttons to start the simulation/computation. The app also has 5 different axis displays where we can see the azimuth, range, elevation, Link margin over time and System-Wide Access Timeline. This last one allows us to see when a satellite is available to our network.

Error Handling

Most of the error handling (unless critical forcing Matlab to shutdown) is done through the message area included in the delivered GUI. There we will be able to see the results of some calculations, confirmation that our proceedings are going well or warnings about errors caused by our code.


MATLAB App

Orbit Propagator

Select Constellation: Galileo

TLE (Name (1st line) + 2 TLE lines:
```
GSAT0221 (GALILEO 25)
1 43564U 18060A   25152.37148195
-.00000096  00000+0  00000+0 0
9994
2 43564  57.2718 352.0586 0002857
309.5303  50.4667  1.70475899 42686
```

GS Longitude (°): -4

GS Latitude (°): 40

Duration (sec): 120

MessageArea: Propagation complete.

Link Budget Parameters

Freq (GHz): 1.5

Tx Power: 50

Tx Gain: 300

Rx Gain: 50

Required SNR: 2

The matlab code for the GUI is as follows:

```
classdef SatelliteApp < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                      matlab.ui.Figure
        RxGainField                   matlab.ui.control.NumericEditField
        RxGainEditFieldLabel       matlab.ui.control.Label
        RequiredSNRField           matlab.ui.control.NumericEditField
        RequiredSNREditFieldLabel  matlab.ui.control.Label
        TxGainField                matlab.ui.control.NumericEditField
        TxGainEditFieldLabel       matlab.ui.control.Label
        FreqGHzField               matlab.ui.control.NumericEditField
        FreqGHzEditFieldLabel      matlab.ui.control.Label
        TxPowerField               matlab.ui.control.NumericEditField
        TxPowerEditFieldLabel      matlab.ui.control.Label
        LinkBudgetParametersLabel  matlab.ui.control.Label
        OrbitPropagatorLabel       matlab.ui.control.Label
        MessageArea                matlab.ui.control.TextArea
        MessageAreaLabel           matlab.ui.control.Label
        ConstellationDropDown      matlab.ui.control.DropDown
        SelectConstellationDropDownLabel  matlab.ui.control.Label
        TLETextArea                matlab.ui.control.TextArea
        TLEName1stline2TLElinesTextAreaLabel  matlab.ui.control.Label
        GSLonField                 matlab.ui.control.NumericEditField
        GSLongitudeEditFieldLabel  matlab.ui.control.Label
        GSLatField                 matlab.ui.control.NumericEditField
        GSLatitudeLabel            matlab.ui.control.Label
        DurationField              matlab.ui.control.NumericEditField
        DurationsecEditFieldLabel  matlab.ui.control.Label
        CalculateLinkBudgetButton  matlab.ui.control.Button
        PropagateButton            matlab.ui.control.Button
        ElevationPlot              matlab.ui.control.UIAxes
        AzimuthPlot                matlab.ui.control.UIAxes
        RangePlot                  matlab.ui.control.UIAxes
        SNRPlot                    matlab.ui.control.UIAxes
        AccessPlot                 matlab.ui.control.UIAxes
    end

    % Callbacks that handle component events
    methods (Access = private)

        % Button pushed function: PropagateButton
        function PropagateButtonPushed(app, event)
             try
                % Retrieve GUI input values
                simTime = app.DurationField.Value;
                gslat = app.GSLatField.Value;
                gslon = app.GSLonField.Value;

                % Extract satellite TLE lines from input area
                tleLines = app.TLETextArea.Value;
                if length(tleLines) < 2
                    error('TLE must include at least two lines per
satellite.');
                end

                % Write TLE data to a temporary file
                tlePath = fullfile(tempdir, 'temp_tle.txt');
                fid = fopen(tlePath, 'w');
                for i = 1:length(tleLines)
                    fprintf(fid, '%s\n', tleLines{i});
                end
                fclose(fid);

                % Simulation parameters
                sampleTime = 60; % seconds
                startTime = datetime('now');

                % Clear old viewer window (if any)
                delete(findall(0, 'Type', 'Figure', 'Name', 'Satellite
Scenario Viewer'));

                % Call the orbit propagation function
                axes(app.AccessPlot);  % Make GUI axes the current target
                [sat, sc] = orbitPropagator(startTime, simTime, sampleTime,
tlePath, gslat, gslon, app.AccessPlot);
                axes(app.AccessPlot);

                % Stored values for auxiliar functions like Link Budget
                setappdata(0, 'TLE_StartTime', startTime);
                setappdata(0, 'TLE_SimTime', simTime);
                setappdata(0, 'TLE_SatelliteObj', sat);
                setappdata(0, 'TLE_Scenario', sc);


                app.MessageArea.Value = {'Propagation complete.'};
            catch ME
                app.MessageArea.Value = {['Error: ' ME.message]};
            end
        end

        % Button pushed function: CalculateLinkBudgetButton
        function CalculateLinkBudgetButtonPushed(app, event)
            % Get inputs from GUI
            lat     = app.GSLatField.Value;
            lon     = app.GSLonField.Value;
            freqGHz = app.FreqGHzField.Value;
            txPower = app.TxPowerField.Value;
            txGain  = app.TxGainField.Value;
            rxGain  = app.RxGainField.Value;
            reqSNR  = app.RequiredSNRField.Value;

            % Load satellite/scenario
            sat = getappdata(0, 'TLE_SatelliteObj');
            sc  = getappdata(0, 'TLE_Scenario');

            % Compute SNR time series
            [timeVec, marginVec, azVec, elVec, rangeVec] =
computeSNRTimeSeries(sat, sc, lat, lon, freqGHz, txPower, txGain, rxGain,
reqSNR);

            % Plot on dedicated SNR axes
            plot(app.SNRPlot, timeVec, marginVec, 'LineWidth', 2);
            xlabel(app.SNRPlot, 'Time');
            ylabel(app.SNRPlot, 'Link Margin (dB)');
            title(app.SNRPlot, 'Link Margin Over Time');
            grid(app.SNRPlot, 'on');

            plot(app.AzimuthPlot, timeVec, azVec, 'LineWidth', 2);
            plot(app.ElevationPlot, timeVec, elVec, 'LineWidth', 2);
            plot(app.RangePlot, timeVec, rangeVec, 'LineWidth', 2);
            %%

            % try
            %     % Retrieve stored scenario and satellite
            %     sat = getappdata(0, 'TLE_SatelliteObj');
            %     sc = getappdata(0, 'TLE_Scenario');
            %     if isempty(sat) || isempty(sc)
            %         error('Please propagate the orbit first.');
            %     end
            %
            %     % Get user input from GUI
            %     lat     = app.GSLatField.Value;
            %     lon     = app.GSLonField.Value;
            %     freqGHz = app.FreqGHzField.Value;
            %     txPower = app.TxPowerField.Value;
            %     txGain  = app.TxGainField.Value;
            %     rxGain  = app.RxGainField.Value;
            %     reqSNR  = app.RequiredSNRField.Value;
            %
            %     % Call external function
            %     result = computeLinkBudget(sat, sc, lat, lon, freqGHz, ...
            %                                 txPower, txGain, rxGain,
reqSNR);
            %
            %     % Show results in GUI
            %     app.MessageArea.Value = {
            %         char(sprintf("Range: %.1f km", result.RangeKm))
            %         char(sprintf("FSPL: %.2f dB", result.FSPL))
            %         char(sprintf("Received Power: %.2f dBW",
result.ReceivedPower))
            %         char(sprintf("Link Margin: %.2f dB", result.Margin))
            %     };
            %
            %
            % catch ME
            %     app.MessageArea.Value = {['Link budget error: '
ME.message]};
            % end
        end

        % Value changed function: ConstellationDropDown
        function ConstellationDropDownValueChanged(app, event)

            selection = lower(app.ConstellationDropDown.Value);

            % Folder where your TLE files are stored
            % Folder where your TLE files are stored
            thisFolder = fileparts(mfilename('fullpath'));
            tleFolder = fullfile(thisFolder, 'tle_files');
            tleFile = fullfile(tleFolder, [selection '.tle']);

            disp([selection '.tle'])

            if isfile(tleFile)
                try
                    tleText = readlines(tleFile);  % read entire TLE file
                    app.TLETextArea.Value = tleText;
                    app.MessageArea.Value = {['Loaded TLE for: '
upper(selection)]};
                catch ME
                    app.MessageArea.Value = {['Error reading TLE: '
ME.message]};
```

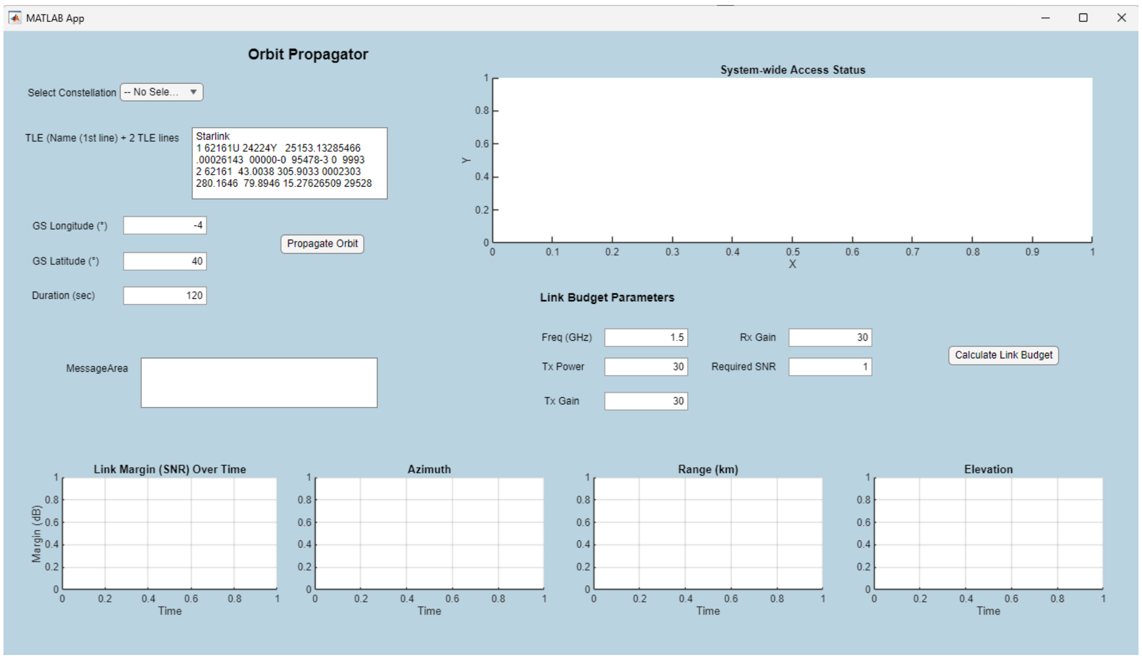
```
                end
            else
                app.TLETextArea.Value = {''};
                app.MessageArea.Value = {['TLE file not found for: '
upper(selection)]};
            end


        end

        % Callback function
        function LoadTLEButtonPushed(app, event)

        end
    end

    % Component initialization
    methods (Access = private)

        % Create UIFigure and components
        function createComponents(app)

            % Create UIFigure and hide until all components are created
            app.UIFigure = uifigure('Visible', 'off');
            app.UIFigure.Color = [0.7294 0.8275 0.8784];
            app.UIFigure.Position = [100 100 1361 746];
            app.UIFigure.Name = 'MATLAB App';

            % Create AccessPlot
            app.AccessPlot = uiaxes(app.UIFigure);
            title(app.AccessPlot, 'System-wide Access Status')
            xlabel(app.AccessPlot, 'X')
            ylabel(app.AccessPlot, 'Y')
            zlabel(app.AccessPlot, 'Z')
            app.AccessPlot.Position = [549 461 760 248];

            % Create SNRPlot
            app.SNRPlot = uiaxes(app.UIFigure);
            title(app.SNRPlot, 'Link Margin (SNR) Over Time')
            xlabel(app.SNRPlot, 'Time')
            ylabel(app.SNRPlot, 'Margin (dB)')
            zlabel(app.SNRPlot, 'Z')
            app.SNRPlot.XGrid = 'on';
            app.SNRPlot.YGrid = 'on';
            app.SNRPlot.Position = [35 47 300 185];

            % Create RangePlot
            app.RangePlot = uiaxes(app.UIFigure);
            title(app.RangePlot, 'Range (km)')
            xlabel(app.RangePlot, 'Time')
            zlabel(app.RangePlot, 'Z')
            app.RangePlot.XGrid = 'on';
            app.RangePlot.YGrid = 'on';
            app.RangePlot.Position = [687 47 300 185];

            % Create AzimuthPlot
            app.AzimuthPlot = uiaxes(app.UIFigure);
            title(app.AzimuthPlot, 'Azimuth')
            xlabel(app.AzimuthPlot, 'Time')
            zlabel(app.AzimuthPlot, 'Z')
            app.AzimuthPlot.XGrid = 'on';
            app.AzimuthPlot.YGrid = 'on';
            app.AzimuthPlot.Position = [354 47 300 185];

            % Create ElevationPlot
            app.ElevationPlot = uiaxes(app.UIFigure);
            title(app.ElevationPlot, 'Elevation')
            xlabel(app.ElevationPlot, 'Time')
            zlabel(app.ElevationPlot, 'Z')
            app.ElevationPlot.XGrid = 'on';
            app.ElevationPlot.YGrid = 'on';
            app.ElevationPlot.Position = [1022 47 300 185];

            % Create PropagateButton
            app.PropagateButton = uibutton(app.UIFigure, 'push');
            app.PropagateButton.ButtonPushedFcn = createCallbackFcn(app,
@PropagateButtonPushed, true);
            app.PropagateButton.Position = [334 481 100 23];
            app.PropagateButton.Text = 'Propagate Orbit';

            % Create CalculateLinkBudgetButton
            app.CalculateLinkBudgetButton = uibutton(app.UIFigure, 'push');
            app.CalculateLinkBudgetButton.ButtonPushedFcn =
createCallbackFcn(app, @CalculateLinkBudgetButtonPushed, true);
            app.CalculateLinkBudgetButton.Position = [1132 348 132 23];
            app.CalculateLinkBudgetButton.Text = 'Calculate Link Budget';

            % Create DurationsecEditFieldLabel
            app.DurationsecEditFieldLabel = uilabel(app.UIFigure);
            app.DurationsecEditFieldLabel.HorizontalAlignment = 'right';
            app.DurationsecEditFieldLabel.Position = [32 420 80 22];
            app.DurationsecEditFieldLabel.Text = 'Duration (sec)';

            % Create DurationField
            app.DurationField = uieditfield(app.UIFigure, 'numeric');
            app.DurationField.Position = [146 420 100 22];

            % Create GSLatitudeLabel
            app.GSLatitudeLabel = uilabel(app.UIFigure);
            app.GSLatitudeLabel.HorizontalAlignment = 'right';
            app.GSLatitudeLabel.Position = [32 461 85 22];
            app.GSLatitudeLabel.Text = 'GS Latitude (°)';

            % Create GSLatField
            app.GSLatField = uieditfield(app.UIFigure, 'numeric');
            app.GSLatField.Position = [146 461 100 22];

            % Create GSLongitudeEditFieldLabel
            app.GSLongitudeEditFieldLabel = uilabel(app.UIFigure);
            app.GSLongitudeEditFieldLabel.HorizontalAlignment = 'right';
            app.GSLongitudeEditFieldLabel.Position = [32 503 95 22];
            app.GSLongitudeEditFieldLabel.Text = 'GS Longitude (°)';

            % Create GSLonField
            app.GSLonField = uieditfield(app.UIFigure, 'numeric');
            app.GSLonField.Position = [146 504 100 22];

            % Create TLEName1stline2TLElinesTextAreaLabel
            app.TLEName1stline2TLElinesTextAreaLabel = uilabel(app.UIFigure);
            app.TLEName1stline2TLElinesTextAreaLabel.HorizontalAlignment =
'right';
            app.TLEName1stline2TLElinesTextAreaLabel.Position = [23 608 190
22];
            app.TLEName1stline2TLElinesTextAreaLabel.Text = 'TLE (Name (1st
line) + 2 TLE lines';

            % Create TLETextArea
            app.TLETextArea = uitextarea(app.UIFigure);
            app.TLETextArea.Position = [228 546 234 86];

            % Create SelectConstellationDropDownLabel
            app.SelectConstellationDropDownLabel = uilabel(app.UIFigure);
            app.SelectConstellationDropDownLabel.HorizontalAlignment =
'right';
            app.SelectConstellationDropDownLabel.Position = [27 662 111 22];
            app.SelectConstellationDropDownLabel.Text = 'Select
Constellation';

            % Create ConstellationDropDown
            app.ConstellationDropDown = uidropdown(app.UIFigure);
            app.ConstellationDropDown.Items = {'-- No Selection --',
'Galileo', 'GPS', 'Starlink', 'GLONASS'};
            app.ConstellationDropDown.ValueChangedFcn =
createCallbackFcn(app, @ConstellationDropDownValueChanged, true);
            app.ConstellationDropDown.Position = [142 663 100 22];
            app.ConstellationDropDown.Value = '-- No Selection --';

            % Create MessageAreaLabel
            app.MessageAreaLabel = uilabel(app.UIFigure);
            app.MessageAreaLabel.HorizontalAlignment = 'right';
            app.MessageAreaLabel.Position = [73 333 79 22];
            app.MessageAreaLabel.Text = 'MessageArea';

            % Create MessageArea
            app.MessageArea = uitextarea(app.UIFigure);
            app.MessageArea.Position = [167 297 283 60];

            % Create OrbitPropagatorLabel
            app.OrbitPropagatorLabel = uilabel(app.UIFigure);
            app.OrbitPropagatorLabel.FontSize = 18;
            app.OrbitPropagatorLabel.FontWeight = 'bold';
            app.OrbitPropagatorLabel.Position = [295 708 149 23];
            app.OrbitPropagatorLabel.Text = 'Orbit Propagator';

            % Create LinkBudgetParametersLabel
            app.LinkBudgetParametersLabel = uilabel(app.UIFigure);
            app.LinkBudgetParametersLabel.FontSize = 14;
            app.LinkBudgetParametersLabel.FontWeight = 'bold';
            app.LinkBudgetParametersLabel.Position = [644 418 166 22];
            app.LinkBudgetParametersLabel.Text = 'Link Budget Parameters';

            % Create TxPowerEditFieldLabel
            app.TxPowerEditFieldLabel = uilabel(app.UIFigure);
            app.TxPowerEditFieldLabel.HorizontalAlignment = 'right';
            app.TxPowerEditFieldLabel.Position = [641 335 56 22];
            app.TxPowerEditFieldLabel.Text = 'Tx Power';
            % Create TxPowerField
            app.TxPowerField = uieditfield(app.UIFigure, 'numeric');
            app.TxPowerField.Position = [721 335 100 22];

            % Create FreqGHzEditFieldLabel
            app.FreqGHzEditFieldLabel = uilabel(app.UIFigure);
            app.FreqGHzEditFieldLabel.HorizontalAlignment = 'right';
            app.FreqGHzEditFieldLabel.Position = [641 370 65 22];
            app.FreqGHzEditFieldLabel.Text = 'Freq (GHz)';

            % Create FreqGHzField
            app.FreqGHzField = uieditfield(app.UIFigure, 'numeric');
            app.FreqGHzField.Position = [721 370 100 22];

            % Create TxGainEditFieldLabel
            app.TxGainEditFieldLabel = uilabel(app.UIFigure);
            app.TxGainEditFieldLabel.HorizontalAlignment = 'right';
            app.TxGainEditFieldLabel.Position = [644 294 47 22];
            app.TxGainEditFieldLabel.Text = 'Tx Gain';

            % Create TxGainField
            app.TxGainField = uieditfield(app.UIFigure, 'numeric');
            app.TxGainField.Position = [721 294 100 22];

            % Create RequiredSNREditFieldLabel
            app.RequiredSNREditFieldLabel = uilabel(app.UIFigure);
            app.RequiredSNREditFieldLabel.HorizontalAlignment = 'right';
            app.RequiredSNREditFieldLabel.Position = [844 335 82 22];
            app.RequiredSNREditFieldLabel.Text = 'Required SNR';

            % Create RequiredSNRField
            app.RequiredSNRField = uieditfield(app.UIFigure, 'numeric');
            app.RequiredSNRField.Position = [941 335 100 22];

            % Create RxGainEditFieldLabel
            app.RxGainEditFieldLabel = uilabel(app.UIFigure);
            app.RxGainEditFieldLabel.HorizontalAlignment = 'right';
            app.RxGainEditFieldLabel.Position = [878 370 48 22];
            app.RxGainEditFieldLabel.Text = 'Rx Gain';

            % Create RxGainField
            app.RxGainField = uieditfield(app.UIFigure, 'numeric');
            app.RxGainField.Position = [941 370 100 22];

            % Show the figure after all components are created
            app.UIFigure.Visible = 'on';
        end
    end

    % App creation and deletion
    methods (Access = public)

        % Construct app
        function app = SatelliteApp

            % Create UIFigure and components
            createComponents(app)

            % Register the app with App Designer
            registerApp(app, app.UIFigure)

            if nargout == 0
                clear app
            end
        end

        % Code that executes before app deletion
        function delete(app)

            % Delete UIFigure when app is deleted
            delete(app.UIFigure)
        end
    end
end
```

4. Results:

In the video you can appreciate that if the constellation is big the loading times can get close to 30 second to a minute. It is also notable to mention that the blue line is the orbit, the yellow line is the ground track, and the purple is the antenna cone of visibility as seen from the satellite, this antenna is actively looking for the ground station. If the satellites enter the visibility zone from a ground station a green dashed line will appear indicating that communication would be possible between the market ground station and satellite.

The system wide access status gives us the information of whether a satellite can connect to our ground station.

The link to the youtube demo video is: https://youtu.be/cYV2-kcsaMk


MATLAB App

Orbit Propagator

Select Constellation: -- No Sele...

TLE (Name (1st line) + 2 TLE lines:

```
Starlink
1 62161U 24224Y   25153.13285466
.00026143  00000-0  95478-3 0  9993
2 62161  43.0038 305.9033 0002303
280.1646  79.8946 15.27626509 29528
```

GS Longitude (°): -4

GS Latitude (°): 40

Duration (sec): 120

MessageArea

Link Budget Parameters

Freq (GHz): 1.5

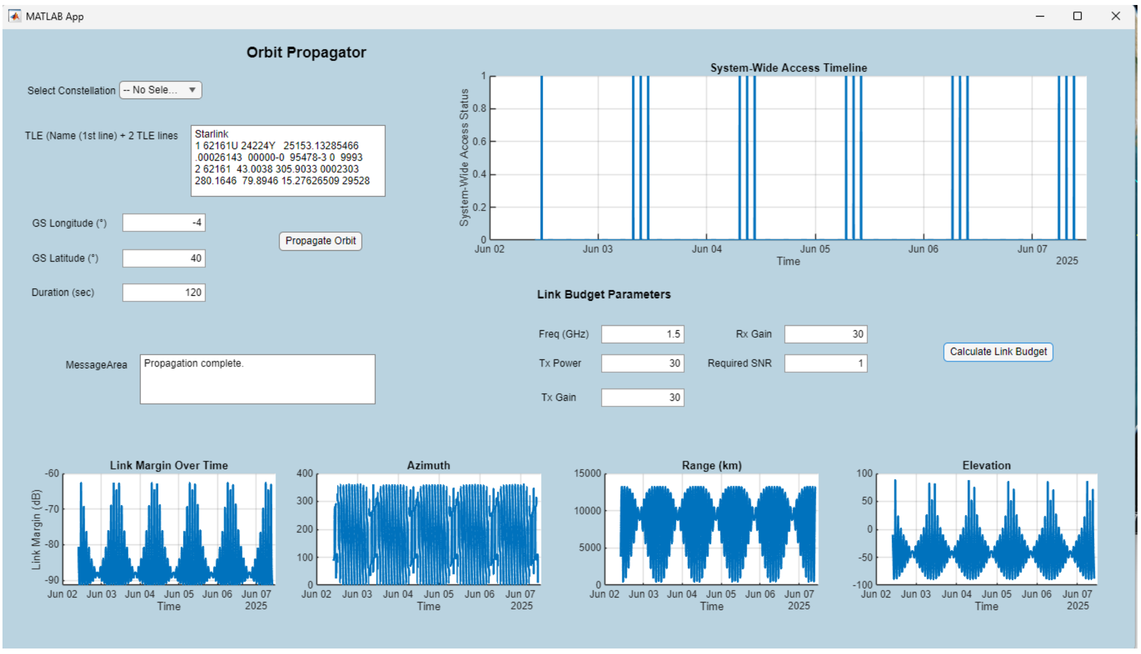
Tx Power: 30

Tx Gain: 30

Rx Gain: 30

Required SNR: 1

Figure 1: Before single satellite propagation. (1 Starlink satellite TLE is in the input box)



Select Constellation: -- No Sele...

TLE (Name (1st line) + 2 TLE lines:

```
Starlink
1 62161U 24224Y   25153.13285466
.00026143  00000-0  95478-3 0  9993
2 62161  43.0038 305.9033 0002303
280.1646  79.8946 15.27626509 29528
```

GS Longitude (°): -4

GS Latitude (°): 40

Duration (sec): 120

MessageArea: Propagation complete.

Freq (GHz): 1.5

Tx Power: 30

Tx Gain: 30

Rx Gain: 30

Required SNR: 1

Figure 2: After propagation and Link budget computation

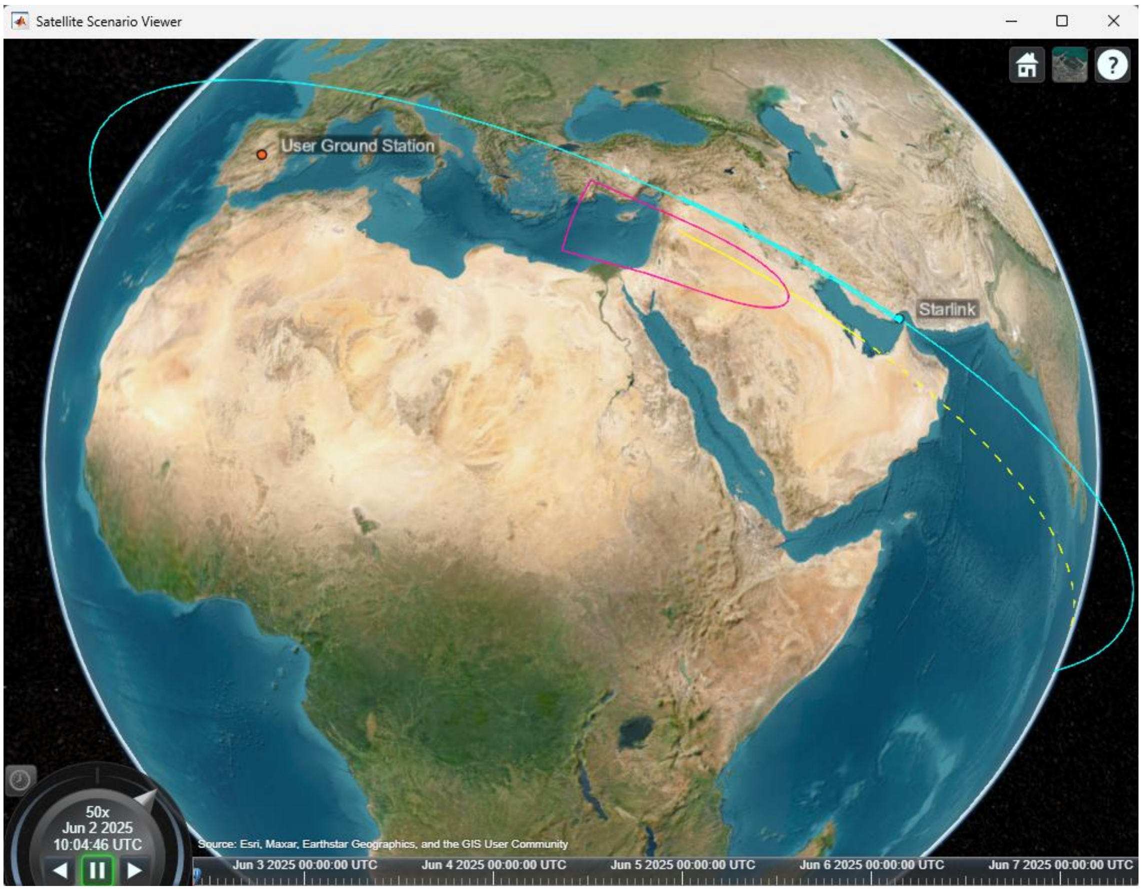


Figure 3: Viewer with Starlink satellite, user input Ground station, orbit, ground track and Conical sensor (purple) aiming towards the ground station.

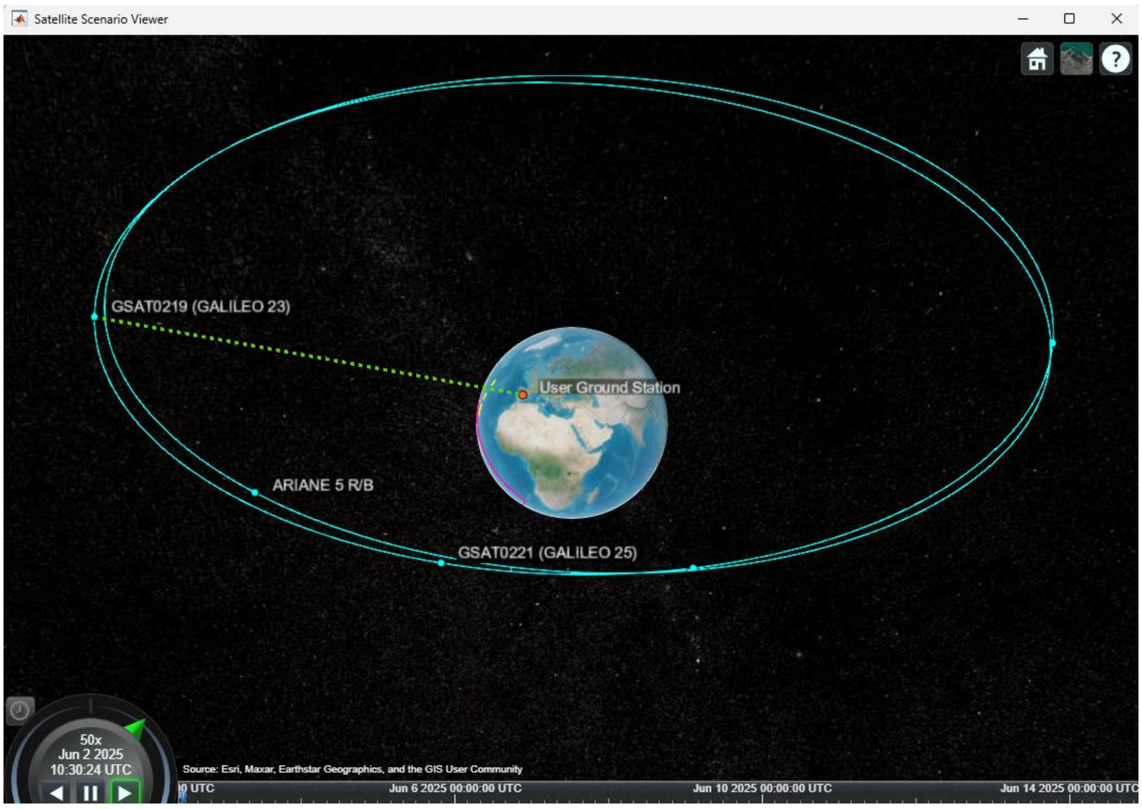


Figure 4: Part of the Galileo constellation with Arianne Spacecraft.

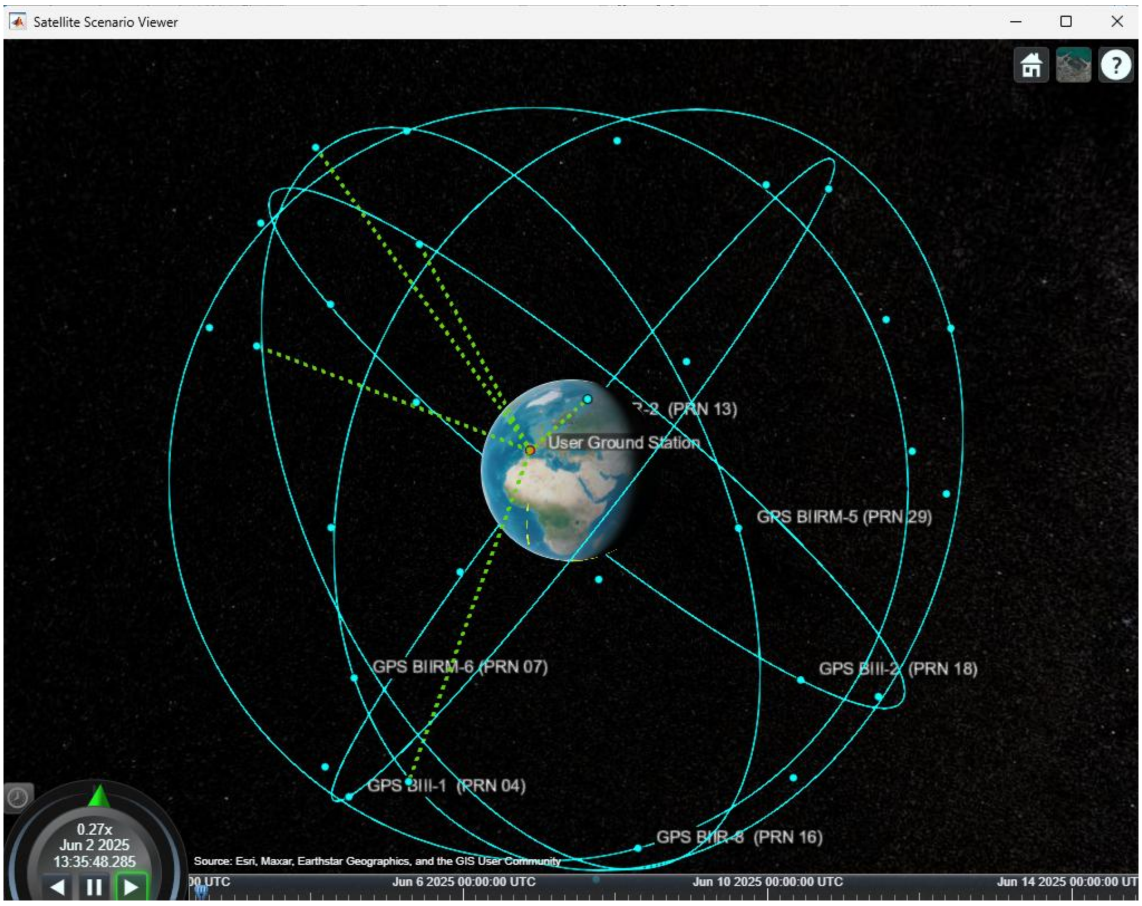


Figure 5: GPS satellite constellation.

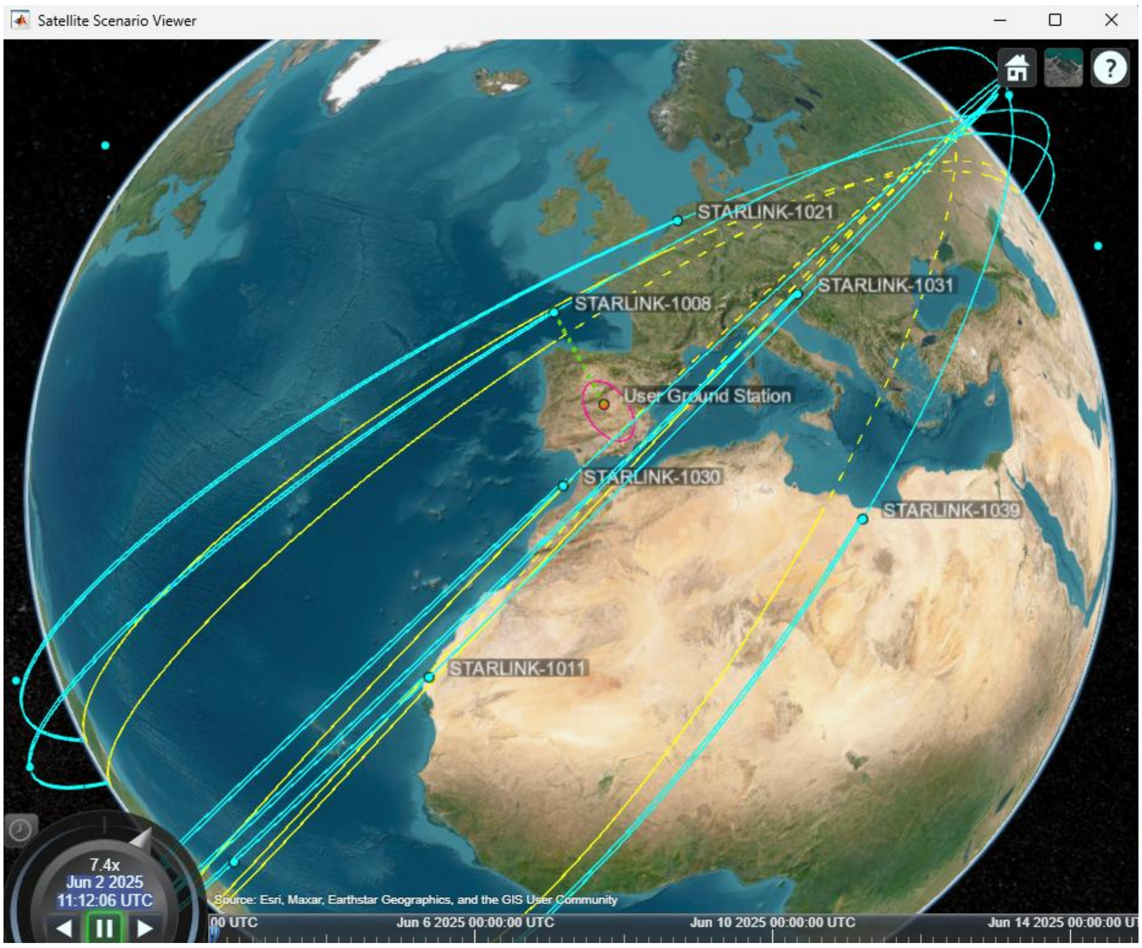
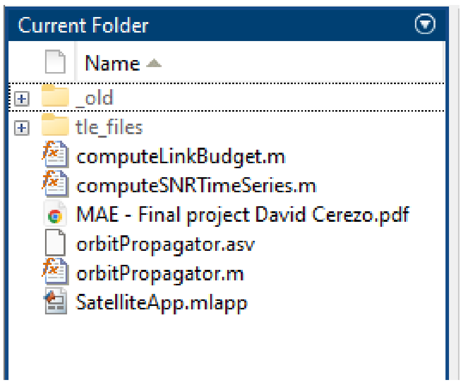


Figure 6: Part of Starlink constellation with active link to Madrid ground station.



```
Current Folder
Name
_old
tle_files
computeLinkBudget.m
computeSNRTimeSeries.m
MAE - Final project David Cerezo.pdf
orbitPropagator.asv
orbitPropagator.m
SatelliteApp.mlapp
```

This structure must be followed as the path is hardcoded into the MATLAB script. The .mlapp should be in the same folder as the tle_files folder which contains all the predefined constellations information.

5. Conclusion

This project successfully demonstrates the integration of orbital mechanics, satellite communications theory, and MATLAB's Aerospace Toolbox into a functional and interactive simulation environment. Using Two-Line Element (TLE) data, the simulator accurately propagates orbits of entire satellite constellations and visualizes their coverage and dynamics in real-time.

The inclusion of a link budget analysis further extends the tool's utility, allowing users to evaluate communication viability between satellites and ground stations. The graphical user interface (GUI) allows both technical users and non-specialists to engage with the simulator effectively.

Through modular code design and clear parameter inputs, this tool provides a scalable foundation for further extensions, such as real-time constellation tracking, multi-ground station coverage, or downlink data rate estimation. In the original draft more features were planned such as the antenna beam lobes displayed in the satellites, a doppler shift effect calculator but due to technical difficulties integrating it with the viewer it was not possible to achieve. It will stay as an upgrade for future versions of the simulator.

Overall, the simulator serves as a practical educational and engineering platform for exploring satellite-ground communication scenarios and the possibility of NTN (Non terrestrial networks.

6. References

The references used for this project are as follow:

[1] https://www.n2yo.com/satellite/?s=62161

[2] https://celestrak.org/NORAD/Elements/table.php?GROUP=starlink&FORMAT=csv

[3] https://www.n2yo.com/

[4] https://www.youtube.com/watch?v=9kDCaQ9_UrY&ab_channel=MATLAB

[5] https://es.mathworks.com/help/fusion/ug/detect-and-track-a-LEO-satellite-constellation-with-ground-radars.html

[6]https://es.mathworks.com/help/releases/R2024b/satcom/ref/satellitescenario.satellite.html#d126e17092

[7] https://es.mathworks.com/help/aerotbx/ug/tleread.html

[8] https://es.mathworks.com/help/releases/R2024b/satcom/ug/modeling-constellation-using-ephemeris-data.html

[9]https://es.mathworks.com/help/aerotbx/ug/matlabshared.satellitescenario.conicalsensor.html

[10] https://insidegnss.com/a-look-at-the-stars-navigation-with-multi-constellation-leo-satellite-signals-of-opportunity/

[11] https://es.mathworks.com/help/aerotbx/satellite-mission-analysis.html

[12]https://es.mathworks.com/help/aerotbx/ug/matlabshared.satellitescenario.conicalsensor.fieldofview.html