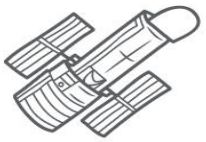
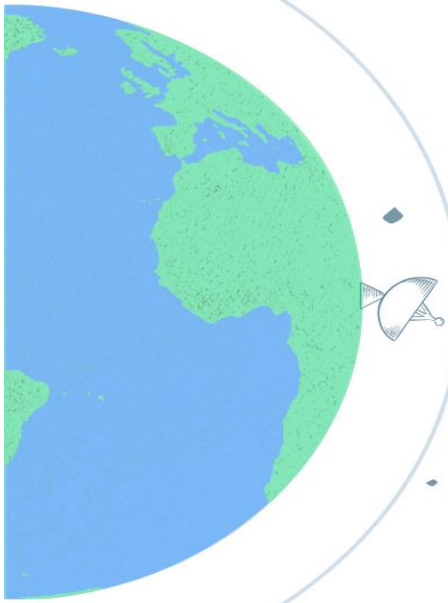
ODM

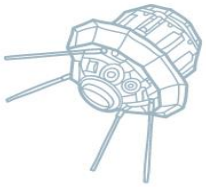
ORBIT DETERMINATION MANAGER





Prepared By:

Abdalla Ahamed Hannora
Kareem Hamada Abdelwahab
Rowida Elaraby Reda
Fatam El Zahraa Gabar
Nourhan Rashad Mohammed
Fatma Eid Mohammed



supervised by:

Dr/Fayza Ahmed Nada
Eng/Tamer Mohammed Anwar

2021-2022

ACKNOWLEDGEMENT

Apart from our team's efforts, the success of any project is mostly dependent on the support and guidance of many people. we'd like to take this opportunity to thank everyone who has contributed to the successful completion of this project.
We would like to express our gratitude to Dr. Fayza Ahamed Nada, Eng. Tamer Mohammed Anwar, and EGSA Egyptian space agency for helpful comments and sound counsel, as well as the entire faculty staff.
Finally, we'd like to express our joy at having arrived at this unique point in time.
Our current efforts are in vain. The actual accomplishment right now is our family's efforts. Thanks

Team members:

Abdalla Ahamed Elsayed	180900015	CS
Kareem Hamada Abdelwahab	180900017	CS
Rowida Elaraby Reda	180900005	CS
Fatma El Zahraa Gabar	180900019	CS
Nourhan Rashad Mohammed	170900005	IT
Fatma Eid Mohammed	180900022	CS

Abstract

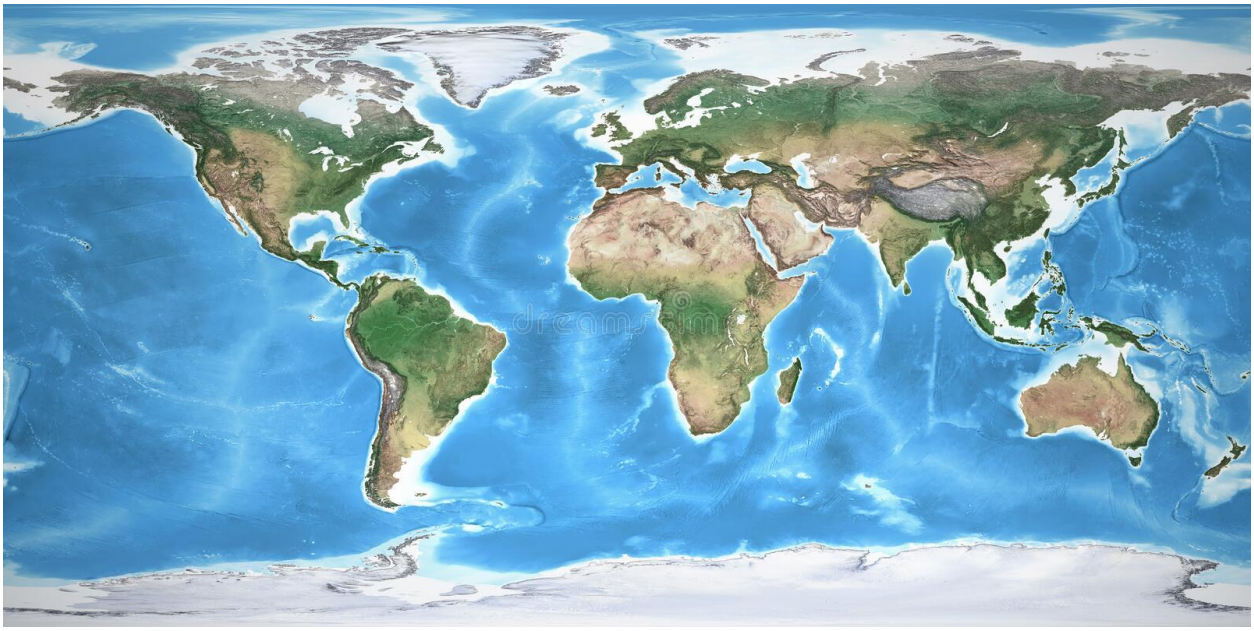
In the space environment, the structure is divided into two main segments: the space segment (satellite) and the ground segment (ground station). The ground station guarantees communication with the satellite through many subsystems responsible for controlling the whole satellite mission. The subsystem that is responsible for providing calculations in the ground station is the orbit subsystem of the Flight Control Center (FCC). The orbit subsystem is responsible for doing the calculations needed to know the position and velocity of the satellite in space. Based on the above data and according to the Egyptian 2030 vision, we developed a software called **Orbit Determination Manager (ODM).** This software package calculates the satellite trajectory in space "orbit propagation" relative to earth's International Terrestrial Reference Frame (ITRF). It also calculates satellite "Ground Track" which is the satellite trajectory projection on the earth taking into consideration the earth's rotation around itself, which is important in the missions that involve, for example earth imagery. It also calculates data for Radio visibility Zones (RVZ) providing elevation and azimuth of the satellite along with the time in which ground stations' antenna can connect and communicate with the satellite. All the three available data calculations are done based on the Two-Line Elements (TLE) data set that describes the satellite's six orbital elements. Because ODM is mainly a scientific program, it was important to do powerful visualization of the calculated data and handle multiple satellites in different conditions. Powerful features like auto TLE update, full user-defined scenario, and a user-friendly interface were also provided to make it more fixable and easier to use. Because the main purpose of ODM is to provide accurate data that will lead the antenna towards successful connection, it's important to mention that after several steps of development ODM managed to reach data calculation accuracy that is almost 99.98% accurate based on the real tests and compared to the actual standard data used in Egyptian Space Agency (EGSA) so the software is reliable and able to be used in ground stations.

Table of Contents:

Chapter 1 Introduction

We will study in this chapter:

- What prompted us to work on this project?
- Let's take a brief overview of the project
- Who is the target group of the project?
- How did we arrive and achieve the project?



1.1 Motivation

First, we trained at EGSA in the summer and successfully produced our first project with the hardware and this is a real challenge for us. We learned a lot from this experience and we are very proud of this.
So, we decided to continue with them for the graduation project which is also a challenge; Because they chose half of the participants in the summer training

Fortunately, we were chosen for our competence in the summer training to complement our graduation project under their supervision and in cooperation with the supervision of the college and to gain from the two experiences together.
It was the first time that the University of Suez was chosen for this training, and for the first time they completed the university moon project, which they set up every year, and this was another challenge.
There were a lot of ideas for our graduation project but we chose this one which aims to develop the Flight Control Center (FCC): an orbiting and propagation determination software package (ground station subsystem) that does satellite orbit predictions based on TLE tuned to track satellites Artificial in-orbit, wireless vision zones detection, ground tracking, illumination period detection, end junction for flight schedule management
Including its singularity and some experiences as well that are intended to be learned, but the idea has won our admiration, and we hope that you will also like it...

1.2 System overview

The main mission is to develop Flight Control Center (FCC): Orbit determination and propagation software package that has:
specific technical objectives:

1. Forecasting of satellite movement parameters
2. Calculating of the satellite ground tracks.
3. Build a simple Propagator for satellite Position.
4. Drowning the ground track line on 2D or 3D simulator for visual determination of the satellite motion
5. Calculate the ground station radio visibility zone (Time of Communication sessions).

6. Create full database for all the futures of the orbit parameters with all its classes and schema
7. Build easy to use familiar graphical user interface that helps Operator for easy to access all the program feature
8. Calculate the target designation for ground station antenna (where the antenna will be aiming).
9. Store the results in a local and shared database.

System Engineering Objectives:

The main objective is to make sure that all futures are clear and compatible and that the baseline design of the overall system has converged to satisfy the specifications. To fulfill these objectives, the following tasks were performed.

1. Establishment of a Functional Block Diagram of the overall system.
2. Elaboration and coordination of system level trade-offs.
3. Definition of operational modes (Tracking – Communication Session – Ground Trace).
4. Definition of the system level requirements.
5. Establishment of data flow through all the system features.
6. Documentation, preparation of end-of-phase review.

System Functional Description:

The Orbit subsystem is a system that is composed of a GUI and a Database. Each segment has its own functions to perform, hence each segment is composed of tasks, and each Task has its functions to perform. Next is the functional description for each Task with its segment's functions.

- Satellite orbital motion model.
- Mathematical models of physical objects.
- Numerical methods for tasks solution.

For satellite orbital motion model, the following assumptions are taken:

- Within the range of the task solved separately, the satellite is represented as a material point with a constant mass.
- Angular satellite orbital motion about center of mass is not considered.

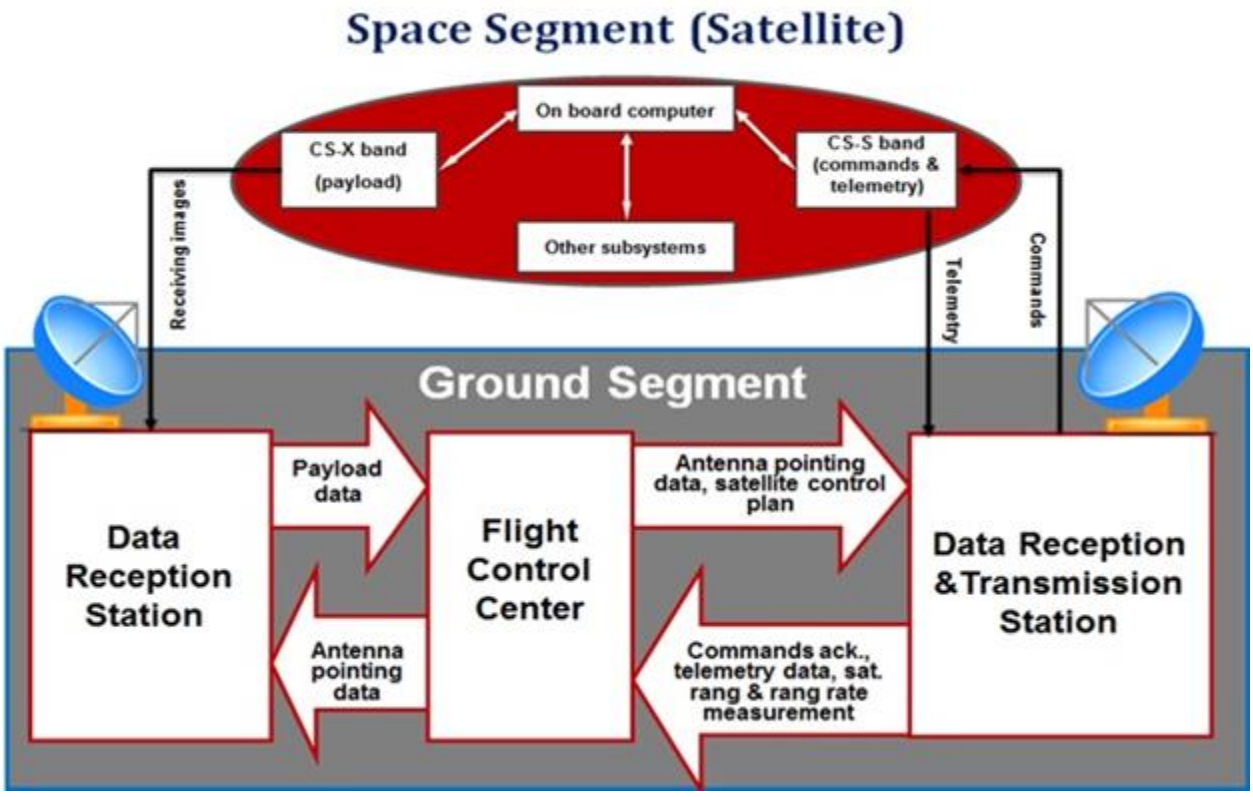
For description of satellite orbital motion, the system of ordinary differential equations is used in osculating elements with a possible account (or without account) of the following disturbing factors in the right sides:

- Earth gravitational field (EGF) non-homogeneity.
- Atmospheric effect.
- Sun and Moon gravitational effects.

for more information
https://www.linkedin.com/company/egyptian-space-agency/

1.3 Target Market:

The ground station (segment) is divided into some subsystems that work in cooperative manner to reach the target goal that is to trace the satellite and to establish the satellite mission and more ….
FCC is a part of the ground segment that is responsible for Controlling and monitoring mission plans and monitoring satellite subsystem status.
(Figure 1)



(Figure 1) ground segment & space segment illustration

Function's overview of project:

- Mathematical model of Processing Measurements.

  1- Reading of unprocessed Two-Line Elements (TLE) MCTP or GPS receiver MCTP (session vectors of states), required for FCC operation.

  2- Determination of true orbit number of GPS receiver measurements and session number in this orbit.

- Mathematical model of Satellite Motion Parameters.

  1- Determination of parameters, characterizing a satellite motion, is performed by measurements received from Two Line Elements

  (TLE) MCTP or from GPS receiver (session state vectors).

  2- Determination of the satellite motion parameters using (TLE) MCTP trajectory information.

  3-Each operation variant provides one execution mode - revise the satellite motion parameters X, Y, Z, Vx, Vy, Vz.

- Mathematical model of initial conditions of satellite motion.
  Calculation and recording of the satellite motion Initial condition (IC) to the database.

  1. For the date and time required.
  2. For the beginning of an orbit given.

- Calculation of Target Designations.

  The purpose of the program is a calculation of target designations for pointing GCS DRTF antenna devices.

  1. TD calculation for the orbit interval.
  2. TD calculation for the time interval.

- Calculation of radio-visibility zones:

  Calculation at a given time interval (interval of orbits) for each given ground point:

  - Orbit numbers in entering the 0-degree and ¡
    - Degree radio-visibility zones (RVZ) (¡ – given minimum elevation angle for the satellite RVZ).
  - Time, range, azimuths at entry/exit moments to/from the 0
    - Degree and ¡- degree RVZ, staying duration in the ¡
    - Degree RVZ.
  - Time, range, azimuth, and elevation angle at the satellite orbit point, remote at minimum distance from the ground point (parameter point).

- Database information input, viewing and updating:

  The purpose of the program is performance of service functions in the work with local PC DB data arrays and server DB tables the access to which is provided for the program.

  - Viewing and updating of the contents of data array records.
  - Delete separate specified records from a data array.
  - Addition of records to the specified data array.
  - Copying of a selected record.
  - printing out of the data array contents (the specified data array records)
  - viewing of the contents of table records, deleting of separate specified records from table

- Calculation of satellite ground-track:

Calculation at a given time interval (of orbits) with a given time step:
Satellite altitude and geodesic coordinates of an orbit projection trace.

- Sun elevation angle at sub- satellite point.

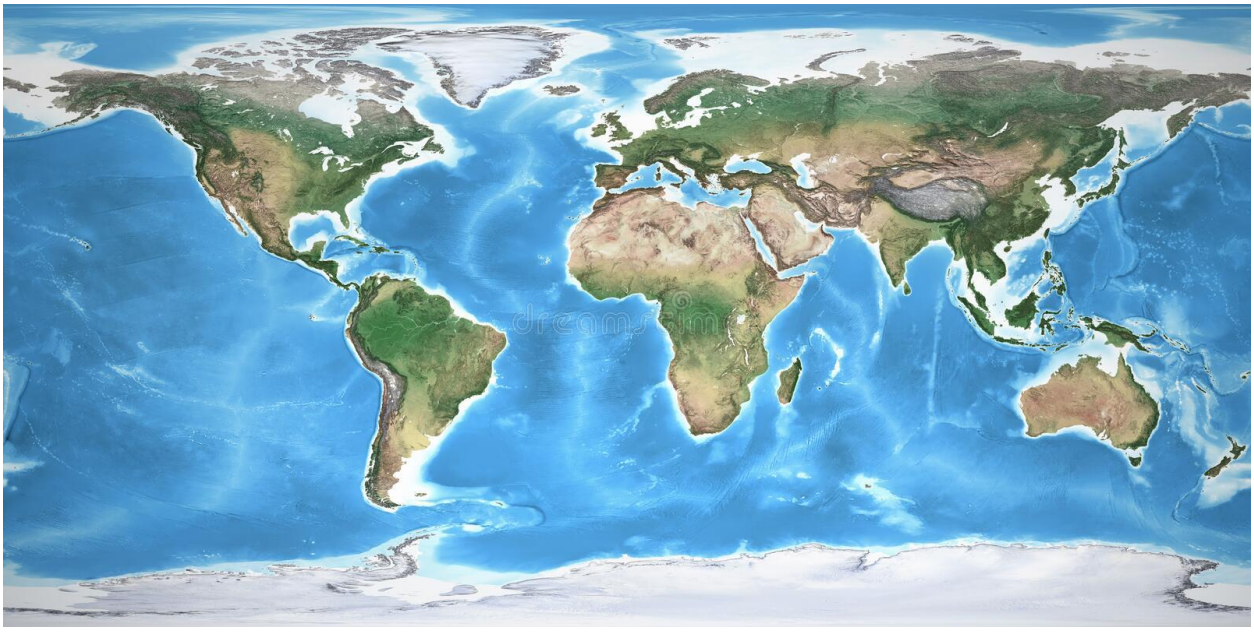
1.4 Cooperation Aspects

How did the agency help us?
Initially, in the summer training, many engineers allowed us to train and know the satellite in general with all its parts, installations and much more.
Which made us start the first track in the graduation project and how to choose the most appropriate project for us
And then they helped us in selecting us for the graduation project with a supervisor from the agency who was familiar with the project and highly qualified to help us
Let's get to know this agency
EGSA, Egyptian Space Agency
is an Egyptian public economic authority established in August 2019, with a legal personality and affiliated with the President of the Arab Republic of Egypt.
Established by Law No. 3 of 2018 which aims to create, transfer space technology development, localization, and own self-capabilities to build & launch satellites from Egyptian territory. In 2019, The Egyptian President Abdel Fattah Al-Sisi announced the appointment of Dr. Mohamed Eloksi as the CEO of the Egyptian Space Agency.

Chapter 2 Literature Review

We will study in this chapter

- All the important definitions to know what the project is about
- The main functions of the satellite missions



2.1 Definition

2.1.1 Orbit

An orbit is a regular, repeating path that one object in space takes around another one. An object in an orbit is called a satellite. A satellite can be natural, like Earth or the moon. Many planets have moons that orbit them. A satellite can also be man-made, like the International Space Station.
Planets, comets, asteroids and other objects in the solar system orbit the sun. Most of the objects orbiting the sun move along or close to an imaginary flat surface. This imaginary surface is called the ecliptic plane.
What Shape Is an Orbit?
Orbits come in different shapes. All orbits are elliptical, which means they are an ellipse, similar to an oval. For the planets, the orbits are almost circular. The orbits of comets have a different shape. They are highly eccentric or "squashed." They look more like thin ellipses than circles.[21]

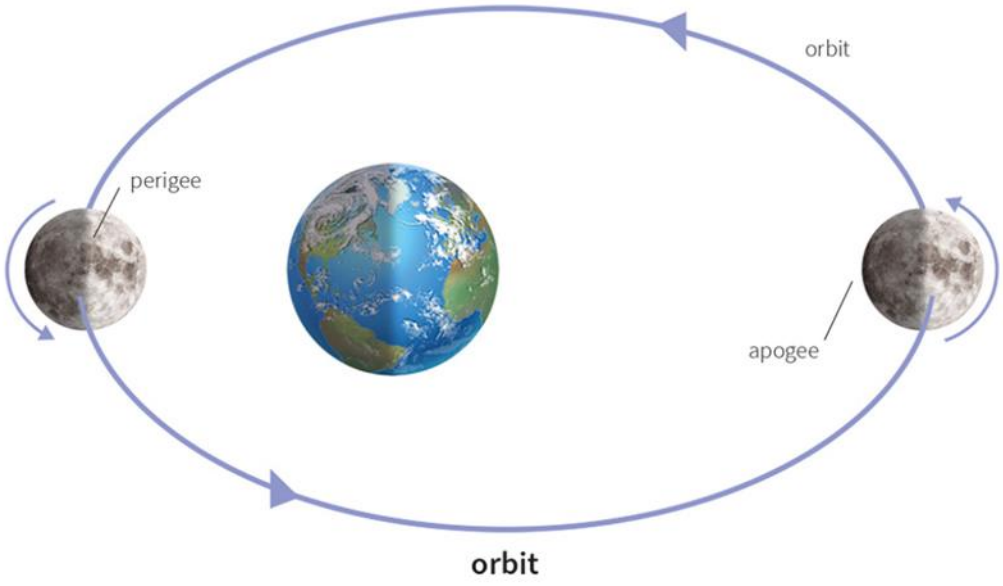


fig2.1 orbit

2.1.2 Payload

Satellite payload is defined as modules carried on satellites with the ability to perform certain functionalities. Satellite consists of the payload and the bus. Satellite bus is used to carry subsystems and payloads, whereas payload is developed to perform specific functions such as communications, imaging, and navigation while in orbit. Satellite payload includes equipment such as transponders, repeaters, antennas, spectrometers, and cameras among others. Earth Observation and remote sensing applications use cameras, radar, and other sensing equipment for collecting relevant data. To enable delivery of satellite communication services, payload includes antennas, receivers, and transmitters, based both in-space and on the ground, for transmission and storage of the data needed.[9]

2.1.3 Orbit Propagation

Propagator is a model whose objective is to determine the position of satellite at any instance of time, with given acceleration and initial velocity. If we assume the earth is spherical and only earth's gravitational field is affecting satellite motion then the problem would be pretty easy to solve but the issue arises when other factors like earth oblateness, gravitational force from moon and sun, atmospheric drag and solar pressure come into play.
There are some factors that affect the propagation more than others. For example, a satellite that is orbiting around the earth will be greatly affected by the earth's oblateness rather than gravitational field of sun and moon (as they are very far away). As the number of factors taken into account in the model increases, the complexity of the model increases.

2.1.4 Ground Track

A ground track or ground trace is the path on the surface of a planet directly below an aircraft's or satellite's trajectory. In the case of satellites, it is also known as a suborbital track, and is the vertical projection of the satellite's orbit onto the surface of the Earth (or whatever body the satellite is orbiting).

A satellite ground track may be thought of as a path along the Earth's surface that traces the movement of an imaginary line between the satellite and the center of the Earth. In other words, the ground track is the set of points at which the satellite will pass directly overhead, or cross the zenith, in the frame of reference of a ground observer.[10]

2.1.5 TLE

The TLE, or Two-Line Elements set, is a data format that contains information about the orbit at a specific epoch of an Earth-orbiting object. The information is split into two lines with 70 characters each. In the following, it is presented an example of a TLE describing the orbit of the Brazilian satellite SCD-1 at 25 December 2018:

SCD 1
1 22490U 93009B 18359.76217587 .00000186 00000-0 84512-6 0 9998
2 \22490 24.9694 116.1709 0043211 90.3968 62.0083 14.44539396366163

The TLE contains all the necessary information to propagate the orbit using, for example, the SGP4 orbit propagator. Hence, this package contains a set of functions that help to load the TLE information to be used in the Orbit propagators.

If the TLEs are stored in one file, then they can be loaded using the function:

function read_tle (tle_filename::String, verify_checksum::Bool = true).[5]

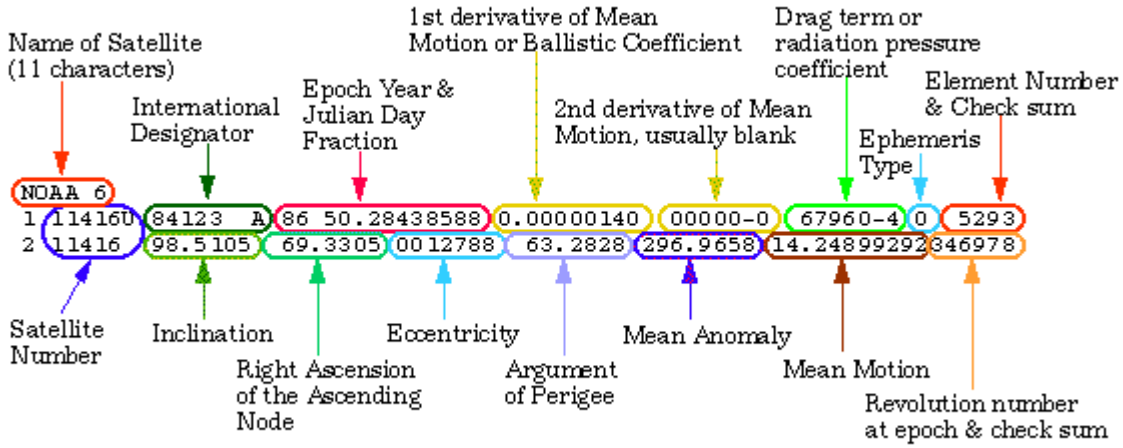


fig2.2 tle

2.1.6 Radio Visibility Zone

Basically, the satellite and the ground terminal must be in radio line of sight of each other, since the frequency is used in commercial and military satellite communications are inherently line of sight frequencies. Terrain, tall buildings, and other obstructions can prevent effective communication.
There are also inherent limits to satellite communication (at least to GEO stationary communications satellites) imposed by the shape of the earth. The global beam coverage map of the NSS-7 satellites shown below is illustrative of the limits of RLOS coverage of a geostationary communications satellite. This coverage map is from the SES World Skies website.
The thin black lines show the limits of coverage for iso-elevation angle at an earth station located at the points where the lines intersect the surface of the earth. My eyes aren't what they used to be, but I suspect the outermost circle (distorted to allow portrayal on a flat map) is at the 0 degree is elevation angle circle. At a zero-degree elevation angle the earth terminal would be pointed along a tangent to the surface of a spherical earth, and communication with the satellite would probably not be possible.
At a 90-degree elevation angle the earth terminal would be at the subsatellite point and would be pointed straight up.
As you can see, communication to the polar regions is limited by the curvature of the earth. As I hazily recall, this limit is about set 72 degrees north and south latitude, but I could be off by a couple of degrees.[15]

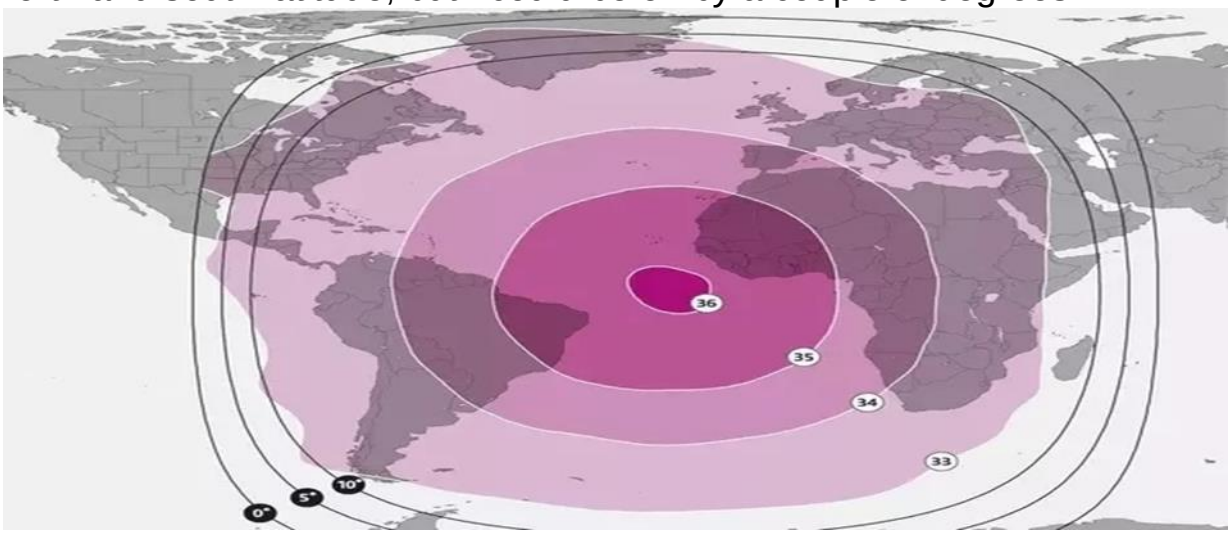


fig2.3 RVZ

2.1.7 Ground Station

A ground station, Earth station, or earth terminal is a terrestrial radio station designed for extraplanetary telecommunication with spacecraft (constituting part of the ground segment of the spacecraft system) or reception of radio waves from astronomical radio sources. Ground stations may be located either on the surface of the Earth, or in its atmosphere.[1] Earth stations communicate with spacecraft by transmitting and receiving radio waves in super high frequency (SHF) or extremely high frequency (EHF) bands (e.g., microwaves). When a ground station successfully transmits radio waves to a spacecraft (or vice versa), it establishes a telecommunications link. The principal telecommunications device of the ground station is the parabolic antenna.

2.4 ground track

2.2 FCC Main Functions(intro)

The ground station (segment) is divided into some subsystems that work in cooperative manner to reach the target goal that is to trace the satellite and to establish the satellite mission and more ….

FCC is a part of ground segment that is responsible for Control and monitoring mission plans and monitoring satellite subsystem status

2.2.1 FCC Main Functions:

1. Schedule satellite operations, payload tasks and ground data reception station;

2. Transmit commands to control the satellite subsystems and monitoring transmission of these commands
3. Determination of satellite orbit parameters
4. Calculate trajectory (navigational) data and data for pointing Data Reception and Transmission Facilities (DRTF)antenna;
5. Monitoring of operation of the satellite subsystem through processing of telemetry data received from the satellite;
6. Analysis statues of satellite subsystem (real time) in off-nominal situations.
7. synchronize onboard time and ground time;
8. Perform data exchange between the FCC subsystems and DRTF.
9. Archive and manage the data received or generated by FCC

2.2.2 main subsystems

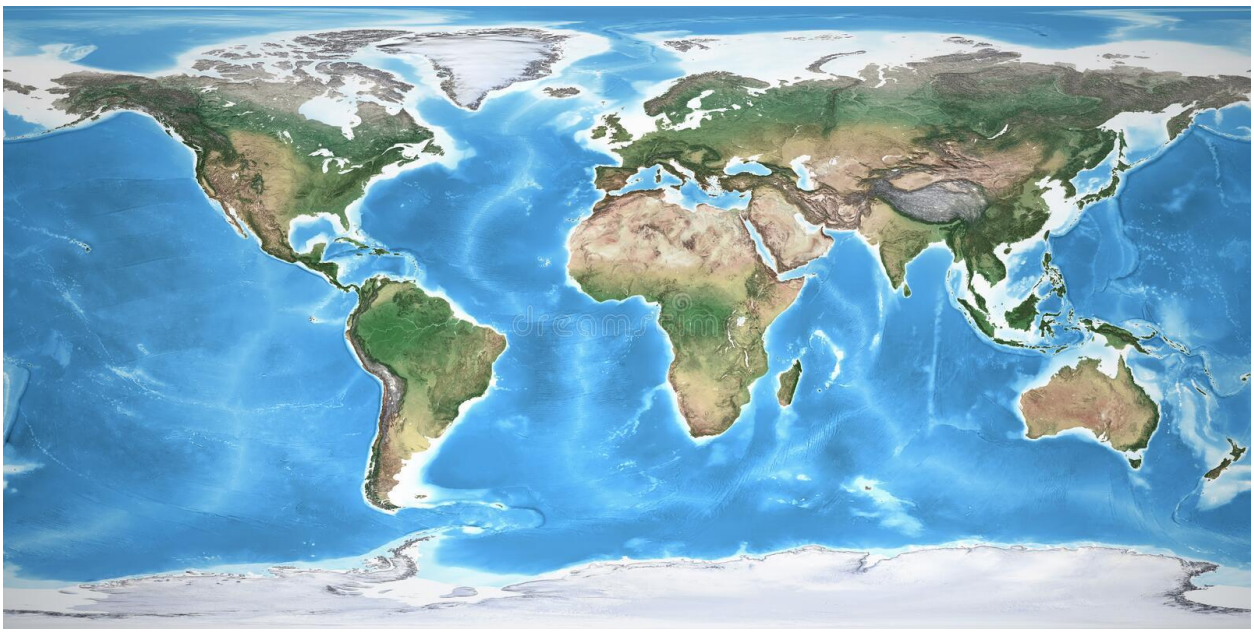
The FCC main subsystems:

- FCC server.
- Telemetry subsystem.
- Payload planning subsystem
- Orbit determination and propagation subsystem
- Satellite control subsystem

Chapter 3 Tools and Technologies

We will study in this chapter

- What languages are used in the project?
- The most important libraries
- The general interface of the project



3.1 Programing Language

3.1.1 Python

Python is a high-level, interpreted, general-purpose programming language. Its design philosophy emphasizes code readability with the use of significant indentation.
Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly procedural), object-oriented and functional programming. It is often described as a "batteries included" language due to its comprehensive standard library.

Advantages of Python:

Easy to Read, Learn and Write. Python is a high-level programming language that has English-like syntax.

- Improved Productivity. Python is a very productive language
- Interpreted Language.
- Dynamically Typed.
- Free and Open-Source.
- Vast Libraries Support.
- Portability.

3.1.1.1 Skyfield

is a pure-Python astronomy package that is compatible with both Python 2 and 3 and makes it easy to generate high precision research-grade positions for planets and Earth satellites.
Has NumPy as its only binary dependency, the fundamental package for scientific computing with Python, whose vector operations make Skyfield efficient.[1]

3.1.1.2 matplotlib

Matplotlib is a plotting library available for the Python programming language as a component of NumPy, a big data numerical handling resource. Matplotlib uses an object-oriented API to embed plots in Python applications.
https://matplotlib.org/

3.1.1.2.1 Pyplot

is an API (Application Programming Interface) for Python's matplotlib that effectively makes matplotlib a viable open-source alternative to MATLAB. Matplotlib is a library for data visualization, typically in the form of plots, graphs and charts.
https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.html
matplotlib.pyplot is a collection of functions that make matplotlib work like MATLAB. Each pyplot function makes some change to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc.
In matplotlib.pyplot various states are preserved across function calls, so that it keeps track of things like the current figure and plotting area, and the plotting functions are directed to the current axes (please note that "axes" here and in most places in the documentation refers to the axes part of a figure and not the strict mathematical term for more than one axis).

3.1.1.3 sgp4

Library for calculating satellite positions and predicting overpasses. Tested on an ESP8266 and ESP32.
The Simplified General Perturbations (SGP4) propagator is used with two-line mean element (TLE) sets. It considers secular and periodic variations due to Earth oblateness, solar and lunar gravitational effects, gravitational resonance effects, and orbital decay using a drag model. By default, STK utilizes the CSSI SGP4 routine, Version 2008-11-03.
The SGP4 propagator generates ephemeris in the True Equator Mean Equinox coordinate system based on the epoch of the specified TLE. The information contained in a TLE is a set of mean orbital elements that are specific to the SGP4 propagator.
The Air Force makes available TLEs for a subset of space objects being monitored (about 12,000 as of April 2009). These TLEs are then provided to customers using AGI online server.
A TLE contains an epoch (in YYDDD.ddddd format); ephemeris produced by SGP4 using this TLE is generally valid for a few days about the epoch [No official validity interval is provided by the Air Force]. New TLEs are produced over time for each space object because of this limitation. In using TLEs, then, it is important to use the correct set of TLEs for the time interval of interest.

You can input the initial conditions for the SGP4 propagator manually, but it is much more common to input TLE sets from a TLE file. TLE files must be in the TLE File format (*.tce/*.tle). STK provides several options for managing element sets, thus avoiding the need for manual input.
(Pip install sgp4).[2]

3.1.1.4 requests

Requests is an HTTP library for the Python programming language. The goal of the project is to make HTTP requests simpler and more human-friendly. The current version is 2.27.1. Requests are released under the Apache License 2.0. Requests are one of the most popular Python libraries that is not included with Python.

3.1.1.5 CSV

A comma-separated values (CSV) file is a delimited text file that uses a comma to separate values. Each line of the file is data record. Each record consists of one or more fields, separated by commas. The use of the comma as a field separator is the source of the name for this file format. A CSV file typically stores tabular data (numbers and text) in plain text, in which case each line will have the same number of fields.
The CSV file format is not fully standardized. Separating fields with commas is the foundation, but commas in the data or embedded line breaks have to be handled specially. Some implementations disallow such content while others surround the field with quotation marks, which yet again creates the need for escaping if quotation marks are present in the data.
The term "CSV" also denotes several closely-related delimiter-separated formats that use other field delimiters such as semicolons.[2] These include tab-separated values and space-separated values. A delimiter guaranteed not to be part of the data greatly simplifies parsing.
Alternative delimiter-separated files are often given a ".csv" extension despite the use of a non-comma field separator. This loose terminology can cause problems in data exchange. Many applications that accept CSV files have options to select the delimiter character and the quotation character. Semicolons are often used instead of commas in many European locales in order to use the comma as the decimal separator and, possibly, the period as a decimal grouping character.

3.1.1.6 Cartop

Cartopy is a Python package designed for geospatial data processing in order to produce maps and other geospatial data analyses. Cartop makes use of the powerful PROJ, NumPy and Shapely libraries and includes a programmatic interface built on top of Matplotlib for the creation of publication quality maps.
Key features of cartopy are its object oriented projection definitions, and its ability to transform points, lines, vectors, polygons and images between those projections.
You will find cartopy especially useful for large area / small scale data, where Cartesian assumptions of spherical data traditionally break down. If you've ever experienced a singularity at the pole or a cut-off at the dateline, it is likely you will appreciate cartop's unique features!

3.1.1.7 NumPy

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays

Adv

- Multi-dimensional Slicing
- Broadcasting Functionality
- Processing Speed
- Memory Footprint

Dis

- Library-Independent
- Intuitive

3.1.1.7 pandas

Pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license
https://pandas.pydata.org/

3.1.1.8 sys

System Library means anything that is normally distributed (in either source or binary form) with the major components (kernel, window system etc.) of the operating system(s) on which the Software or Modification runs, or a compiler used to produce the Object Code, or an object code interpreter used to run it.

3.1.1.9 DOCX

A DOCX file is a Microsoft Word document that typically contains text. DOCX is the newer version of DOC, the original official Microsoft Word file format. They are both opened using Microsoft Word, though alternate software programs open them as well. A DOCX is a convenient XML format, making it incredibly popular.

3.1.1.10 datetime

Python Datetime module supplies classes to work with date and time. These classes provide a number of functions to deal with dates, times and time intervals. Date and datetime are objects in Python, so when you manipulate them, you are actually manipulating objects and not string or timestamps.

3.1.1.11 OS

The OS module in Python provides functions for creating and removing a directory (folder), fetching its contents, changing and identifying the current directory, etc. You first need to import the os module to interact with the underlying operating system.

3.2 Front End

3.2.1 PYQT5

PyQt5 is a comprehensive set of Python bindings for Qt v5. It is implemented as more than 35 extension modules and enables Python to be used as an alternative application development language to C++ on all supported platforms including iOS and Android.
PyQt5 may also be embedded in C++ based applications to allow users of those applications to configure or enhance the functionality of those applications.
Qt is a set of cross-platform C++ libraries that implement high-level APIs for accessing many aspects of modern desktop and mobile systems. These include location and positioning services, multimedia, NFC and Bluetooth connectivity, a Chromium-based web browser, as well as traditional UI development.

3.3 Mathematics task

In this part of the project step 1 that is getting TLE data and process, which is done
So, the second step of the process is to get and convert the orbital elements to get the initial position and velocity at the exact epoch in which the TLE set was taken using some sort of calculations that are mentioned below
Using the position and velocity at this exact time of epoch will take us to the next step which is a prediction of the satellite pass of the next (X) orbits until the TLE set values are not valid anymore

- Obtain the position and velocity vector o(t) and o(t), respectively, in the orbital frame (z-axis perpendicular to orbital plane, x-axis pointing to periapsis of the orbit):

$$f(E) = E - e\sin E - M$$

$$E_{j+1} = E_j - \frac{f(E_j)}{\frac{\mathrm{d}}{\mathrm{d}E_j} f(E_j)} = E_j - \frac{E_j - e\sin E_j - M}{1 - e\cos E_j}, \quad E_0 = M$$

- Obtain the true anomaly *ν*(*t*):

$$\nu(t) = 2 \cdot \operatorname{arctan2}\left(\sqrt{1+e}\sin\frac{E(t)}{2}, \sqrt{1-e}\cos\frac{E(t)}{2}\right),$$

where arctan2 is the two-argument arctangent function:

$$\operatorname{arctan2}(y,x) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ \text{undefined} & y = 0, x = 0 \end{cases}.$$

- Use the eccentric anomaly E(t) to get the distance to the central body (r_c) :

$$r_c(t) = a(1 - e\cos E(t)).$$

Obtain the position and velocity vector o(t) and o(t), respectively, in the orbital frame (z-axis perpendicular to orbital plane, x-axis pointing to periapsis of the orbit):

$$\mathbf{o}(t) = \begin{pmatrix} o_x(t) \\ o_y(t) \\ o_z(t) \end{pmatrix} = r_c(t) \begin{pmatrix} \cos\nu(t) \\ \sin\nu(t) \\ 0 \end{pmatrix}, \quad \dot{\mathbf{o}}(t) = \begin{pmatrix} \dot{o}_x(t) \\ \dot{o}_y(t) \\ \dot{o}_z(t) \end{pmatrix} = \frac{\sqrt{\mu a}}{r_c(t)} \begin{pmatrix} -\sin E \\ \sqrt{1-e^2}\cos E \\ 0 \end{pmatrix}$$

- Transform o(t) and o(t), to the inertial frame in body-centric (in case of the Sun as central body: heliocentric) rectangular coordinates r(t) and r(t), with the rotation matrices $\underline{R_z}(\varphi)$ and $\underline{R_x}(\varphi)$ using the transformation sequence:

$$\mathbf{r}(t) = \underline{R}_z(-\Omega)\underline{R}_x(-i)\underline{R}_z(-\omega)\mathbf{o}(t)$$

$$\underline{\underline{o_z(t)=0}} \begin{pmatrix} o_x(t)(\cos\omega\cos\Omega - \sin\omega\cos i\sin\Omega) - o_y(t)(\sin\omega\cos\Omega + \cos\omega\cos i\sin\Omega) \\ o_x(t)(\cos\omega\sin\Omega + \sin\omega\cos i\cos\Omega) + o_y(t)(\cos\omega\cos i\cos\Omega - \sin\omega\sin\Omega) \\ o_x(t)(\sin\omega\sin i) + o_y(t)(\cos\omega\sin i) \end{pmatrix}$$

$$\dot{\mathbf{r}}(t) = \underline{R}_z(-\Omega)\underline{R}_x(-i)\underline{R}_z(-\omega)\dot{\mathbf{o}}(t)$$

$$\underline{\underline{\dot{o}_z(t)=0}} \begin{pmatrix} \dot{o}_x(t)(\cos\omega\cos\Omega - \sin\omega\cos i\sin\Omega) - \dot{o}_y(t)(\sin\omega\cos\Omega + \cos\omega\cos i\sin\Omega) \\ \dot{o}_x(t)(\cos\omega\sin\Omega + \sin\omega\cos i\cos\Omega) + \dot{o}_y(t)(\cos\omega\cos i\cos\Omega - \sin\omega\sin\Omega) \\ \dot{o}_x(t)(\sin\omega\sin i) + \dot{o}_y(t)(\cos\omega\sin i) \end{pmatrix}.$$

At this step, the position (r), and velocity (v) obtained mathematically in the **ECI** reference frame in an ideal case (other Orbital Perturbations) are note considered and this is the next and final step in the orbit prediction process.

The previous steps were written in code and represented in tle2rv class that have two internal methods prepare_orbital_elements(), orbital2rv () (fig.2)

```
class tle2rv:
    def prepare_orbital_elements(self, inclination, raan, argument_of_perigee, mean_anomaly, mean_motion):...

    def orbital2rv(self, inclination, raan, eccentricity, argument_of_perigee, mean_anomaly, mean_motion,
                   gravitational_parameter,
                   time):...
```

le2rv class implementation (fig.2)

The **first** (prepare_orbital_elements()): do the conversion of TLE set element's units to the standard used in mathematical calculations [Rad] for angels, [Rad/s] for the mean motion (fig.3)

```
def prepare_orbital_elements(self, inclination, raan, argument_of_perigee, mean_anomaly, mean_motion):
    deg2rad = pi / 180.0  # [degree] to [radian]  deg2rad = 0.01745329251994330
    revday2rads = (2 * pi) / 86400  # [revolution/day] to [radian/second]

    # change units of orbital elemnts to be used in calculations
    inclination = inclination * deg2rad
    raan = raan * deg2rad
    argument_of_perigee = argument_of_perigee * deg2rad
    mean_anomaly = mean_anomaly * deg2rad
    mean_motion = mean_motion * revday2rads
    return inclination, raan, argument_of_perigee, mean_anomaly, mean_motion
```

prepare_orbital_elements method (fig.3)

The **second** orbital2rv (): do the calculations listed above and return the position and velocity vectors (fig.4).[8]

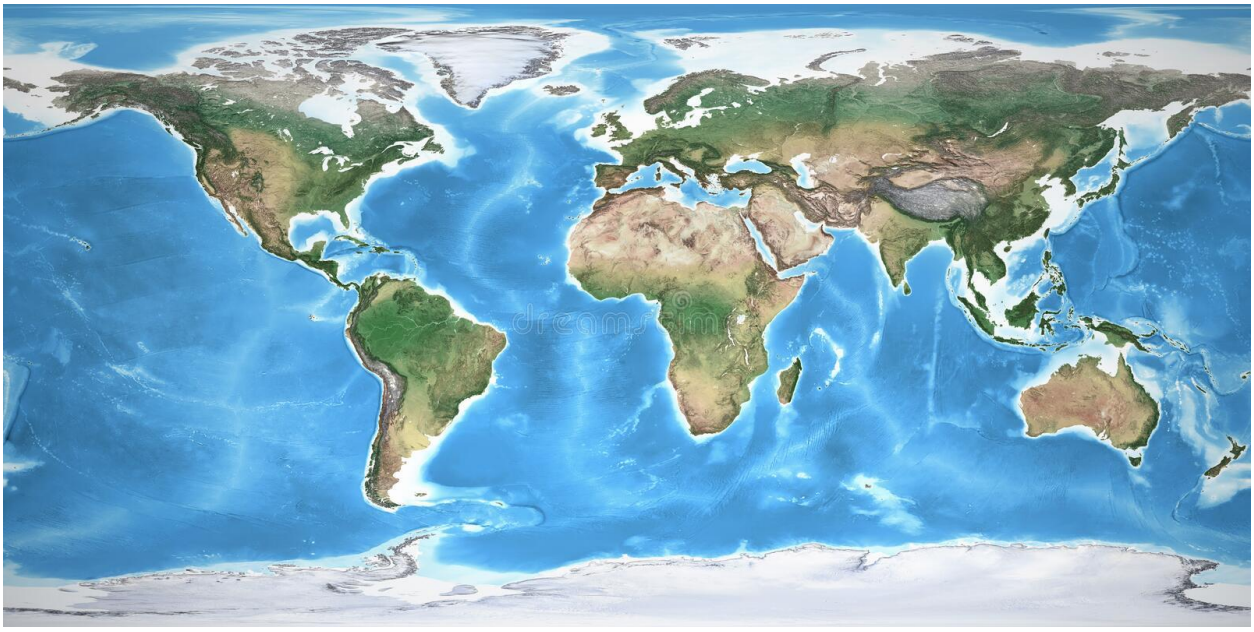
```
def orbital2rv(self, inclination, raan, eccentricity, argument_of_perigee, mean_anomaly, mean_motion,
               gravitational_parameter,
               time):
```

Orbital2rv method (fig.4)

Chapter 4 System analysis and Design

We will study in this chapter

- A simplified explanation of each diagram
- Diagrams of the project
- Design for the project
- The competent program for design and how to use it



4.1 Flow Chart

4.1.1 What is a Flowchart

A flowchart is a diagram that depicts a process, system or computer algorithm. They are widely used in multiple fields to document, study, plan, improve and communicate often complex processes in clear, easy-to-understand diagrams. Flowcharts, sometimes spelled as flow charts, use rectangles, ovals, diamonds and potentially numerous other shapes to define the type of step, along with connecting arrows to define flow and sequence. They can range from simple, hand-drawn charts to comprehensive computer-drawn diagrams depicting multiple steps and routes. If we consider all the various forms of flowcharts, they are one of the most common diagrams on the planet, used by both technical and non-technical people in numerous fields

Log in flowchart:

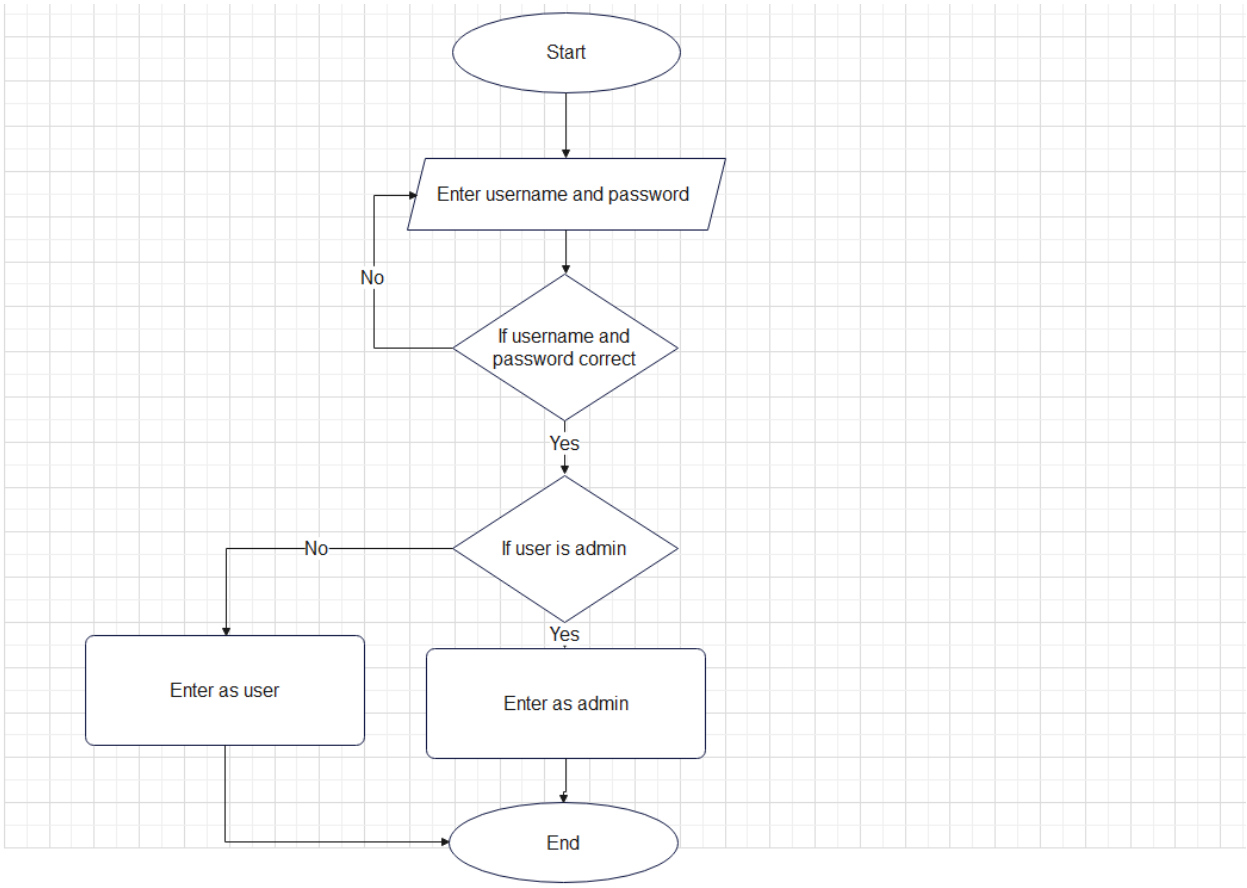


Fig 4.1 Log in flowchart

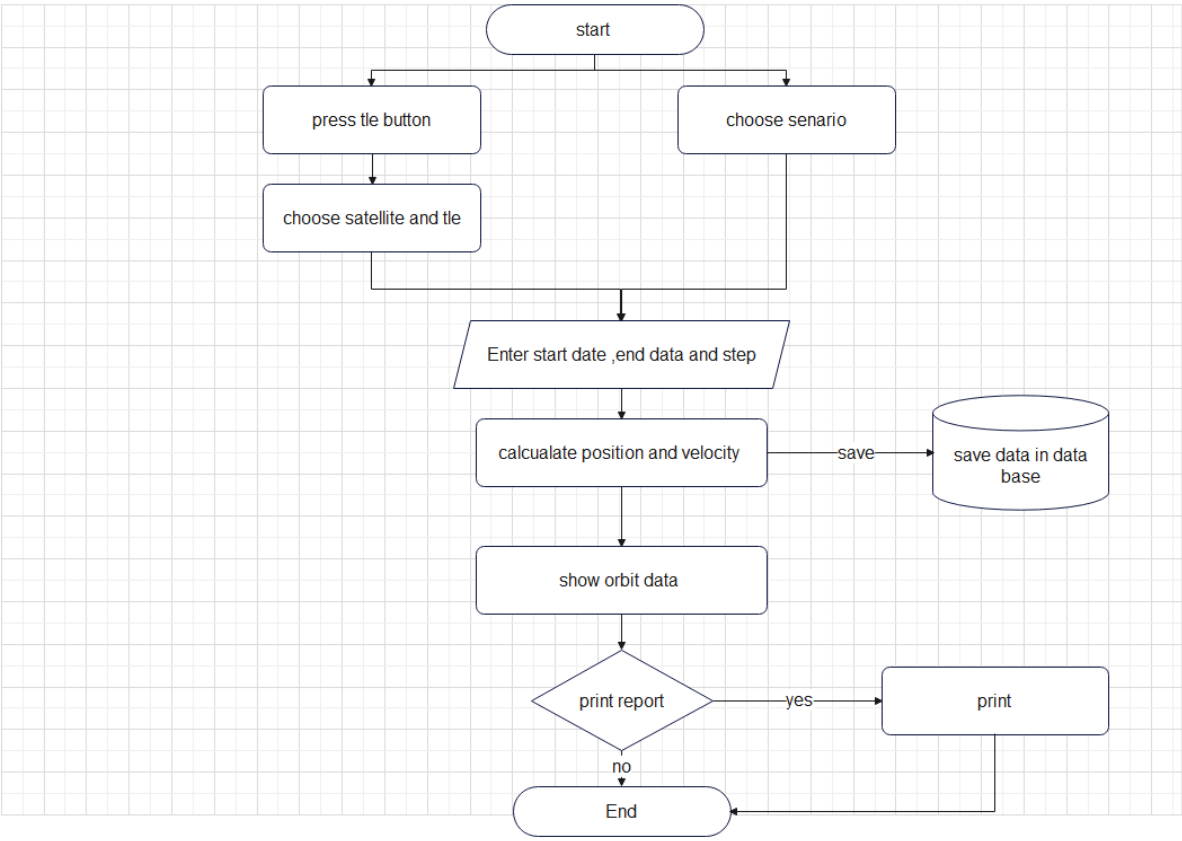


fig4.2Orbirt flowchart

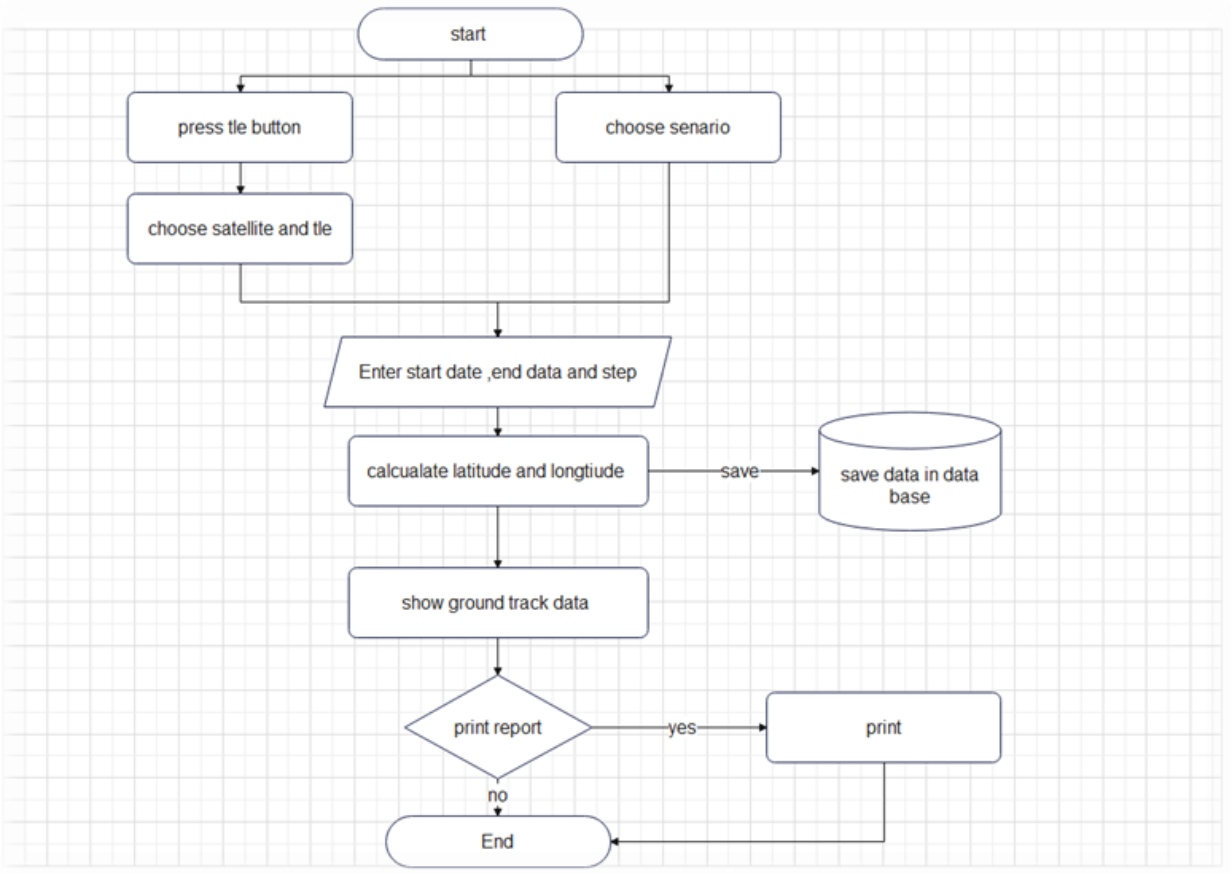


Fig4.3 Ground track flowchart

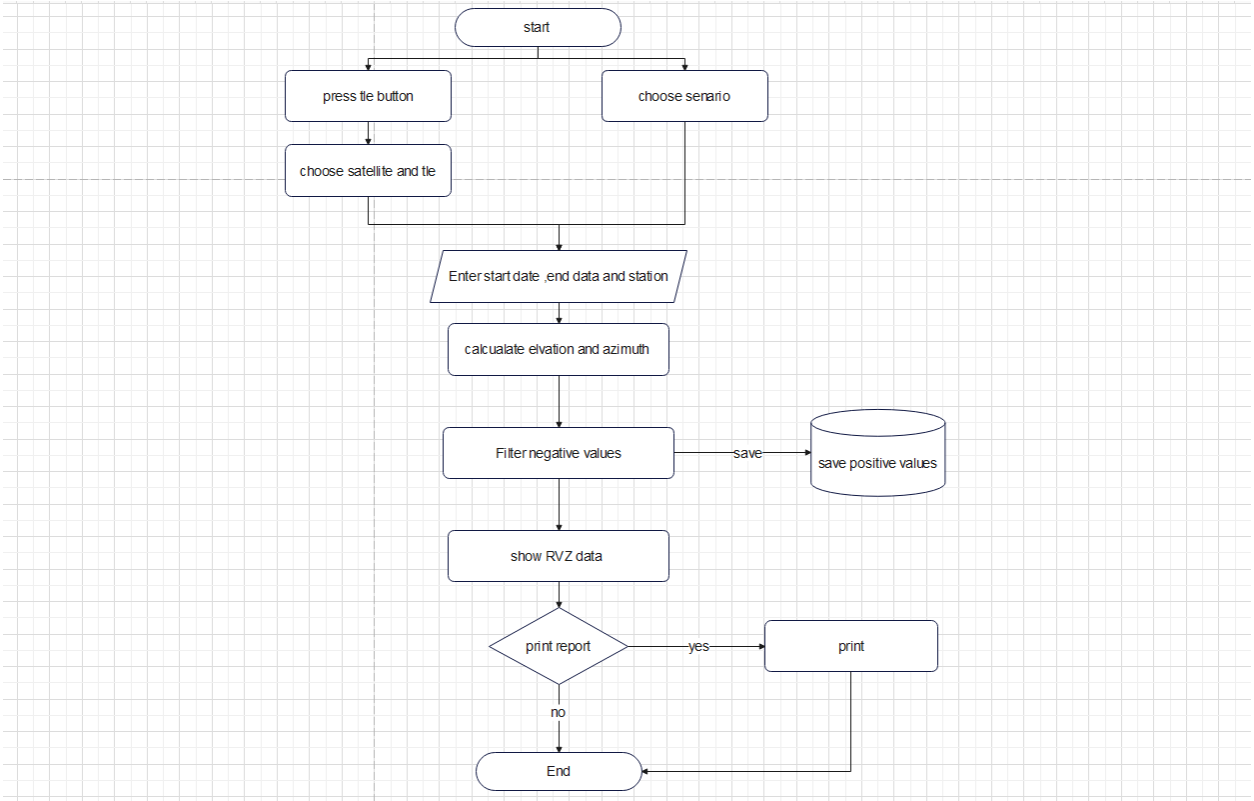


fig4.4 RVZ flowchart

4.1.2 Advantages of Flowchart

- **Effective Communication:** Flowcharts are a better way of communicating the logic of the system.
- **Effective Analysis:** Using flowchart problems can be analyzed more efficiently.
- **Easy Debugging and Efficient Testing:** The Flowchart helps in debugging and testing process.
- **Efficient Coding:** The flowcharts are very useful during program development phase.
- **Proper Documentation:** Flowcharts serve as a good program documentation, which is needed for various purposes.
- **Efficient Program Maintenance:** Maintenance of operating programs becomes easy with the help of flowchart.

4.1.3 Disadvantages of Flowchart

- **Complex Logic:** For complicated logic, flowchart becomes complex and clumsy.
- **Difficulty in Modifications:** If change is required in the logic, then flowchart needs to be redrawn and requires a lot of time.

4.2 UML Diagrams

4.2.1 What is a UML Diagram

UML is an acronym that stands for Unified Modeling Language. Simply put, UML is a modern approach to modeling and documenting software. In fact, it's one of the most popular business process modeling techniques.

It is based on diagrammatic representations of software components. As the old proverb says: "a picture is worth a thousand words". By using visual representations, we are able to better understand possible flaws or errors in software or business processes.

UML was created as a result of the chaos revolving around software development and documentation. In the 1990s, there were several different ways to represent and document software systems. The need arose for a more unified way to visually represent those systems and as a result, in 1994-1996, the UML was developed by three software engineers working at Rational Software. It was later adopted as the standard in 1997 and has remained the standard ever since, receiving only a few updates.

4.2.2 What are the uses of UML

Mainly, UML has been used as a general-purpose modeling language in the field of software engineering. However, it has now found its way into the documentation of several business processes or workflows. For example, activity diagrams, a type of UML diagram, can be used as a replacement for flowcharts. They provide both a more standardized way of modeling workflows as well as a wider range of features to improve readability and efficacy.

4.2.3 Types of UML Diagram

here are several types of UML diagrams and each one of them serves a different purpose regardless of whether it is being designed before the implementation or after (as part of documentation).

The two broadest categories that encompass all other types are the Behavioral UML diagram and the Structural UML diagram. As the name suggests, some UML diagrams try to analyze and depict the structure of a system or process, whereas others describe the behavior of the system, its actors, and its building components. The different types are broken down as follows:

Behavioral UML Diagram:

- Activity Diagram
- Use Case Diagram
- Interaction Overview Diagram
- Timing Diagram
- State Machine Diagram
- Communication Diagram
- Sequence Diagram

Structural UML Diagram

- Class Diagram
- Object Diagram
- Component Diagram
- Composite Structure Diagram
- Deployment Diagram
- Package Diagram
- Profile Diagram

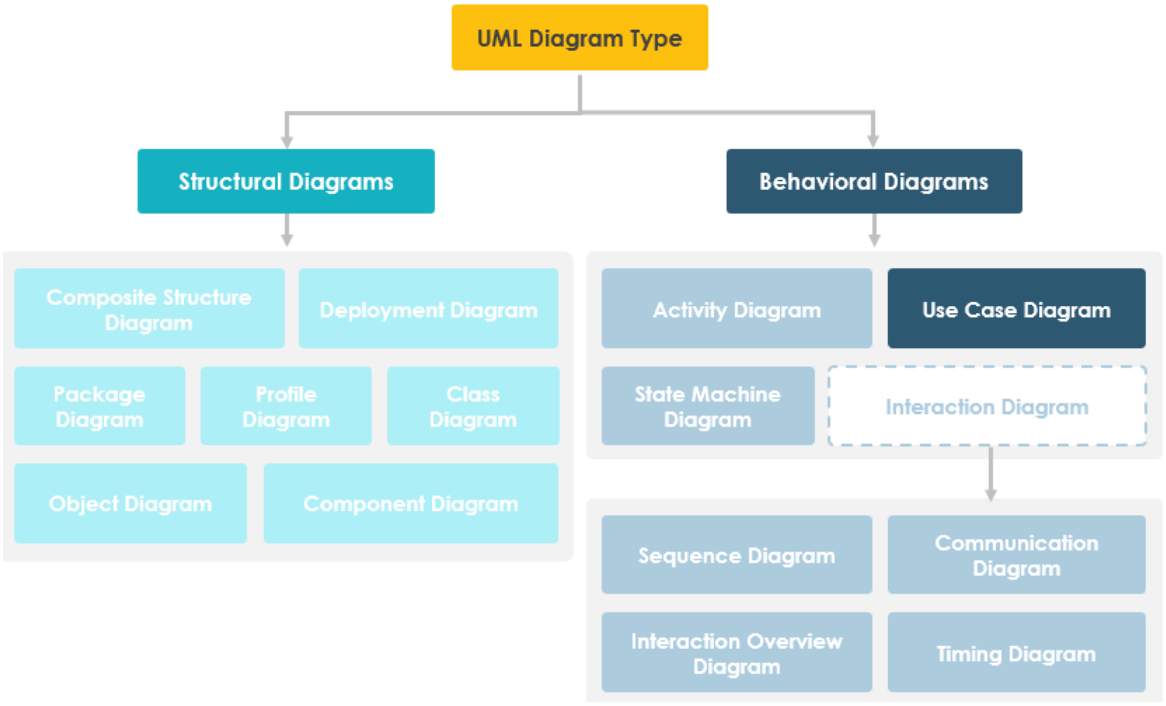


fig4.5 UML diagram

Not all of the 14 different types of UML diagrams are used on a regular basis when documenting systems and/or architectures. The Pareto Principle seems to apply in terms of UML diagram usage as well – 20% of the diagrams are being used 80% of the time by developers. The most frequently used ones in software development are Use Case diagrams, Class diagrams, and Sequence diagrams.[7]

4.2.3.1 Use case diagram

What is a use case diagram?

A use case diagram is a graphical depiction of a user's possible interactions with a system. A use case diagram shows various use cases and different types of users the system has and will often be accompanied by other types of diagrams as well. The use cases are represented by either circles or ellipses.

Purpose of Use Case Diagram:

Use case diagrams are typically developed in the early stage of development and people often apply use case modeling for the following purposes:

- Specify the context of a system
- Capture the requirements of a system
- Validate a systems architecture
- Drive implementation and generate test cases
- Developed by analysts together with domain experts

Symbols used in the UML use case Diagram:

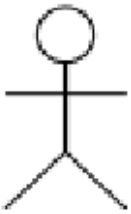
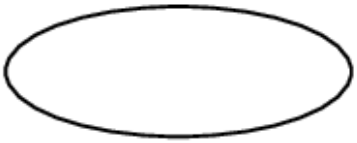
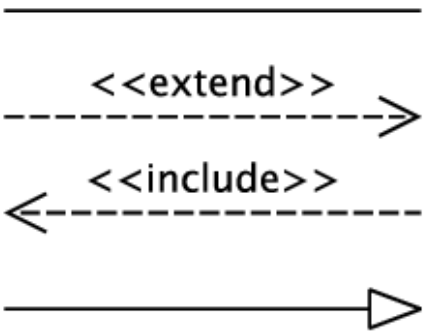
Symbol	Reference Name
	Actor
	Use case
	Relationship

Table 4.1 Use Case Symbols

ADVANTAGES:

- The use cases are mainly composed of narrative text. Hence, unlike many other modeling techniques, the non-technical stake holders (e.g., customers, end users, salesperson, etc.) are also able understand the model for the software system. This means that feedback can be obtained at a very early stage of the development from the customers and the end users.
- Another major advantage of use case modeling is that it requires the identification of exceptional scenarios for the use cases. This helps in discovering subtle alternate requirements in the system.
- The use case model can be utilized in several other aspects of software development as well, e.g., Cost Estimation, Project Planning, Test Case Preparation and User Documentation.
- The use case diagram provides a comprehensive summary of the whole software system in a single illustration.

DISADVANAGES:

- They do not capture the non-functional requirements easily.
- There might be a learning curve for the developer and/or specially, the client in using these use cases.

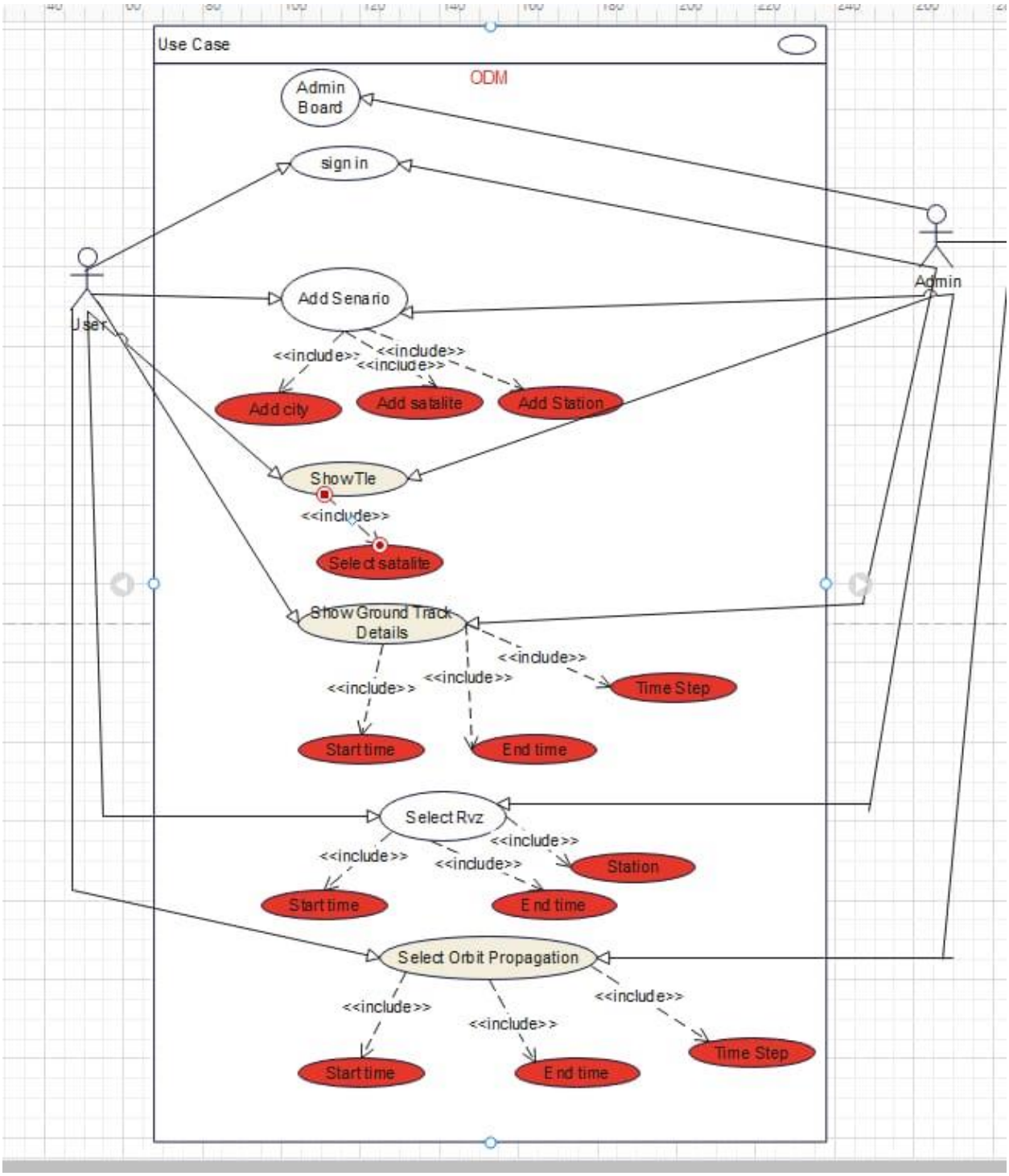


fig4.6 Use Case Diagram

4.2.3.2 Activity diagram

What is Activity Diagram:

The activity diagram is another important behavioral diagram in the UML diagram to describe dynamic aspects of the system. An activity diagram is essentially an advanced version of a flow chart that models the flow from one activity to another activity.

When to Use Activity Diagram:

Activity Diagrams describe how activities are coordinated to provide a service that can be at different levels of abstraction. Typically, an event needs to be achieved by some operations, particularly where the operation is intended to achieve a number of different things that require coordination, or how the events in a single-use case relate to one another, in particular, use cases where activities may overlap and require coordination. It is also suitable for modeling how a collection of use cases coordinates to represent business workflows

1. Identify candidate use cases, through the examination of business workflows
2. Identify pre-and post-conditions (the context) for use cases
3. Model workflows between/within use cases
4. Model complex workflows in operations on objects
5. Model in detail complex activities in a high-level activity Diagram

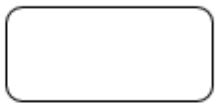
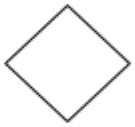
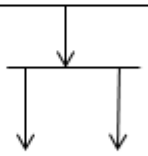
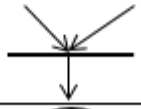
Sr. No	Name	Symbol
1.	Start Node	
2.	Action State	
3.	Control Flow	
4.	Decision Node	
5.	Fork	
6.	Join	
7.	End State	

Table4.2 Activity Diagram

Advantages:

- Complex stages or steps in a software system can be explained easily diagrammatically.
- Dynamic modeling of a software system.
- Each and every activity flow in the system can be explained as it is.
- Methods, functions, and operations can be explained in detail.
- Business processes and flows can be depicted easily.
- Simplified view, though the complex system.

- Business requirement analysis.
- The understanding of system requirements is explained in a lucid and simple manner.
- The workflow of the user and the system and the user with the system is explained in detail.

Disadvantage:

- The only drawback is the UML Activity Diagram is the messages or the communications between two components, or the user cannot be shown.

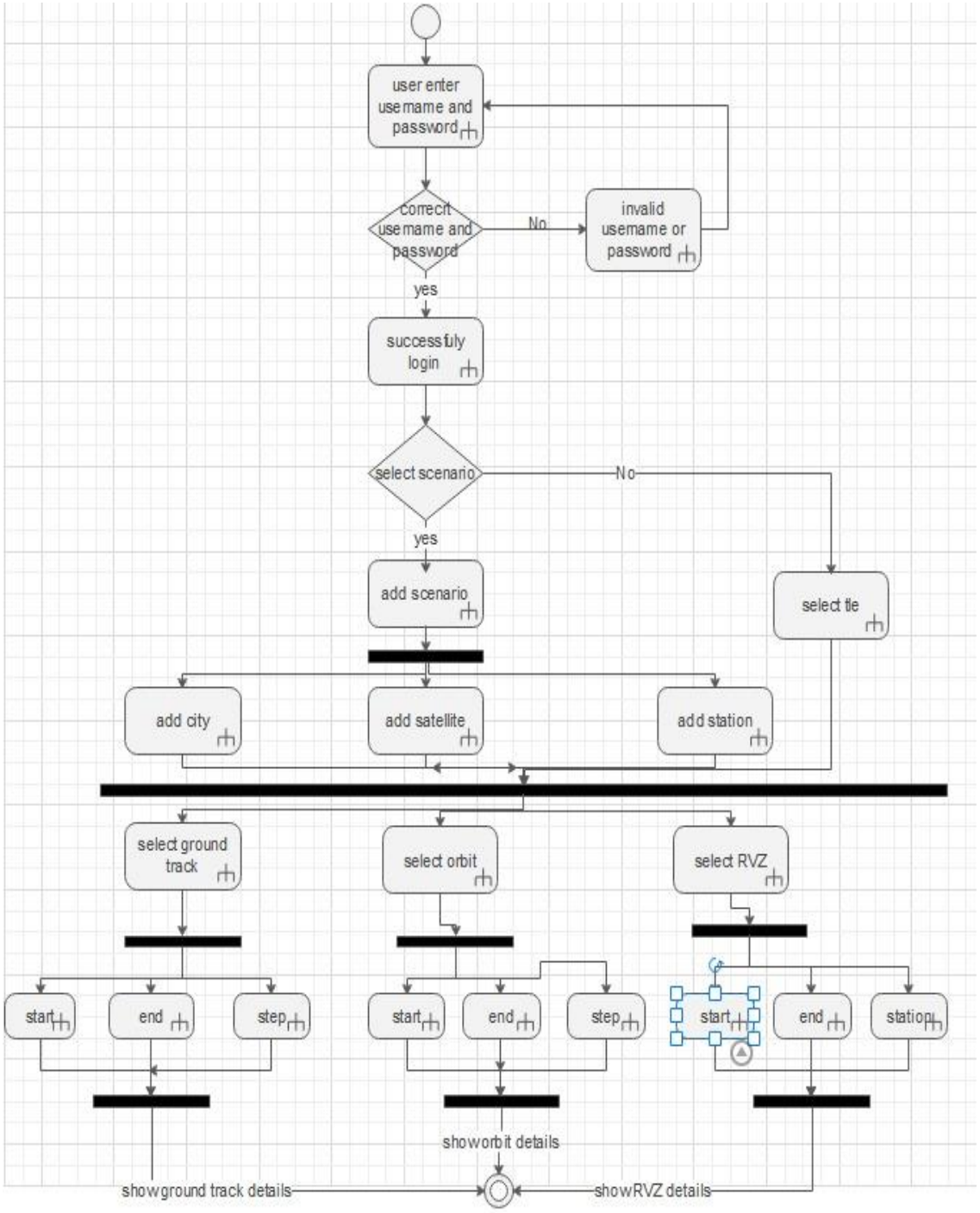


fig4.7 Activity Diagram

4.2.3.3 Sequence diagram

A sequence diagram is a type of interaction diagram because it describes how and in what order a group of objects works together. These diagrams are used by software developers and business professionals to understand requirements for a new system or to document an existing process. Sequence diagrams are sometimes known as event diagrams or event scenarios.

Benefits of sequence diagrams

Sequence diagrams can be useful references for businesses and other organizations. Try drawing a sequence diagram to:

- Represent the details of a UML use case.
- Model the logic of a sophisticated procedure, function, or operation.
- See how objects and components interact with each other to complete a process.
- Plan and understand the detailed functionality of an existing or future scenario.

The drawback of a Sequence Diagram

- In the case of too many lifelines, the sequence diagram can get more complex.
- The incorrect result may be produced, if the order of the flow of messages changes.
- Since each sequence needs distinct notations for its representation, it may make the diagram more complex.
- The type of sequence is decided by the type of message.

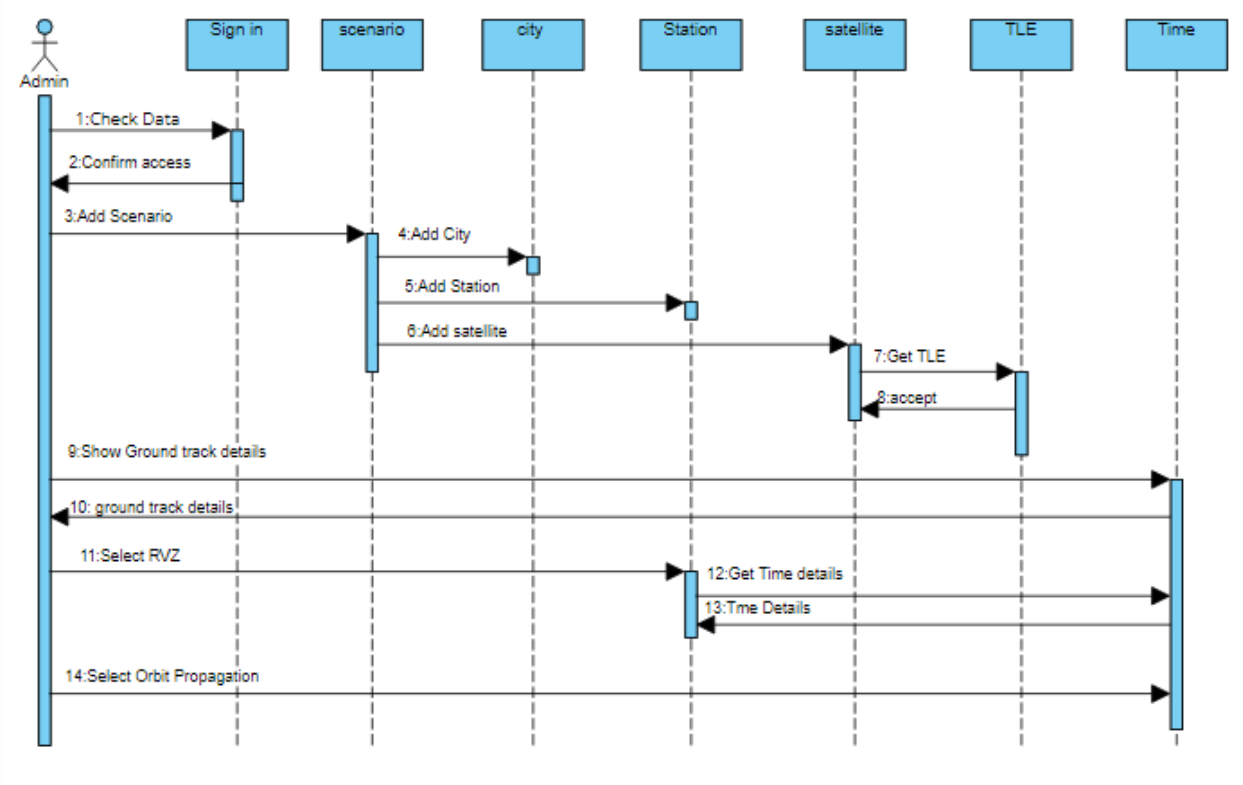


fig4.8 Sequence Diagram

4.2.3.4 Class diagram

What is Class Diagram?

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

Purpose of Class Diagrams:

1. Shows the static structure of classifiers in a system
2. Diagram provides a basic notation for other structure diagrams prescribed by UML
3. Helpful for developers and other team members too
4. Business Analysts can use class diagrams to model systems from a business perspective

Symbols used in the UML Class Diagram:

A UML class diagram is made up of:

- A set of classes

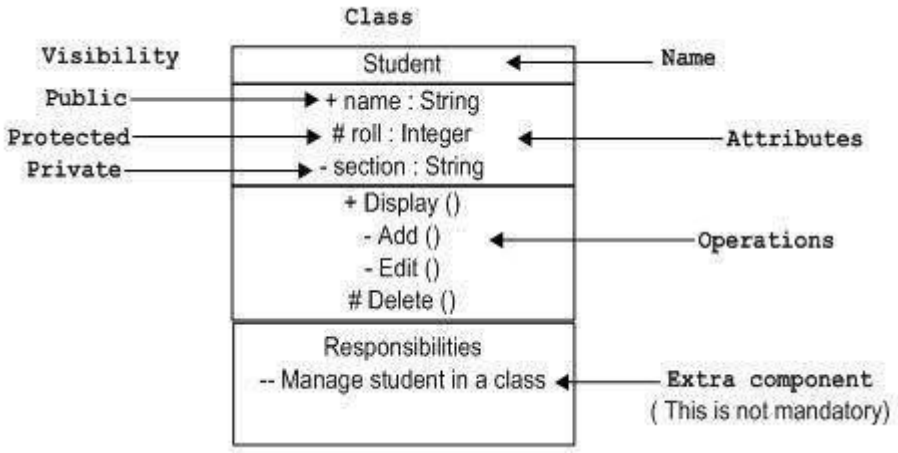


fig4.8 class Diagram

Class Diagram Relationship Type	Notation
Association	
Inheritance	
Realization/ Implementation	
Dependency	
Aggregation	
Composition	

Table4.3 relations Class Diagram

Advantages of Class Diagram:

- Any simple or complex data model could be illustrated using the class diagram to gain maximum information.
- The schematics of an application could be understood with the help of it.
- Any system needs could be visualized and passed across to the business for specific action to be taken.
- Any requirement to implement a specific code could be highlighted through charts and programmed to the structure described.
- A description that is implementation-independent could be provided and passed on to the components.

Disadvantages of Class Diagram:

Though Class Diagram is the first thing to consider in a production environment to build a flawless system, it certainly has its fair share of cons as well.

- The class diagrams might often take a longer time to manage, and maintain, which is sometimes annoying for a developer. It requires time for synchronization with the software code to set it up and maintain. Often developers or small companies find it difficult to synchronize the code as it requires an added amount of work.
- A lack of clarity in understanding the beneficiary of the diagram is also a disadvantage. As software developers work with code, sometimes the class diagrams are not that helpful much. However, project managers could benefit from the diagrams as they give an overview of the workflow of a particular tool. Hence, there is often an argument to not waste time on the class diagrams and focus rather on using a whiteboard or paper to draw the diagram.
- An overcomplicated or overwhelming diagram doesn't help software developers in their work. There could be situations when the developers are frustrated due to the structure of the class diagrams. Mapping out every single scenario could make the diagram messy and hard to work with. Using high-level information could somehow help to combat such issues.

- Putting overemphasis on design could cause a hindrance to the developers and companies. The stakeholders could easily overanalyze the problems after looking into the class diagram, and putting too much effort into the features of software might lead to a loss of focus. People need to get down to the actual work rather than spending time looking into the diagram and solving issues.
- As you can see, despite the importance of the Class Diagram in the software development life cycle, it is certainly not without any shortcomings and could make life difficult for the developers and companies if not used wisely.

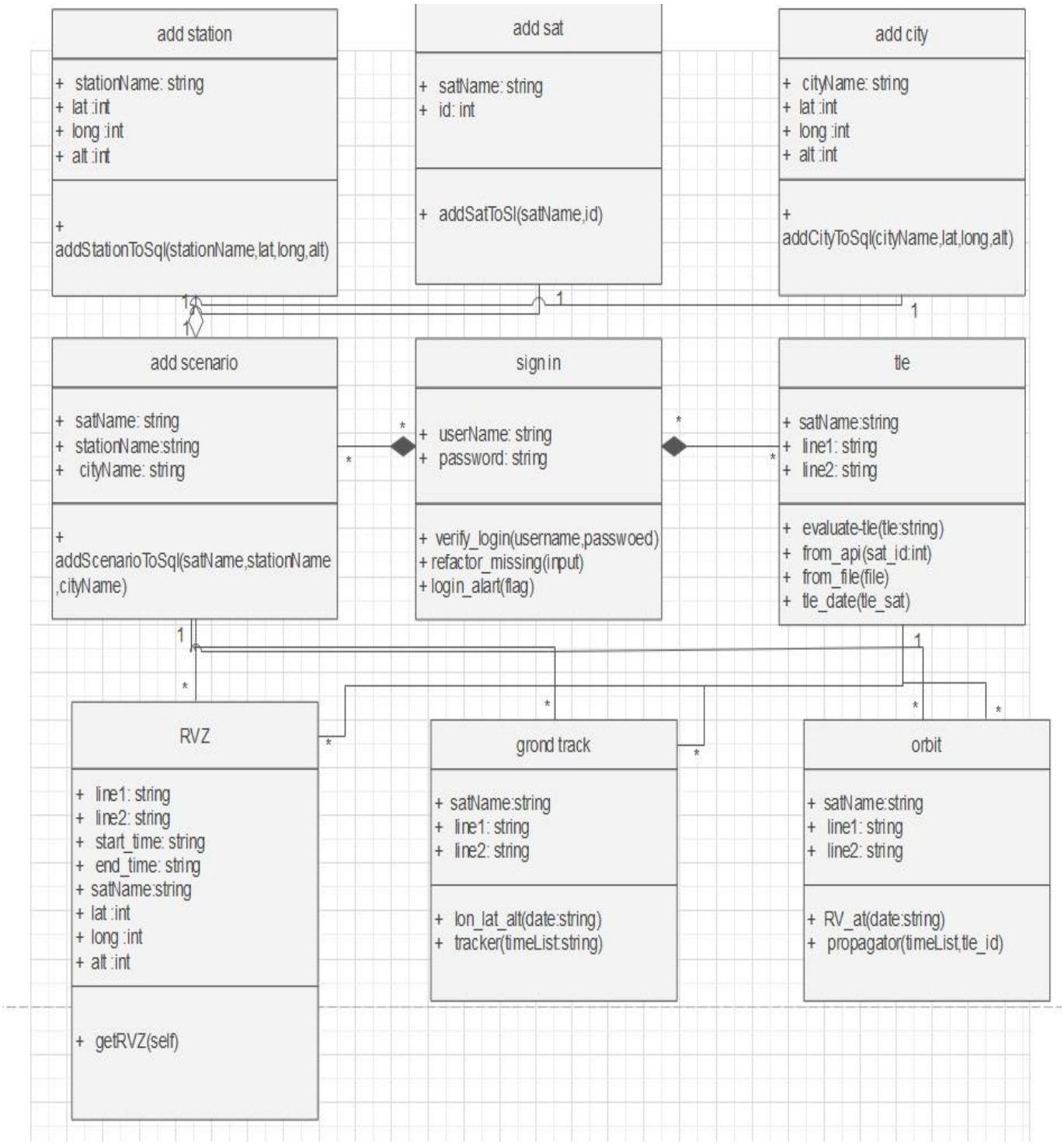


fig4.8 Class Diagram

4.2.3.5 State diagram

A state diagram is a diagram used in computer science to describe the behavior of a system considering all the possible states of an object when an event occurs. This behavior is represented and analyzed in a series of events that occur in one or more possible states. Each diagram represents objects and tracks the various states of these objects throughout the system.

When to Use State Diagram:

A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It's a behavioral diagram and it represents the behavior using finite state transitions. State diagrams are also referred to as State machines and State-chart Diagrams.

Symbols used in the UML State Diagram:

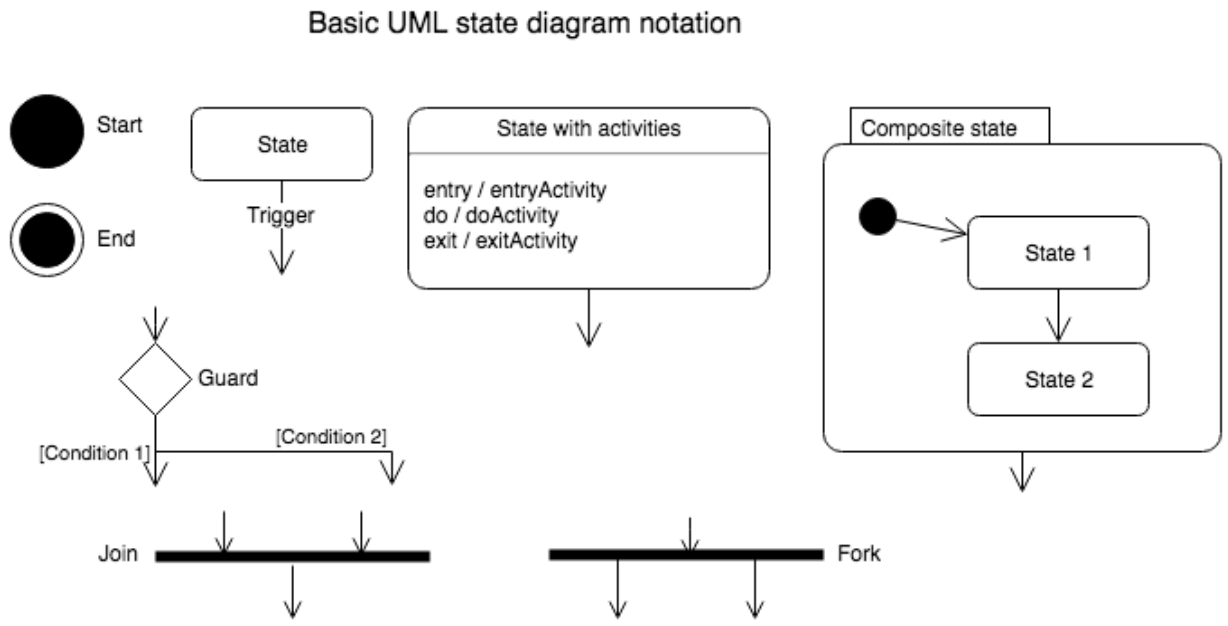


fig4.10 State Symbols

What are the Advantages of Using a State Design Pattern Model?

To begin with, a major advantage of the state model is its capability to minimize conditional complexity. It eliminates the need for it and switches statements on objects with different behavioral requirements, unique to

different state transitions. So, representing an object's state using a finite state machine diagram simplifies the conversion of the diagram into a state design model's types & method.

What are the Disadvantages of Using this Model?

A developer needs to write a large amount of code for the state schema. Depending on the number of different defined state transition methods and possible object states, you can write numerous methods. Thus, for N states with M transition methods, the total number of methods required will be (N+1) *M.

An insurance policy with 5 different states and 5 methods for each (ignoring the ListValidOperations method) requires 25 methods in total. The policy context type must also define the 5 state transition methods, increasing the total methods required to 30.

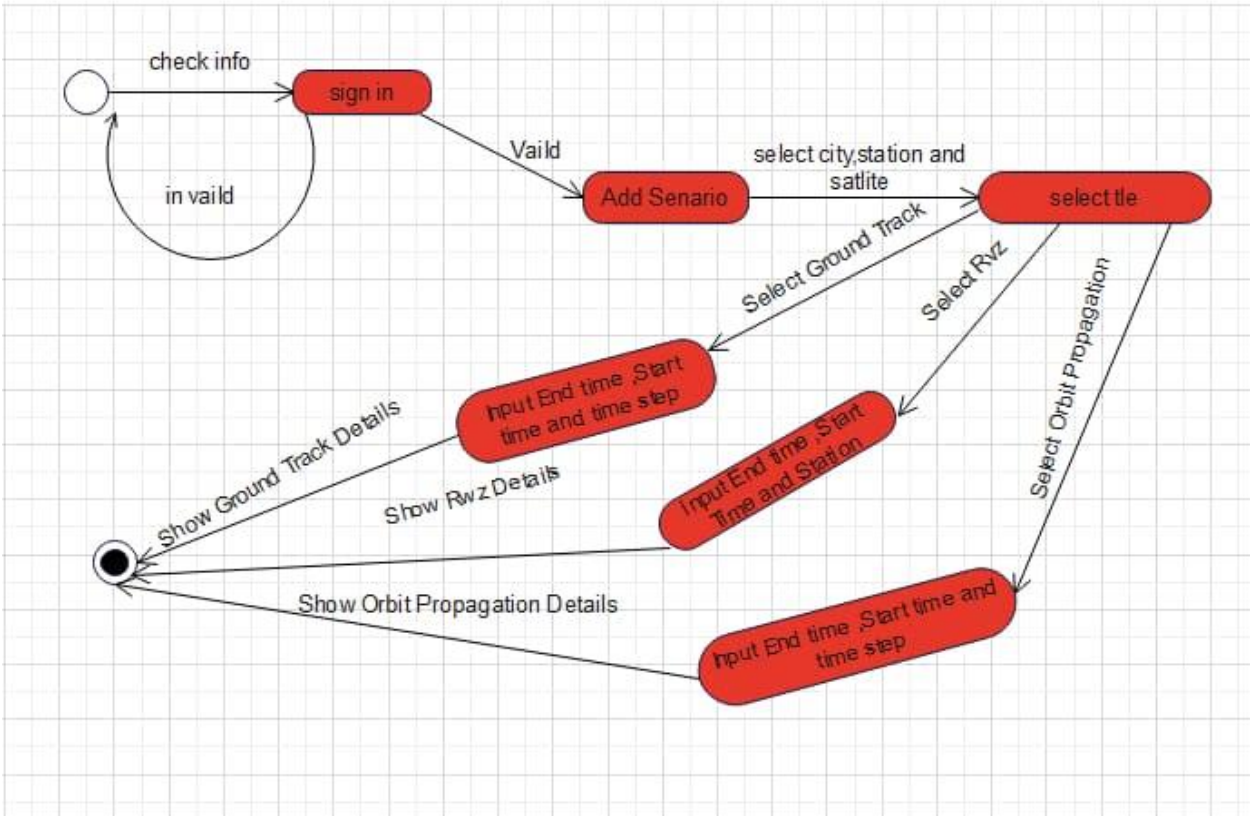


fig4.11 State Diagram

4.2.3.6 Object diagram

A UML object diagram represents a specific instance of a class diagram at a certain moment in time. When represented visually, you'll see many similarities to the class diagram.
An object diagram focuses on the attributes of a set of objects and how those objects relate to each other. For instance, in this object diagram below, all three bank accounts tie back to the bank itself. The class titles show the type of accounts (savings, checking, and credit card) that a given customer could have with this particular bank. The class attributes are different for each account type. For example, the credit card object has a credit limit, while the savings and checking accounts have interest rates. To take a closer look at this document
Object diagrams are not limited to banking use cases, however, as you can easily make an object diagram for family trees, corporate departments, or any other system with interrelated parts.

Uses of an Object Diagram

- Model the static design (similar to class diagrams) or structure of a system using prototypical instances and real data.
- Helps us to understand the functionalities that the system should deliver to the users.
- Understand relationships between objects.
- Visualizes, document, construct and designs a static frame showing instances of objects and their relationships in the dynamic story of life of a system.
- Verify the class diagrams for completeness and accuracy by using Object Diagrams as specific test cases.
- Discover facts and dependencies between specific instances and depict specific examples of classifiers.

Basic Object Diagram Symbols and Notations

Object Names:

- Every object is actually symbolized like a rectangle, that offers the name from the object and its class underlined as well as divided with a colon.

Fig4.12 Class

Object Attributes:

- Similar to classes, you are able to list object attributes inside a separate compartment. However, unlike classes, object attributes should have values assigned for them.

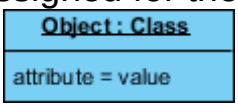


Fig4.13 Attribute

Links:

- Links tend to be instances associated with associations. You can draw a link while using the lines utilized in class diagrams.

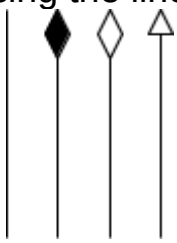


Fig4.14 Links

4.3 Front End Design

4.3.1 Figma

Figma is one of the most revolutionary graphics editing apps that's taking over the design world by storm. What makes it so attractive is the fact that it's free to use.

If you're still not familiar with this wondering web-based app or looking for tips on how to make the most of the software, you've come to the right place.

In this beginner guide, we outline some of the tutorials and video guides in one place. You can use them to learn everything you need to know about Figma without having to spend hours searching on Google and getting frustrated.

Let's get started.

Figma is a web-based graphics editing and user interface design app. You can use it to do all kinds of graphic design work from wireframing websites,

designing mobile app interfaces, prototyping designs, crafting social media posts, and everything in between.
Figma is different from other graphics editing tools. Mainly because it works directly on your browser. This means you get to access your projects and start designing from any computer or platform without having to buy multiple licenses or install software.
Another reason why designers love this app is that Figma offers a generous free plan where you can create and store 3 active projects at a time. It's more than enough for you to learn, experiment, and work on small projects.

4.3.2 How To use figma

As we mentioned earlier, Figma is a web-based app. All you need to start using the app is a desktop computer or a laptop with a good browser and an internet connection. Then you can visit **Figma website** to register a free account. And you can start working on your designs right away.
The official Figma team has put together a series of courses to teach you the basics of Design. It includes a dozen video tutorials with design exercises. If you're new to UI and UX design, these videos are worth exploring.
This is the site of figma
https://www.figma.com/

UI/UX Diagram

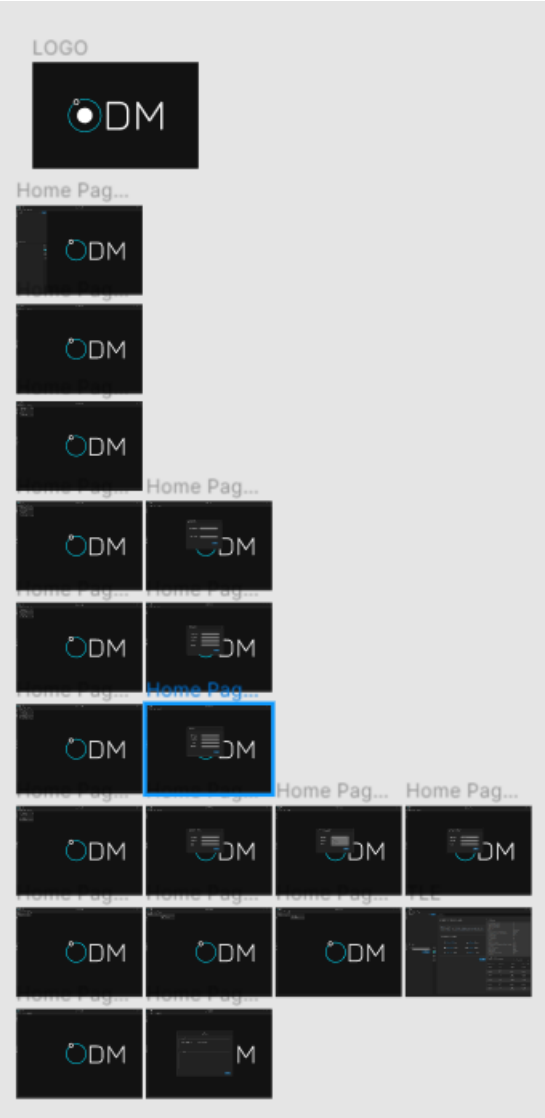

LOGO
Home Pag...
Home Pag...
Home Pag...
Home Pag...
Home Pag...
Home Pag...

Fig4.14 Home


Orbit porpagation
Orbit porpagation
Orbit porpagation

Fig4.15 Orbit propagation

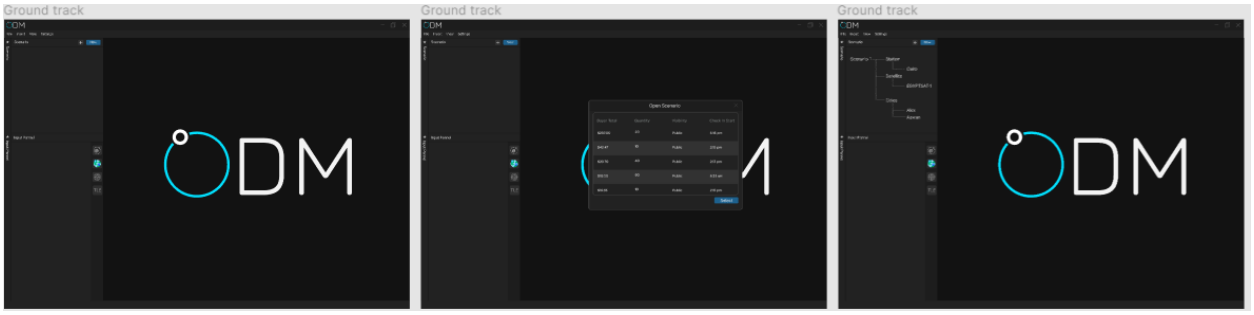


Fig4.16 Ground Track

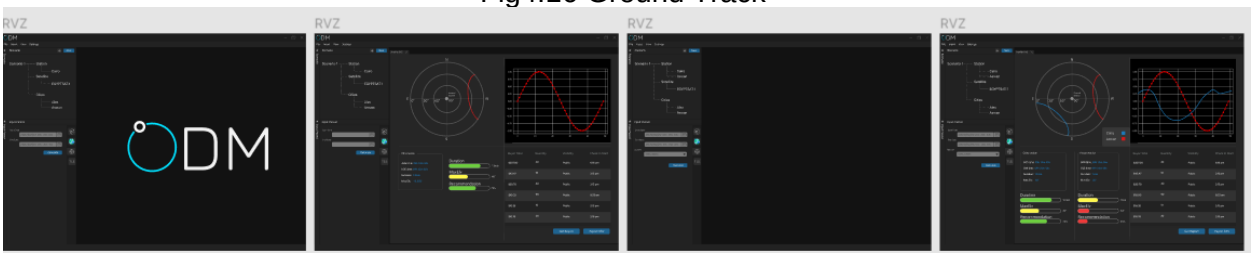


Fig4.17 RVZ

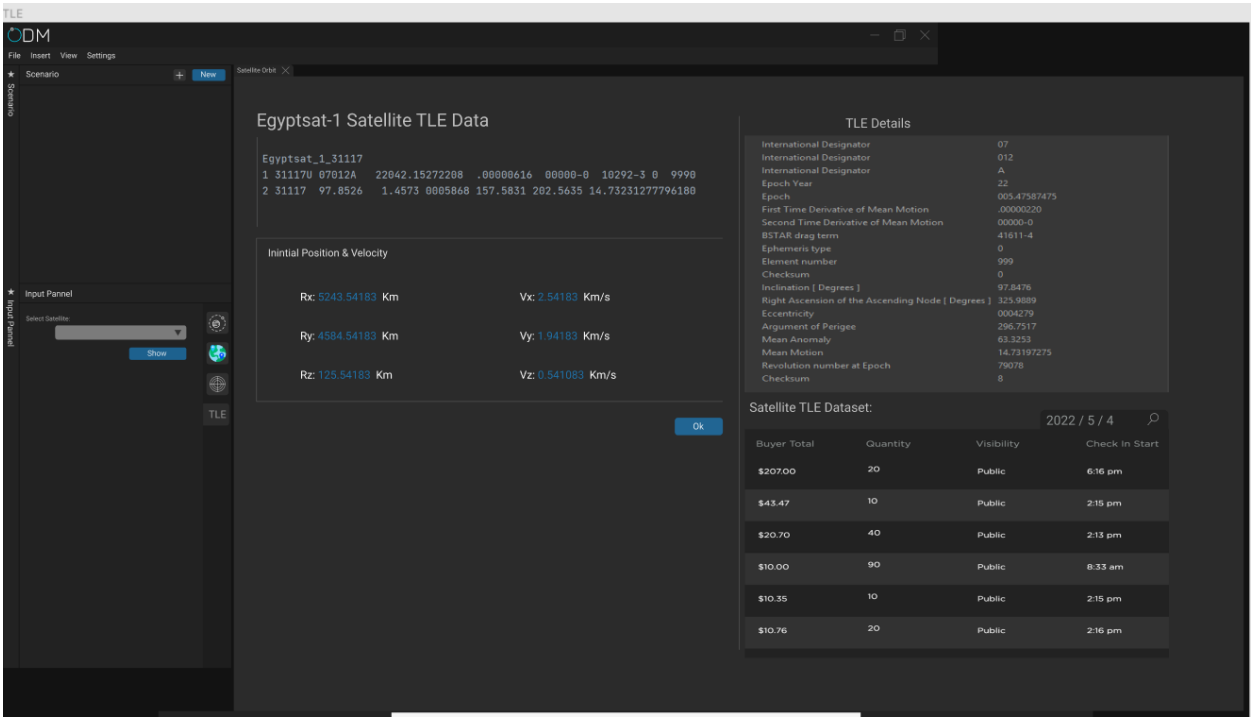
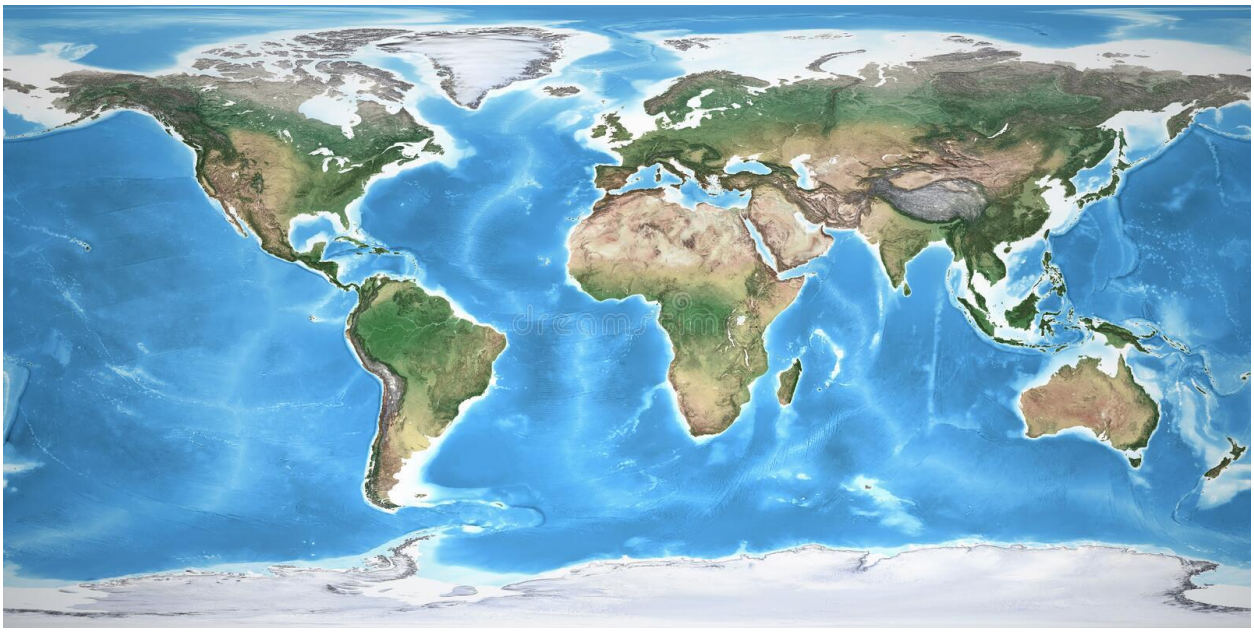

TLE
ODM
File Insert View Settings
Scenario
New
Satellite Orbit
Egyptsat-1 Satellite TLE Data
Egyptsat_1_31117
1 31117U 07012A 22042.15272208 .00000616 00000-0 10292-3 0 9998
2 31117 97.8526 1.4573 0005868 157.5831 202.5635 14.73231277796180
Inintial Position & Velocity
Rx: 5243.54183 Km
Vx: 2.54183 Km/s
Ry: 4584.54183 Km
Vy: 1.94183 Km/s
Rz: 125.54183 Km
Vz: 0.541083 Km/s
Ok
Input Pannel
Select Satellite
Show
TLE
TLE Details
International Designator 07
International Designator 012
International Designator A
Epoch Year 22
Epoch 005.47587475
First Time Derivative of Mean Motion .00000220
Second Time Derivative of Mean Motion 00000-0
BSTAR drag term 41611-4
Ephemeris type 0
Element number 999
Checksum 0
Inclination [Degrees] 97.8478
Right Ascension of the Ascending Node [Degrees] 325.9889
Eccentricity 0004279
Argument of Perigee 296.7517
Mean Anomaly 63.3253
Mean Motion 14.73197275
Revolution number at Epoch 79078
Checksum 8
Satellite TLE Dataset:
2022 / 5 / 4
Buyer Total Quantity Visibility Check In Start
$207.00 20 Public 6:16 pm
$43.47 10 Public 2:15 pm
$20.70 40 Public 2:13 pm
$10.00 90 Public 8:33 am
$10.35 10 Public 2:15 pm
$10.76 20 Public 2:16 pm

Fig4.18 TLE

Chapter 5 Database

We will study in this chapter

- It's all about database
- Why did we specialize Microsoft SQL Server?
- Tables for the project's database
- Dedicated anal for each table



A database, in the most general sense, is an organized collection of data. More specifically, a database is an electronic system that allows data to be easily accessed, manipulated and updated.
In other words, a database is used by an organization as an electronic way to store, manage and retrieve information. The database is one of the cornerstones of enterprise IT, and its ability to organize, process and manage information in a structured and controlled manner is the key to many aspects of modern business efficiency.
However, database go way beyond simply storing data. As we'll see later, the inherent logic and efficiency in how the data is stored and retrieved can provide an incredibly powerful business tool to an organization.

5.1 Microsoft SQL Server

So, we used SQL language in our project.
concept of SQL has been around for four decades now. However, it has been evolving and continues to evolve. We all studied SQL at some point or another in our education. We are here to make it better by moving away from the archaic tools and practices.

Why is SQL important? What problem is it solving?

Database administration or data management is incomplete without SQL. For comfortable use of SQL as part of your administration or development requires that you understand the basics of SQL, which will take you a long way in your career.
You will also understand that SQL is a special-purpose programming language; special-purpose, as in, it is different from the general-purpose programming languages such as C, C++, Java/JavaScript, etc., meaning, it has a very particular purpose: manipulation of datasets. And this manipulation happens using what is known as Relational Calculus.
But isn't studying SQL alone restrictive? Turns out, it isn't. Of course, we can use SQL on any kind of database or data source, but even if we cannot directly use SQL, most query languages of today have some relationship to SQL. In general, once you know SQL, you can effortlessly pick up other query languages too.
Standards are vital because every relational database must build its framework around this framework in order to ensure compatibility. This means that the learning curve is greatly reduced. SQL is ANSI as well as ISO-compliant, along with other standards, which emphasize the fact that you have to learn the concept only once.

5.2 Tables Of database

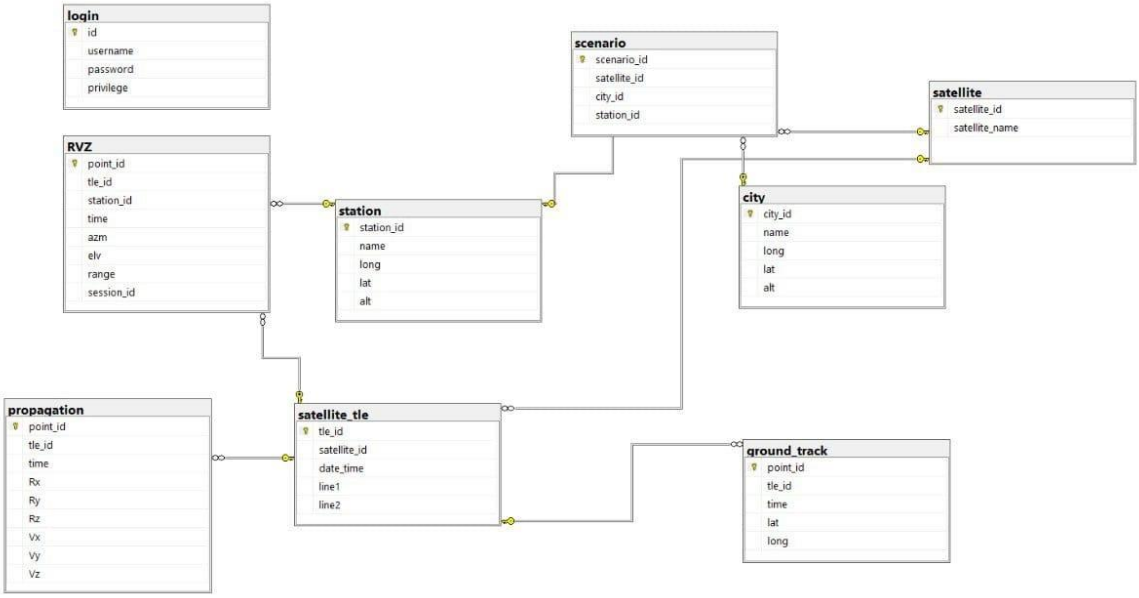


Fig5.1database

The contents of our database are:

Scenario: save all data that activation for our project.

Satellite: each satellite connects with a special TLE.

Satellite TLE: each TLE has a special time, line 1,2 and related to his satellite.

Station: connect with the scenario to calculate RVZ and content which name for satellite, long, and alt for know where the station and when it Cross with the path of satellite and every detail about timing and crossing for satellite and station.

RVZ: to calc it should to get point time “each point inside path of satellite
azm ‘the line connects with station and satellite for the north side’
Elv ‘the angle of inclination between the station and the satellite for horizontal tilt’
to know where the antenna goes

Propagation: main relation is TLE which active for now and take (initial villosity, initial position) to calculation orbit which has (time of each point, Rx, Ry, Rz, Vx, Vy, Vz)

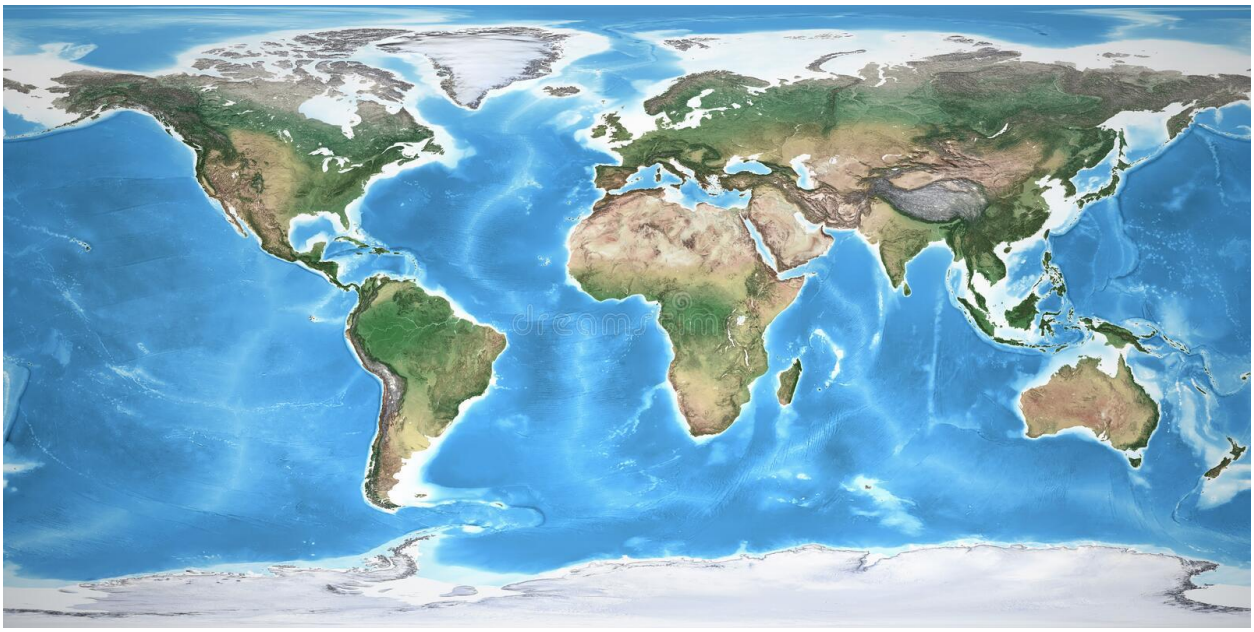
Ground track: calc long, alt, lat that determined where is it located in relation to latitude and longitude and its height from the ground.

Login: to more security for our project that has user name and password

Chapter 6 Prototype

We will study in this chapter:

- Detailed explanation of the project Interface.
- How to use the project step by step.



Home page:

This page starts all options in system
If user entered correct username and password home page is opened

Fig 6.1 home-page

admin dashboard

This page for admin to control all data in the system (Add, Update, delete users and to view all users)

in our case the user is an admin, so the admin menu is available for him
In admin dashboard admin have:

1. Add button to add new user.
2. Update button to update existing users.
3. Delete button to delete users.

4. User info panel to get data for new users or change data of existing users.
5. Users table to view all users in database.

Fig 6.2 admin dashboard

Login

- This page to check user information
  If true check if information about user
take it for next page
else about admin
add new page “administrator” to get
more access for admin
  Else can’t be register without admin

In action of starting the application the first thing to appear is the login form:
The user Enters his (preassigned username and password) to access the system

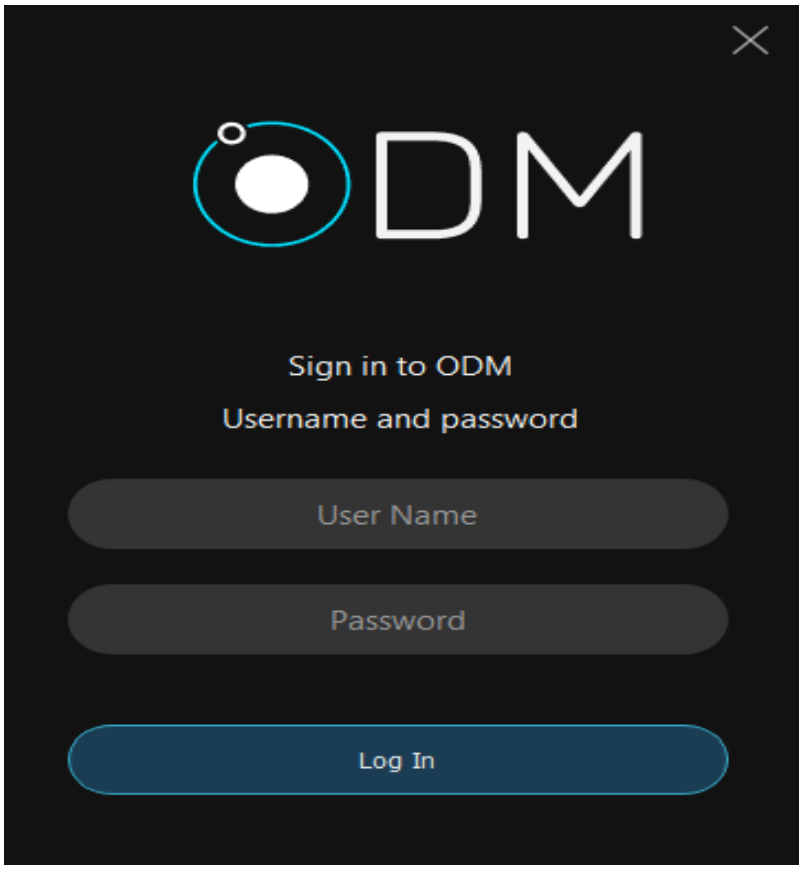


Fig 6.3 Login

add scenario:

This window to add more options for satellite to get some information about special area

Selected scenario panel

1. Selected City
2. Selected station
3. Selected Station

Fig 6.4 add scenario

satellite TLE

This page to show which TLE is selected from database.
When selecting satellite from database a confirmation message is displayed on the screen:

Then TLE tab is opened and give the following insights:

1. Satellite selection input panel.
2. TLE page.
3. Active TLE three lines.
4. Active TLE initial position and velocity.
5. Active TLE details.
6. Selected satellite available TLE sets.

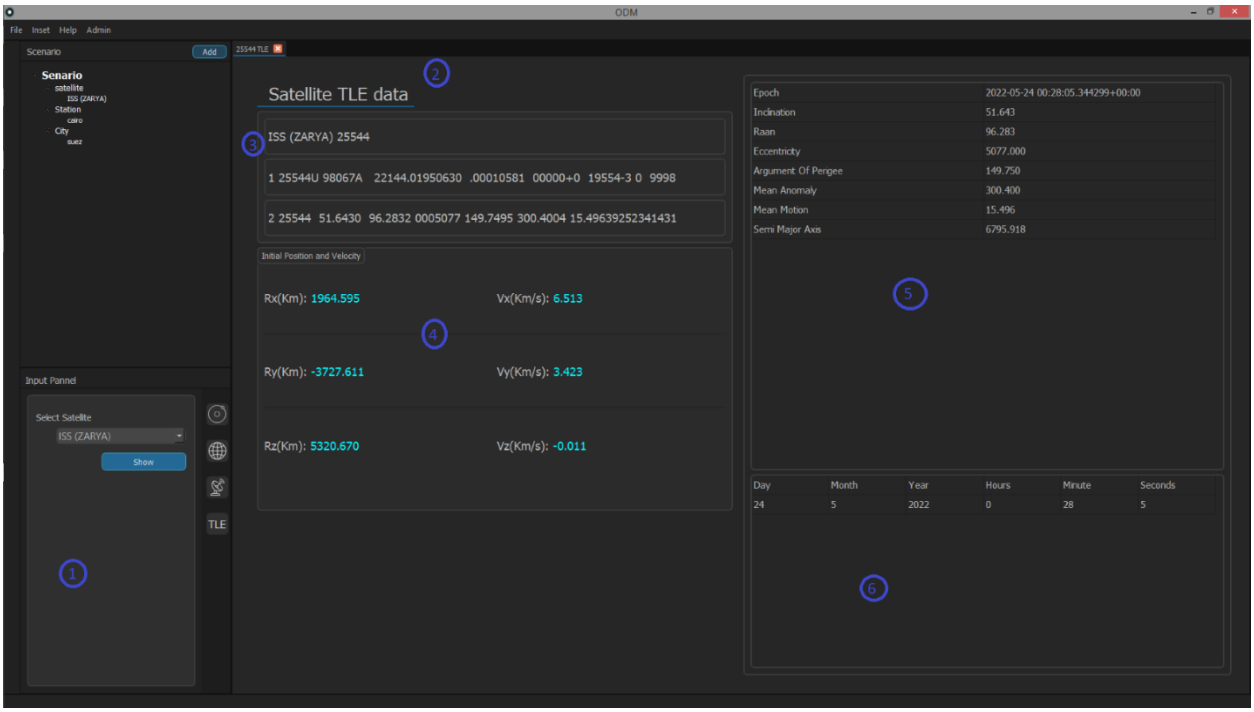


Fig 6.5 satellite TLE

ground-track

This page shows the map world, satellite position, satellite orbit, all information panels and all reports panels.

When doing Ground track for a satellite the **Ground** tab is opened and gives the following insights:

1. Ground track input panel.

2. Map of the world that show:
   - Satellite position on the earth **(denoted by red point).**
   - satellite orbit which the selected point is part of **(denoted by red curve)**.
3. Animation control panel **(timeline).**

4. information panel:
   - ground track calculation starts and end time.
   - Total number of orbit satellite do in this calculation period.
   - Current orbit displayed on the map (of the selected point).

- Longitude, latitude, time (of the currently selected point).

5. Ground track pointes table (time, longitude, latitude).

6. Reports panel:
   - Get report button to report a **.word** file.
   - Export csv button to report a **.csv** file of the propagation.
   - Search by position button.
   - Search by time button.

7. Options panel that has two modes:
   - **Animation mode** and you can optimize:
     - number of orbits to display.
   - **Point-time mode** in it you can search with:
     - Position (search to know when satellite pass on a certain position).
     - Position (search to know where satellite will be in a specific time).

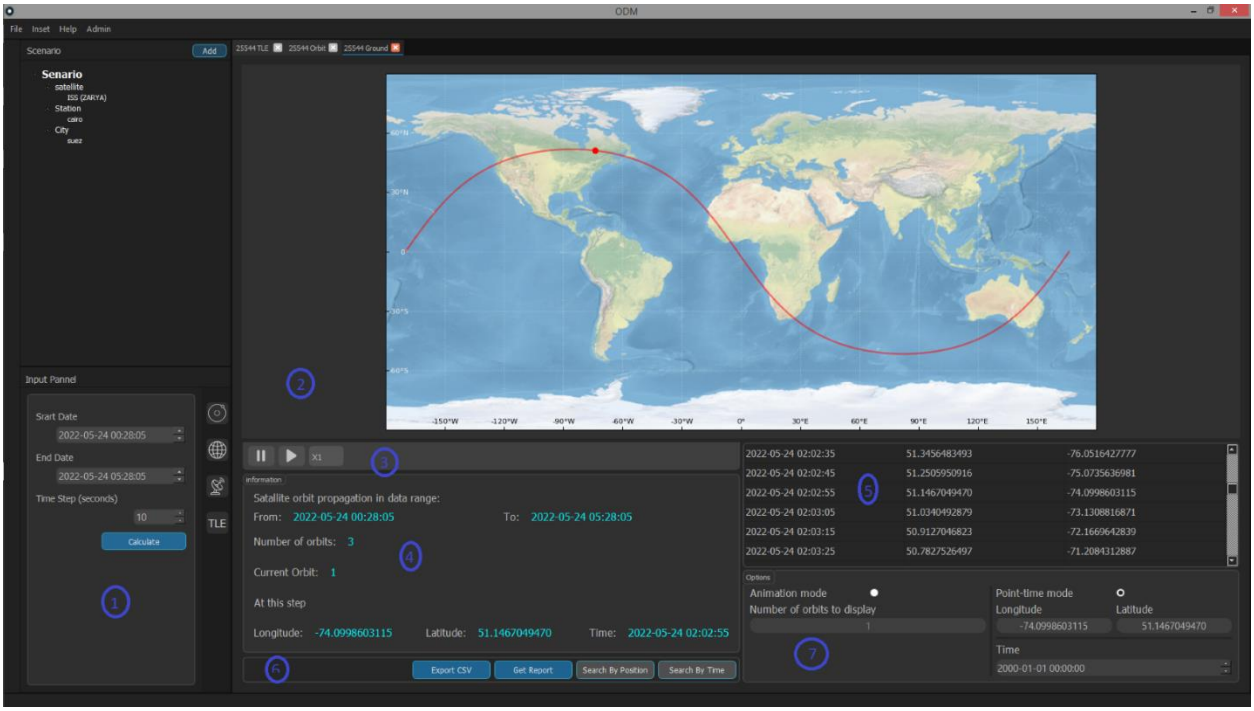


Fig 6.6 satellite Tle

RVZ

This page shows session visualization figure, session acquisition time (AOS), session lose time (LOS), session duration insights, reports panel

When doing RVZ for a satellite the **RVZ** tab is opened and gives the following insights:

1. RVZ input panel.
2. Session visualization figure.
3. Elevation visualization figure.
4. Available sessions table.
5. Information panel:
    - Session Acquisition time (AOS).
    - Session Lose time (LOS).
    - Session Duration.
    - Session maximum Elevation angle (Max-Elv).
6. insights panel:
    - session duration progress bar.
    - maximum Elevation angle progress bar.
    - how much this session is recommended to connect satellite at.

7. Reports panel:
    - Get the report button to report a **.word** file.
        - Export csv button to report a **.csv** file of the propagation.

Fig 6.7 rvz

orbit propagation

This page shows propagation pointes table and orbit propagation input panel.

When doing Orbit propagation for a satellite the **Orbit** tab is opened and give the following insights:

1. Orbit propagation input panel.
2. Propagation pointes table (time, position: x y z, velocity: x y z).
3. Information panel show details of the selected point.
4. Reports panel:
    - Get the report button to report a **.word** file.
    - Export csv button to report a **.csv** file of the propagation.
5. Satellite orbit display area and its control panel (reserved for future work).

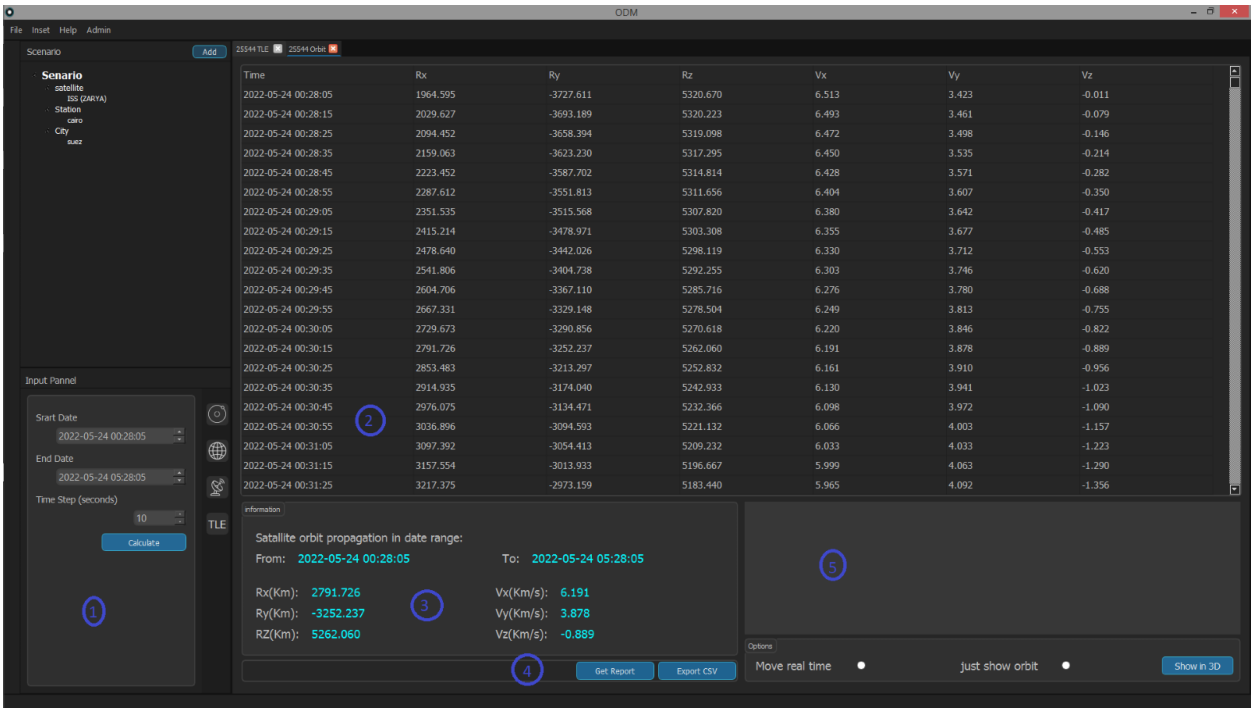


Fig 6.8 orbit propagation

6.1.9 Sample code:

In this section we will show some of our code

6.1.9.1 Updata tle:

```python
import os
import requests
from skyfield.api import EarthSatellite, load
import numpy as np
from PyQt5.QtWidgets import QMessageBox

class UpdateTle:
    """
    class UpdateTLE is used to read the TLE set and handle any possible Error that may occur
    """

    @staticmethod
    def from_api(satellite_id):
        """
        read TLE set from API
        :param satellite_id: (string)
        the satellite unique number that exist as an identifier in TLE set
        :return:
        latest available TLE set
        """
```

Fig 6.9 Update tle class

```
22      try:
23          api_request = 'https://tle.ivanstanojevic.me/api/tle/{}'.format(satellite_id)
24          response = requests.get(api_request)
25          responses = response.json()
26          name, line1, line2 = responses["name"], responses["line1"], responses["line2"]
27          tle = '{}\n{}\n{}'.format(name, line1, line2)
28          return tle
29
30      except:
31          dialog = QMessageBox()
32          dialog.setWindowTitle("Update TLE")
33          dialog.setText(f"satellite {satellite_id} not found or there is no internet")
34          dialog.setIcon(QMessageBox.Warning)
35          dialog.exec_()
36
```

Fig 6.10 Update tle class

```
38  def from_file(self, file):
39      """
40      read TLE set from local file
41      :param file: (str)
42      TLE set file path
43      :param three_lines: (bool)
44      if TLE fie has three or two lines
45      :return: (str)
46      TLE set read from file
47      """
48      # get tle from local file and check it's values
49      with open(file, 'r') as text_file:
50          if os.stat(file).st_size == 0:
51              raise Exception("Sorry, file is empty")
52          else:
53              tle = text_file.read()
54              return tle
```

Fig 6.11 Update tle class

```
94      @staticmethod
95      def tle_details(line1,line2):
96          #get data from two lines
97
98          """
99              return TLE details from line1 and line2 like time , satellite_id, inclination,
                raan, eccentricity, arg_of_perigee, mean_anomaly, mean_motion, semi_major_axis
100         """
101
102         ts = load.timescale()
103
104         satellite = EarthSatellite(line1, line2, 'ISS (ZARYA)', ts)
105
106         mu = 3.98600441 * (10 ** 14)  # standard gravitational parameter
107
108
109         time = satellite.epoch.utc_datetime()
110
111         id = int(line2[2:7])
```

Fig 6.12 Update tle class

```
112
113         inclination = float(line2[8:16])
114
115         raan = float(line2[17:25])
116
117         eccentricity = float(line2[26:33])
118
119         arg_of_perigee = float(line2[34:42])
120
121         mean_anomaly = float(line2[43:51])
122
123         mean_motion = float(line2[52:63])
124
125         semi_major_axis = np.cbrt(mu / (((mean_motion * 2 * np.pi) / 86400) ** 2)) / 1000
126
127         data = np.array([time,inclination, raan, eccentricity, arg_of_perigee, mean_anomaly,
            mean_motion,semi_major_axis])
128         return data
```

Fig 6.13 Update tle class

In these photos show update tle class:

- This class is used to read the tle and handle any possible error that may occur
- API function read TLE set from API, take parameter satellite-id and return latest available TLE set
- From_file function read TLE set from local file, parameter file and return TLE set read from file
- Tle_details return TLE details from line1 and line2 like time, satellite_id, inclination, raan,….. etc.

6.1.9.2 Orbit propagation:

```python
from skyfield.api import load, EarthSatellite
from skyfield.framelib import itrs
from datetime import datetime
from .tools import Tools
from src.python_tools.sql import conn
import time


class OrbitPropagation:
    """
    class OrbitPropagation is used to propagate (position and velocity) orbiting satellite from
    two-line-element (TLE)
    set in International Terrestrial Reference System (ITRS) reference frames for certain
    amount of time
    Parameters:
    -------------
    line1: string (69char)
    first line of the two-line-element (TLE) set (expected to be exactly 69 characters long)

    line2: string (69char)
    second line of the two-line-element (TLE) set (expected to be exactly 69 characters long)
```

Fig 6.14 Orbit propagation class

```
    satellite name: string
    satellite name is in TLE set in three lines

    Examples
    --------
    >>  # time list for points of satellite orbit
        time_list = np.array(
            [[2022, 2, 12, 2, 9, 10], [2022, 2, 12, 2, 9, 11]])
        # satellite TLE set
        satellite_name = 'Egyptsat_1_31117'
        s = '1 31117U 07012A   22042.15272208  .00000616  00000-0  10292-3 0  9990'
        t = '2 31117  97.8526   1.4573 0005868 157.5831 202.5635 14.73231277796180'

        # create satellite object
        test = OrbitPropagation(s, t, satellite_name)

        # propagate orbit
        position, velocity = test.propagator(time_list)
```

Fig 6.15 Orbit propagation class

```
        for x, y in zip(position, velocity):
            print('position: {},velocity: {}'.format(x, y))

    position: [-1855.8527673  -1190.91640693 -6684.64997705],velocity: [-7.26208326 -0.
    57318663  2.11530683]
    position: [-1863.11385743 -1191.48839079 -6682.53084718],velocity: [-7.26006131 -0.
    57077894  2.12293017]
    """

    def __init__(self, line1, line2, satellite_name):
        self.line1 = line1
        self.line2 = line2
        self.satellite_name = satellite_name
        self.ts = load.timescale()
        # create EarthSatellite object
        self.satellite = EarthSatellite(self.line1, self.line2, self.satellite_name, self.ts)
        self.cursor = conn.cursor()

    def rv_at(self, date):
        """
        calculate position and velocity of an orbiting satellite
```

Fig 6.16 Orbit propagation class

```
        :parameter date: (list_like)
        the time in which position and velocity are calculated
        :returns
         position and velocity
        """

        # get the geocentric information of the satellite in exact moment
        year, month, day, hour, minute, second = Tools.analyse_single_point(date)
        geocentric = self.satellite.at(self.ts.utc(year, month, day, hour, minute, second))

        # change reference frame from (TEME) to (ITRS)
        geocentric = geocentric.frame_xyz_and_velocity(itrs)
        point_position = geocentric[0].km
        point_velocity = geocentric[1].km_per_s

        return point_position, point_velocity

    def propagator(self, timelist, tle_id):
        """
        propagate satellite orbit for certain amount of time
        :parameter timelist: (list_like)
```

Fig 6.17 Orbit propagation class

```
        list of dates position and velocity are calculated in  to form an orbit
        :return:
        satellite position, velocity (numpy array)
        """
        start = time.time()
        # find position, velocity for each date
        for date in timelist:
            r, v = self.rv_at(date)
            tt = datetime(date[0], date[1], date[2], date[3], date[4], date[5])
            self.cursor.execute("INSERT INTO propagation (tle_id,time,Rx,Ry,Rz,Vx,Vy,Vz) values
            (?,?,?,?,?,?,?,?)",
                                (tle_id, tt, r[0], r[1], r[2], v[0], v[1], v[2]))
        conn.commit()
        end = time.time()
        print(f"orbit propagation time = {end - start}")
```

Fig 6.18 Orbit propagation class

In these photos show orbit propagation class:

- This class is used to propagate (position and velocity) orbiting satellite from TLE.
- Parameters are line1, line2, satellite name.

- Rv_at function calculates the position and velocity of an orbiting satellite, parameter date and return position, and velocity.
- Propagator function propagates satellite orbit for a certain amount of time, parameter timelist, and returns satellite position, velocity.

6.1.9.3 RVZ:

```python
from skyfield.api import load, wgs84
from skyfield.api import EarthSatellite
import pandas as pd


class RVZ:
    """
    class RVZ (radio visibility zone) is used to get Azimuth,elevation ,distance from
    two-line-element (TLE)
    by setting ground station position (latitude, Longitude and elevation).

    Parameters:
    ------------
    line1: string (69char)
    first line of the two-line-element (TLE) set (expected to be exactly 69 characters long)

    line2: string (69char)
    second line of the two-line-element (TLE) set (expected to be exactly 69 characters long)

    startTime : datetime
    start time to get Azimuth,elevation ,distance
```

Fig 6.19 RVZ class

```
21
22      endTime : datetime
23      end time to get Azimuth,elevation ,distance
24
25      step : integer
26      step -> seconds like 1s,2s
27
28      satelliteName: string
29      satellite name is in TLE set in three lines
30
31      Examples
32      --------
33      line1 = '1 31117U 07012A   22042.15272208  .00000616  00000-0  10292-3 0  9990'
34      line2 = '2 31117  97.8526   1.4573 0005868 157.5831 202.5635 14.73231277796180'
35
36      obj = RVZ(line1,line2,"2022-2-11 11:50:49.188","2022-2-11 11:55:49.188","1",
        "Egyptsat_1_31117")
37
38      allValues = obj.getRVZ()
39      print(allValues)
40      """
```

Fig 6.20 RVZ class

```
41
42  def __init__(self, line1, line2, startTime, endTime, step, satelliteName, lat, long, alt):
43      self.line1 = line1
44      self.line2 = line2
45      self.startTime = startTime
46      self.endTime = endTime
47      self.step = step
48      self.satelliteName = satelliteName
49      self.lat = lat
50      self.long = long
51      self.alt = alt
52
53  def getRVZ(self):
54      allValues = []  # empty list to get list of all values during start and end time [[1,2,
        3],[4,5,6]]
55      ts = load.timescale()
56      satellite = EarthSatellite(self.line1, self.line2, self.satelliteName, ts)
57
58      bluffton = wgs84.latlon(self.lat, self.long, self.alt)  # lat.degrees, lon.degrees,
        elevation_m = alt
```

Fig 6.21 RVZ class

```
59
60          time_list = pd.date_range(start=self.startTime, end=self.endTime, freq=f"{self.step}s")
61
62          # loop through every date in time_list
63          for date in time_list:
64              t = ts.utc(date.year, date.month, date.day, date.hour, date.minute,
65                         date.second + (date.microsecond / 1000000))
66
67              difference = satellite - bluffton
68              topocentric = difference.at(t)
69
70              alt, az, distance = topocentric.altaz()
71
72              li = [date, az.degrees, alt.degrees, distance.km]  # list of date, Azimuth ,
                elevation ,range
73              allValues.append(li)
74
75          return allValues  # return all values
```

Fig 6.22 RVZ class

In these photos show RVZ class:

- Class RVZ (radio visibility zone) is used to get azimuth, elevation, distance from TLE by setting ground station position (latitude, longitude and elevation), parameters: line1, line2, startTime endTime, stap, satelliteName.

6.1.9.4 Ground track:

```python
from skyfield.api import load, wgs84
from skyfield.api import EarthSatellite

import numpy as np
#from tools import Tools

class GroundTracking:
    """
        class GroundTracking is used to propagate (latitude, longitude) orbiting satellite from
        two-line-element (TLE)
        set for certain amount of time
        Parameters:
        ------------
        line1: string (69char)
        first line of the two-line-element (TLE) set (expected to be exactly 69 characters long)

        line2: string (69char)
        second line of the two-line-element (TLE) set (expected to be exactly 69 characters
        long)

```

Fig 6.23 ground track class

```
        satellite name: string
        satellite name is in TLE set in three lines

        Examples
        --------
        >>  # time list for points of satellite orbit
            time_list = np.array(
                [[2022, 2, 12, 2, 9, 10], [2022, 2, 12, 2, 9, 11]])
            # satellite TLE set
            satellite_name = 'Egyptsat_1_31117'
            s = '1 31117U 07012A   22042.15272208  .00000616  00000-0  10292-3 0  9990'
            t = '2 31117  97.8526   1.4573 0005868 157.5831 202.5635 14.73231277796180'

            # create satellite object
            test = GroundTracking(s, t, satellite_name)

            # propagate orbit
            latitude, longitude = test.Tracker(time_list)
            for x, y in zip(latitude, longitude):
                print('latitude: {},longitude: {}'.format(x, y))

```

Fig 6.24 ground track class

```
39
40          latitude: -71.84675666943076,longitude: -147.3113676089632
41          latitude: -71.79197356693906,longitude: -147.4004939247487
42      """
43
44      def __init__(self, line1: str, line2: str, satellite_name: str = "satellite"):
45          self.line1 = line1
46          self.line2 = line2
47          self.satellite_name = satellite_name
48          self.ts = load.timescale()
49          self.satellite = EarthSatellite(self.line1, self.line2, self.satellite_name, self.ts)
50
51      def lonlat_at(self, date):
52          """
53              calculate latitude, longitude of an orbiting satellite
54              :parameter date: (list_like)
55              the time in which position and velocity are calculated
56              :returns
57              latitude and longitude
58          """
```

Fig 6.24 ground track class

```
59
60          # get the geocentric information of the satellite in exact moment
61          geocentric = self.satellite.at(self.ts.utc(date.year, date.month, date.day, date.hour,
            date.minute, date.second))
62
63          # calculate latitude, longitude at date moment
64          lat, long = wgs84.latlon_of(geocentric)
65
66          return lat.degrees, long.degrees
67
68      def Tracker(self, timelist):
69          datetime_lat_long = []
70          """
71          propagate satellite ground track for certain amount of time
72          :parameter timelist: (list_like)
73          list of dates position and velocity are calculated in  to form an orbit
74          :return:
75          satellite latitude and longitude (numpy array)
76          """
77          #latitude_list, longitude_list = [[], []]
78
```

Fig 6.25 ground track class

```
79        # find latitude, longitude for each date
80        for date in timelist:
81            step_latitude, step_longitude = self.lonlat_at(date)
82
83            li = np.array([date, step_latitude, step_longitude])
84            datetime_lat_long.append(li)
85
86        return datetime_lat_long
87
```

Fig 6.26 ground track class

In these photos show ground track class:

- Ground track class is used to propagate (latitude, longitude) orbiting satellite from TLE, parameters: line1, line2, satellite name.
- Lonlat-at function calculates latitude, longitude of an orbiting satellite, Parameter: date and return latitude and longitude.
- Tracker function propagates satellite ground track for certain amount of time, parameter: timelist, and returns satellite latitude and longitude

6.2 Version 1.1

Some modifications have been added at the request of the Egyptian Space Agency:

- The server information page has been added, and it opens only once when the program is running for the first time, and this is for the purpose of the user entering the name of the server he has and the name of the database, if it is called something other than ODM, and this achieves the ease of dealing with any device and after entering the name, it is stored and the user does not need enter it again.

Fig 6.27 Server information

- The API used has been changed so that it is not constantly updated to another, better one

- The TLE Auto update feature was working every time the user was running the program, but it was restricted that it only works once a day, so there was no inconvenience at the time of opening the program and it is being modified to run faster

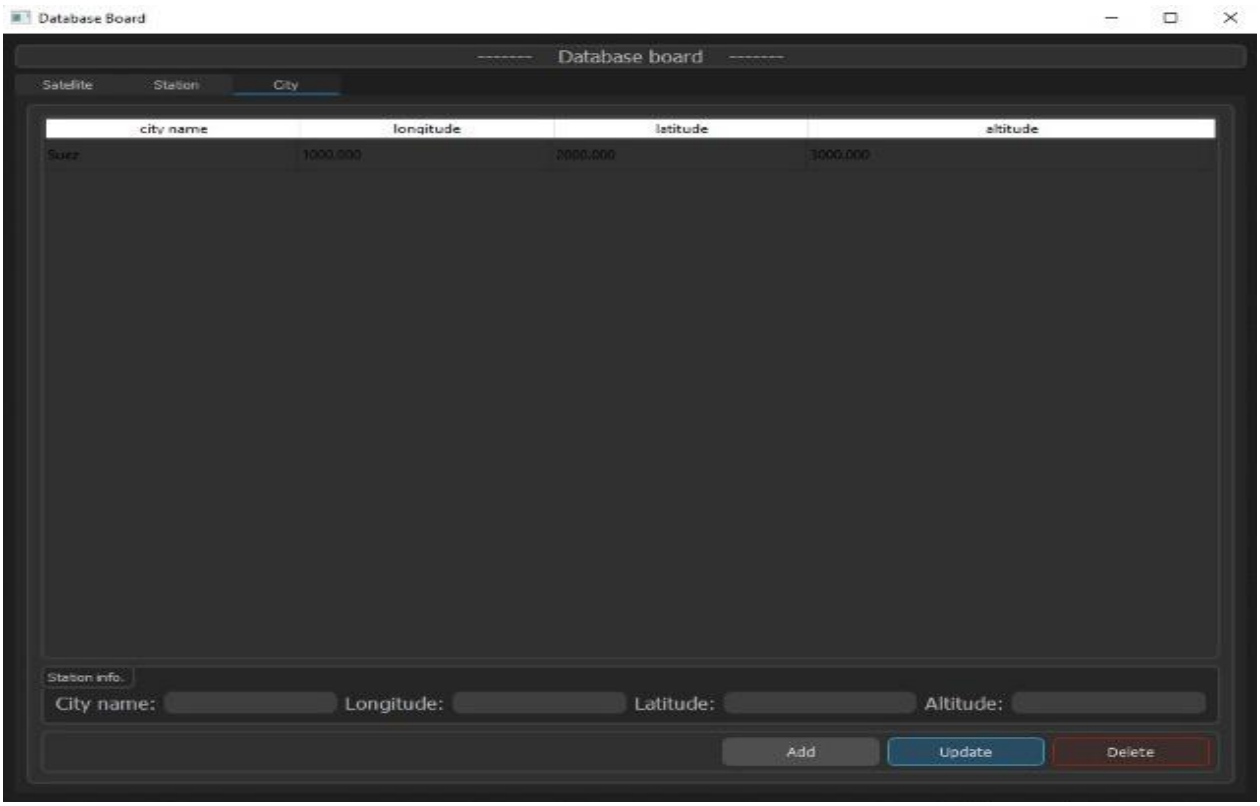


Fig 6.28 Database Board page" City"

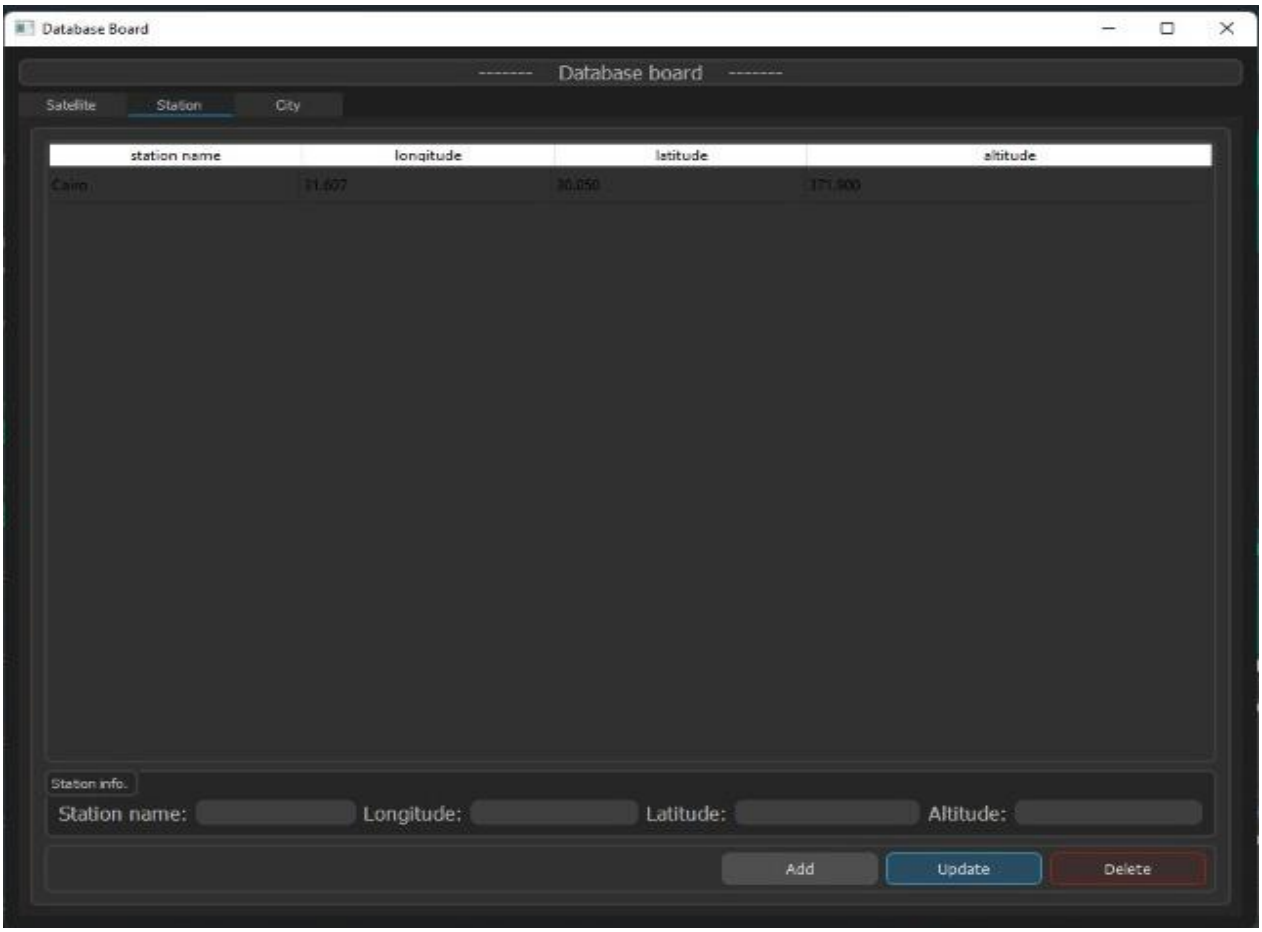


Fig 6.29 Database Board page" Station"

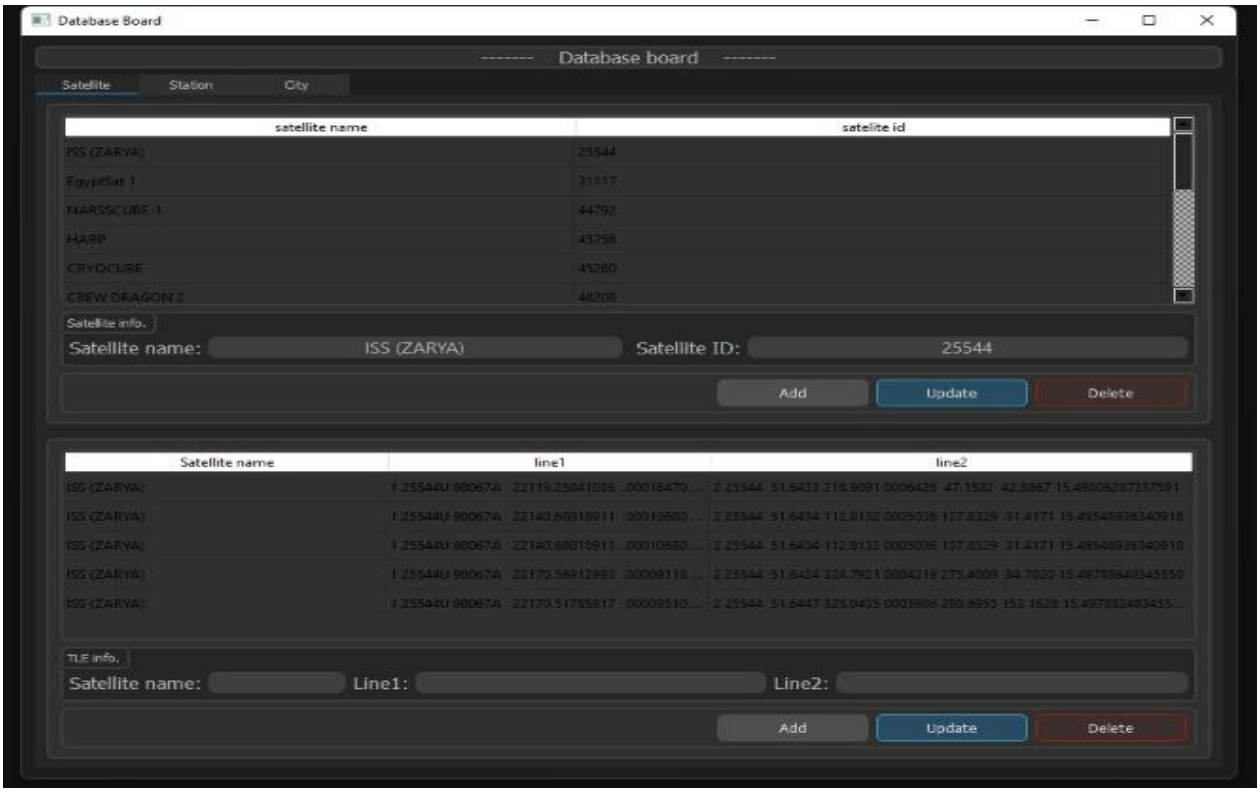


Fig 6.30 Database Board page" Satellite"

- The cancellation of the confirmation message was deleted after choosing the session because it was annoying at the speed

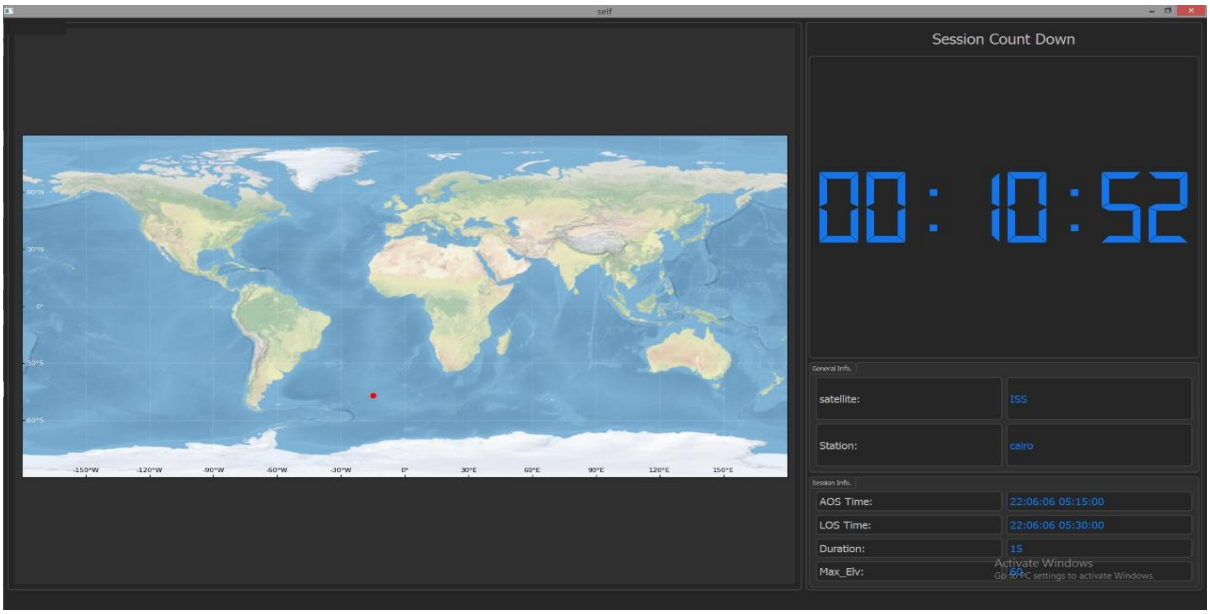


Fig 6.31 Session Count Down

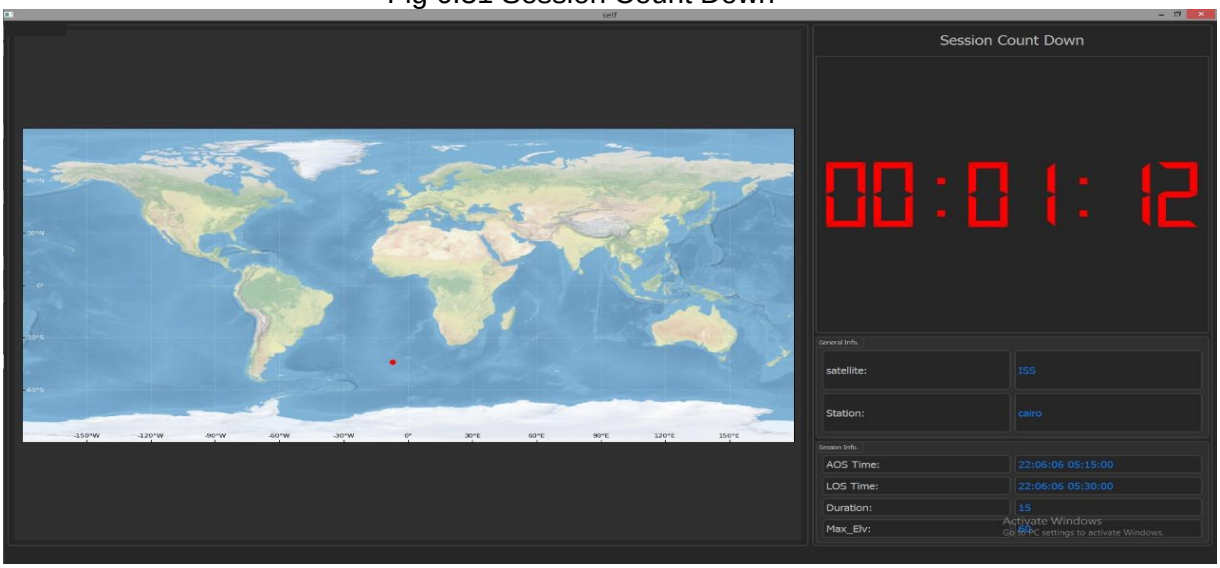


Fig 6.32 Session Count Down

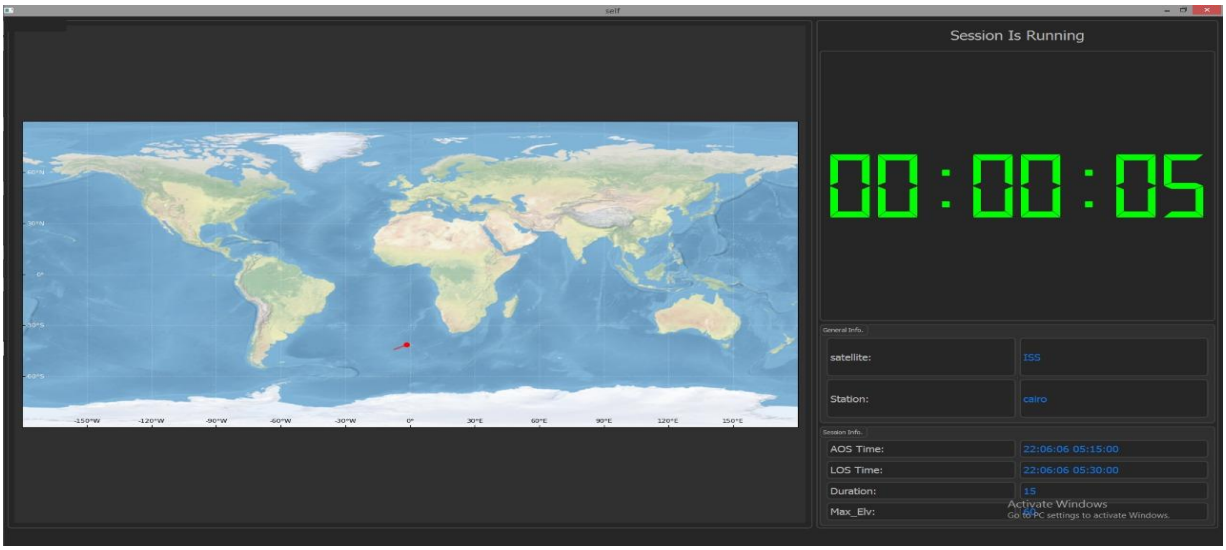
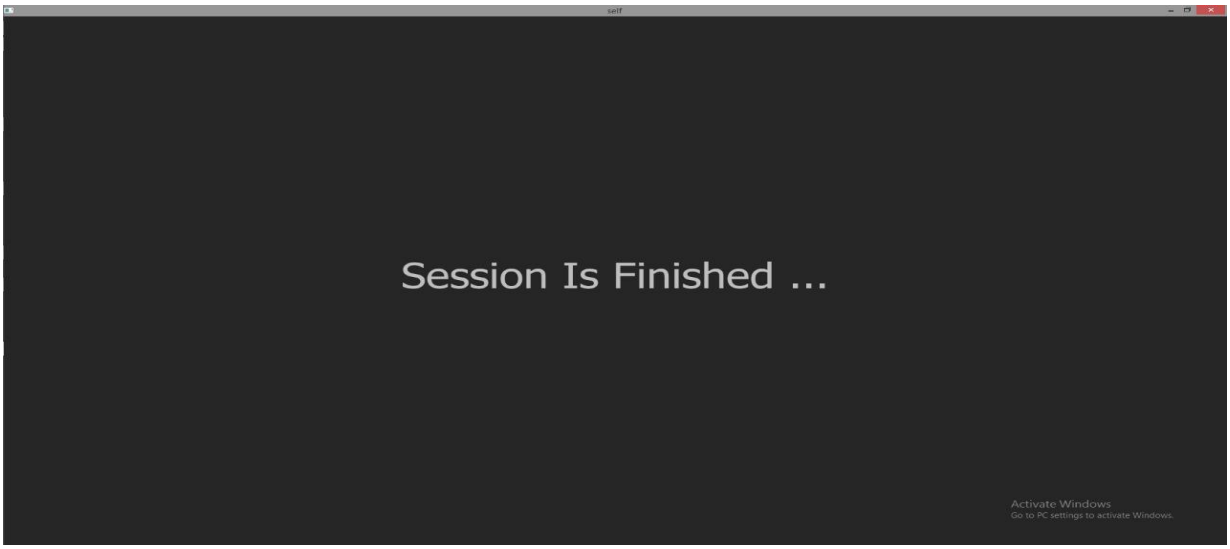
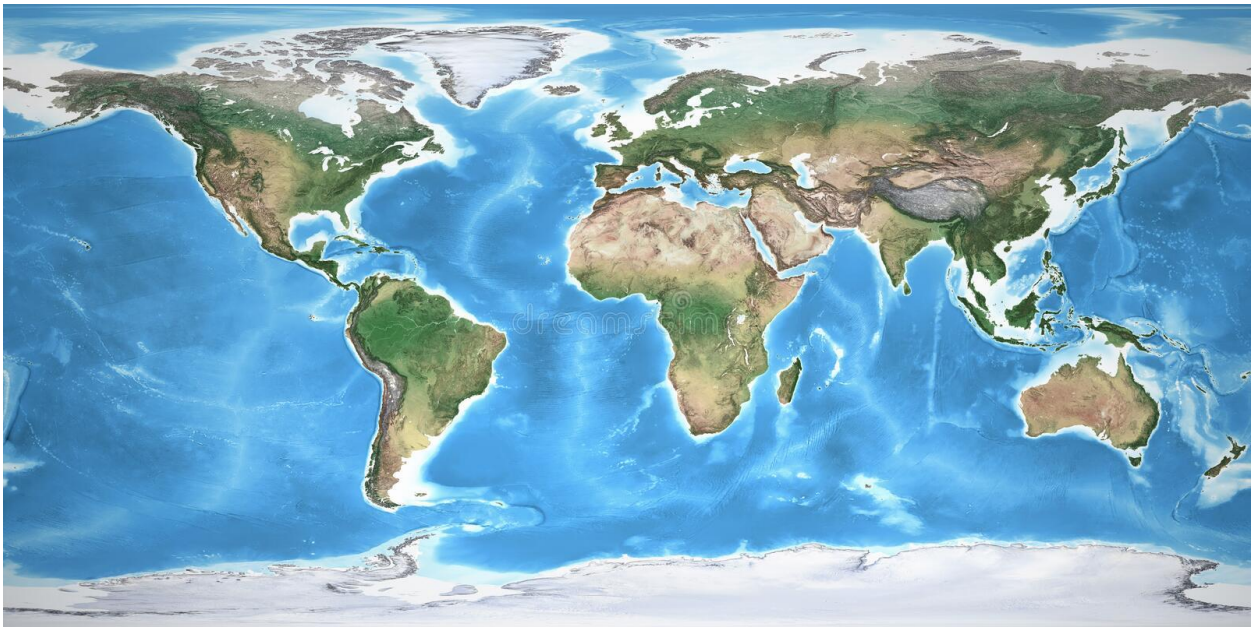


Fig 6.33 Session Run



Chapter 7 Conclusion



7.1 conclusion

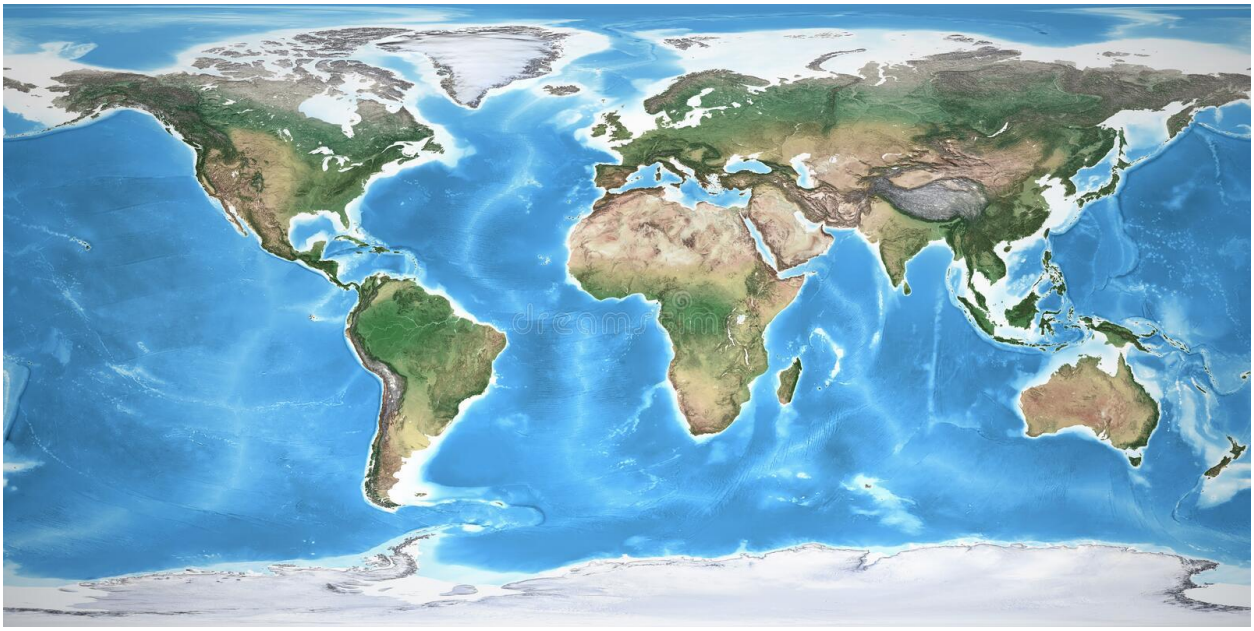
In this documentation display functions in program. Those functions are:

- satellite movement parameters forecasting,
- The satellite ground tracks are being calculated.
- Make a basic Propagator to track satellites.
- Drowning the ground track line on a 2D or 3D simulator to determine satellite motion visually
- Calculate the radio visibility zone for the ground station (Time of Communication sessions).
- Create a comprehensive database for all of the orbit parameters' futures, including all classes and schema.
- Create a user-friendly, familiar graphical user interface that allows the operator to quickly access all of the program's features.
- Calculate the ground station antenna's target designation (where the antenna will be aiming).
- Save the results in a database that is both local and shareable.

7.2 Future work:

- 3D visualization that helps the user to imagine ground tracking features.
- Track elimination in ground track
- Add more hand features to make usage of the program easier.

Chapter 8 References



References

1. https://pypi.org/project/skyfield/#:~:text=Skyfield%20is%20a%20pure%2DPython,for%20planets%20and%20Earth%20satellites
2. https://pypi.org/project/sgp4/

Reference Libaray:

3. https://pypi.org/project/skyfield/#:~:text=Skyfield%20is%20a%20pure%2DPython,for%20planets%20and%20Earth%20satellites
4. https://pypi.org/project/sgp4/
5. https://opensource.gsfc.nasa.gov/projects/ODTBX/
6. https://www.ion.org/publications/abstract.cfm?articleID=15892

Diagrams:

7. https://tallyfy.com/uml-diagram/

Books and Papers:

8. Bem D, Wieckowski T and Zielinski R 2000 Broadband Satellite Systems IEEE Comm. Surveys and Tutorials 3

9. Giambene and Kota S 2008 Special issue on satellite networks for mobile service Space Communications Journal
10. Data of TLE NORAD NORAD Two-Line Element Sets Current Data. Today from The Center for Space Standards & Innovation

11. Data of TLE NORAD NORAD Two-Line Element Sets Current Data. Today from The Center for Space Standards & Innovation

12. Barts R and Stutzman W 1992 Modeling and Simulation of Mobile Satellite Propagation IEEE Transactions on Antenna and Propagation 40 375-82

13. Suh S-Y and Stutzman W A 1998 Land Mobile Satellite Communications Propagation Simulator Space Communications 15 33-53

14. Kostov N 2003 Mobile Radio Channels Modeling in MATLAB Radio Engineering 12

15. Brouwer, D., “Solution of the Problem of Artificial Satellite Theory without Drag”, Astronomical Journal 64, 378—397, November 1959.

16. Hilton, C.G. and Kuhlman, J.R., “Mathematical Models for the Space Defense Center”, PhyloCode Publication No. U-3871, 17—28, Nove

17. Kozai, Y., “The Motion of a Close Earth Satellite”, Astronomical Journal 64, 367—377, November 1959.

18. Lane, M.H. and Hoots, F.R., “General Perturbations Theories Derived from the 1965 Lane Drag Theory”, Project Space Track Report No. 2, December 1979, Aerospace Defense Command, Peterson AFB, C

19. Born, George H., and Kirkpatrick, James C. 1970. Application of Brouwer’s Artificial-Satellite Theory to Computation of the State Transition Matrix, NASA Technical Note NASA TN D-5934, August.

20. Flohrer, Tim, Holger Krag and Heiner Klinkrad. 2008. Assessment and Categorization of TLE Orbit Errors for the US SSN Catalogue. Paper presented at the AMOS Technical Conference. Maui, HI.

21. Levit, Creon, and William Marshall. 2011. Improved orbit predictions using two-line elements Advances in Space Research Journal. Vol 47: 1107. 1 Apr

22. Levit, Creon, and William Marshall. 2011. Improved orbit prediction's using two-line elements Advances in Space Research Journal. Vol 47: 1107. 1 Apr

23. Vallado, David A. 2012. SDC Orbit Determination Verifications. Presentation at the SDA Users Meeting, March 12, 2012

24. Thales Alenia Space; “Innovative GPS Receiver”; Final Presentation Day 2009/09/08, ESA Noordwijk (2009).

25.] Wermuth M., Montenaros O., van Helle Putte T.; “GPS High precision Orbit determination Software Tools (GHOST)”; 4th International Conference on Astrodynamics Tools and Techniques; 3-6 May 2010, Madrid (2010).

26. Wermuth M., Mastenbrook O., van Helleputte T.; “GPS High precision Orbit determination Software Tools (GHOST)”; 4th International Conference on Astrodynamics Tools and Techniques; 3-6 May 2010, Madrid (2010).