UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AERONÁUTICA Y DEL ESPACIO

GRADO EN INGENIERÍA AERONÁUTICA

TRABAJO FIN DE GRADO

Development and validation of an EventSat mission simulator for the quantification of event camera-based space-to-space object detections

AUTOR: Alberto SÁNCHEZ CALVO

ESPECIALIDAD: Ciencias y Tecnologías Aeroespaciales

TUTOR PROFESIONAL: Ramón María GARCÍA ALARCIA

TUTOR DEL TRABAJO: Sergio ÁVILA SÁNCHEZ

COTUTOR DEL TRABAJO: Jose Luis PÉREZ BENEDITO

Febrero de 2025

1 Abstract

In the context of an increasing number of space objects and the necessity of tracking these due to the higher and higher risk of collisions among them, various types of surveillance systems have been developed, most of them relying on ground-based technology. In spite of receiving enhancements that make them more precise, they still have some limitations and cannot detect space objects smaller than a ball of 10 centimeters in diameter. In order to surpass these limitations, some on-orbit tracking systems are under development, including the focus of this bachelor's thesis: the EventSat project.

The aim of this project is the development of a satellite simulator that incorporates an event sensor camera. As a developing technology, event cameras are still maturing, especially in fields such as space. Their greatest advantage is that these kinds of camera sensor can capture information at a high temporal resolution and at the same time produce small data packages: an unquestionably advantage for in-orbit space-to-space object detection CubeSats.

The high temporal resolution of event cameras is crucial for detecting fast-moving objects in space, where relative velocities between satellites and debris can exceed thousands of meters per second. Conventional frame-based sensors risk missing such objects if they cross the camera's field of view between consecutive frames. Event cameras avoid this limitation by near-continuous detection. Simultaneously, their low data output reduces computational and power demands, a critical advantage for CubeSats, which operate under rigorous resource constraints.

Event camera simulators exist, but they are not typically applied for space purposes. The simulator that will be developed in this project queries an up-to-date database of objects orbiting the Earth. Then, the information is downloaded and a set of instructions performs the necessary calculations and adaptations needed to make the dataset optimal for use.

Once the data are stored, the user, through a graphical user interface, has the ability to add their custom satellites to the dataset. For this, the user can add the necessary parameters in a form or they can upload a JSON file containing the needed parameters to model an event camera satellite. The latter will be especially interesting when analyzing the optimal orbit: a set of observant satellites can be added in bunches to obtain the one that gathers a greater number of encounters.

Then, the user can select all of the dataset or a set of custom satellites and objects from the dataset and add them to the simulation. The next step is to adjust the propagation when the user has to define the propagation method and the time window in which this calculation is going to be performed. Once propagation is performed, an algorithm computes which of the selected satellites is observable by the event sensors of the created satellites. This information is displayed in a user-friendly interface, where some clicks lead the user to run a simulation of what a regular sensor camera would see and what an event-based camera would.

Once the software is completed, it will be validated using commercial software that is broadly supported and that has already been tested, so computations can be ensured to be accurate enough. First, the propagation and the 3D visualizations are validated, and then the simulated event camera output is checked.

Finally, a use case is elaborated: several observant satellites are going to be created, and then the observations with surrounding objects will be calculated. By analyzing the mission objectives and constraints, the best orbit will be investigated.

The interested reader can find a copy of the project code in the following link: https://github.com/alberto-sanchezcal/satsimulator

Contents

List of Figures

List of Tables

Acronyms

ν True Anomaly. 14, 34, 50

Ω Right Ascension of the Ascending Node. 13, 34, 50

ω Argument of Periapsis. 13, 34, 50

18th SDS 18th Space Defense Squadron. 12

19th SDS 19th Space Defense Squadron. 12

a Semi-major Axis. 13, 34, 50

CFSCC Combined Force Space Component Command. 12

COE Classical Orbital Elements. 7, 49–51

COSPAR Committee on Space Research. 13

E Eccentric Anomaly. 14

e Eccentricity. 13, 34, 50

ECEF Earth-centered Earth-fixed. 3, 10, 38, 48, 50

ECI Earth Inertial Frame. 3, 10

FOV Field of View. 9, 26, 27, 31

GEO Geostationary Orbit. 4

GMAT General Mission Analysis Tool. 32, 49

GPS Global Positioning System. 1

GSSAP Geosynchronous Space Situational Awareness Program. 5

GUI Graphical User Interface. 40–42

i Inclination. 13, 50

LEO Low Earth Orbit. 1, 3, 4, 6, 32, 54

LVLH Local Vertical Local Horizontal. 10

M Mean Anomaly. 14

NEO Near Earth Object. 5

NSSDCA NASA Space Science Data Coordinated Archive. 13

OMM Orbit Mean Message. 13

RAE Royal Aircraft Establishment. 15

SATCAT Satellite Catalog. 12, 15

SGP Simplified General Perturbation. 3, 24

SSA Space Situational Awareness. 5

SST Space Surveillance and Tracking. 5

STK Systems Tool Kit. 32

SWE Space Weather. 5

TLE Two-Line Elements. 5, 13, 15

US United States. 12, 13

USSPACECOM United States Space Command. 12

2 Introduction

Today, it is impossible to think about a world without satellites. These technological pillars form the modern infrastructure that makes people's lives easier and safer: communication satellites enable global connectivity, Global Positioning System (GPS) networks guide navigation, and meteorological satellites safeguard lives through weather forecasting. The large amount and growing number of applications these satellites have are unimaginable.

To understand the origins of this satellite-dependent era, one must revisit the geopolitical situation of the mid-20th century. In the 1960s, the United States and the Soviet Union were locked in a tense geopolitical situation known as the Cold War. It was characterized by intense competition for global influence, an arms race that included the development of nuclear weapons, and a fervent competition for technological supremacy. Central to this contest was the space race, which began with the Soviet Union's launch of Sputnik-1 in October 1957. Its purpose was to study the higher atmosphere layers and the Earth's electromagnetic field. It was followed one month later by Sputnik-2, which was the first mission to send an animal -a dog named Laika- to space. United States rushed and sent one year later its first satellite to space, Explorer, in 1958. This machine not only marked US's entry into space, but also led to the discovery of the Van Allen radiation belts, underscoring the scientific potential of satellites. What followed was a decade of milestones: the Soviet Union sent the first human, Yuri Gagarin, into space in 1961, while the U.S. countered with the Apollo program, landing astronauts on the Moon in 1969. Ironically, after some demonstrations of technological power by the two superpowers, the rivalry led to collaboration between the two, and the space race culminated in 1975 with the space project Apollo-Soyuz.

In the decades since then, satellite technology has evolved from a tool of superpower competition to a cornerstone of global civilian infrastructure. During all this time, the number of satellites has not stopped growing. However, in recent years, the number of satellites has increased exponentially [2]. Apart from the first uses of these space objects: research, military, Earth and astronomic observation, other commercial applications have arisen: communications, weather forecasting, navigation, and even real-time Earth imaging for urban planning. In particular, the rise of megaconstellations, vast networks of Low Earth Orbit (LEO) satellites such as SpaceX Starlink and OneWeb that will eventually provide global broadband access, also raises concerns about light pollution and orbital congestion. Astronomers warn that the planned launch of more than 100,000 internet satellites in the coming years [4] could obstruct celestial observations and irreversibly alter our view of the night sky.

To contextualize this growth, in about 60 years of space activities, more than 19,000 satellites have been launched. This figure has increased especially rapidly during the last 10 years, as can be observed in Figure 2.1, thanks to the miniaturization of space systems and investments in internet constellations [2].

Now, another thread looms: space debris. Despite the 19,000 satellites launched, more than 60,000 objects have been tracked since then, of which half remain orbiting in space and tracked [5] (see Figure 2.2).

The majority of these are space debris that is not controllable and pose a threat to space safety. Traveling at speeds exceeding 28,000 km/h, even a 1-cm fragment can render a satellite or part of it unusable (see Figure 2.3).

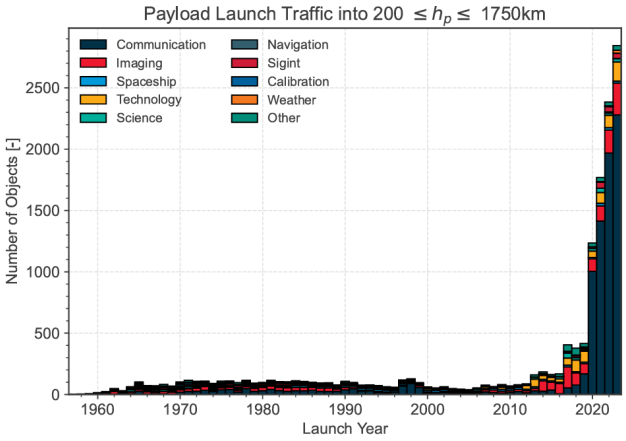


Figure 2.1 Evolution of payload type in LEO, [2]

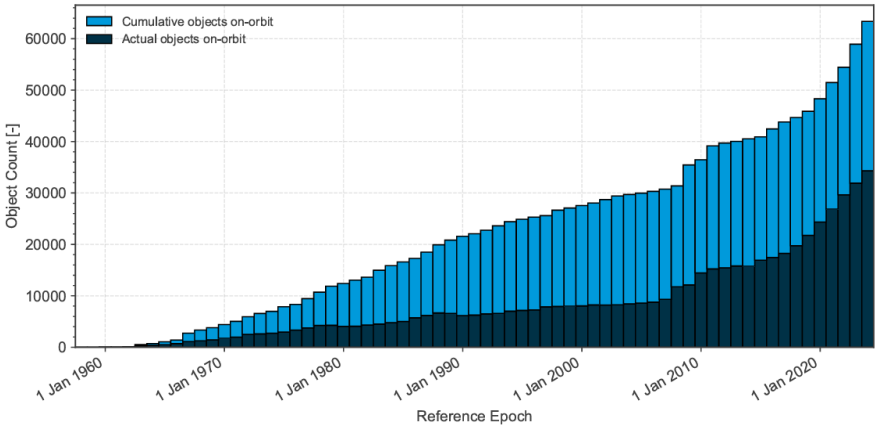


Figure 2.2 Evolution of cumulative and actual number of tracked objects, [2].

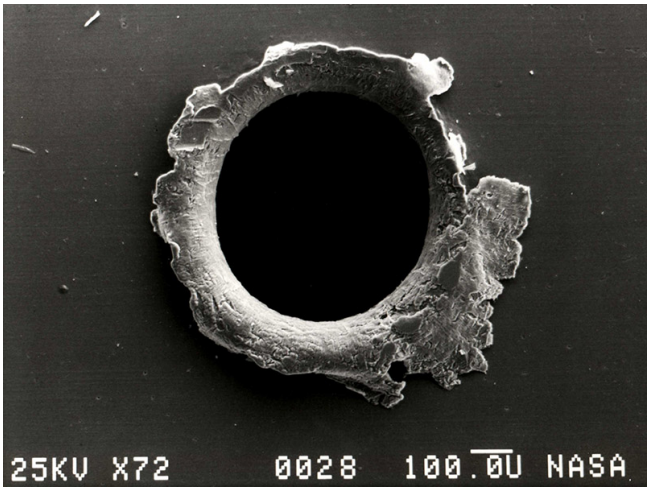


Figure 2.3 Impact of tiny space debris on NASA's Solar Max experiment. Credit: NASA

2.1 Space Debris

From the moment humanity first put an object in space, there have been more space debris than operational satellites. If one looks at Figure 2.4, it can be observed that Payload (PL in the legend) represents a small fraction of the total count of objects.

Space debris are human-made objects that result from space missions that no longer have any functionality once the mission is finished: rocket upper stages, inoperative old satellites. Furthermore, in this group, we can include the products that result from fragmentations (collisions, explosive break-ups, and tear and wear) [6].

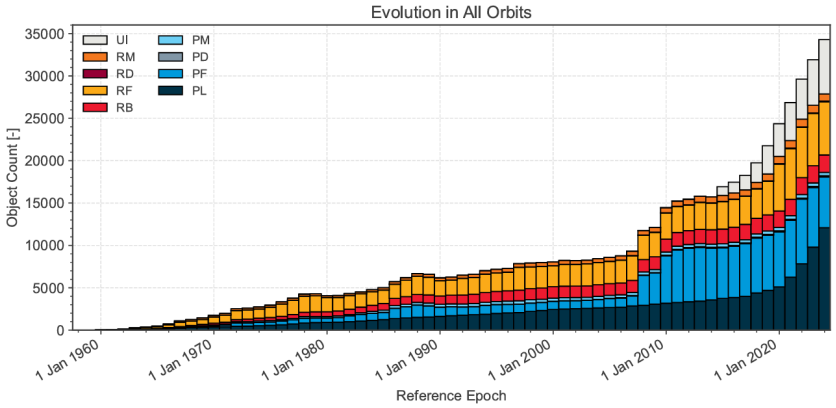


Figure 2.4 Evolution of tracked objects, [2].

As we can see in Figure 2.4, currently there are approximately 35,000 objects tracked. Out of these 35,000 objects, around 10,000 are operative satellites (PL), the rest are rests of their respective missions: payload debris, fragmentation debris and mission-related objects (PD, PF, PM), rocket body, debris, fragmentation debris and mission-related objects (RB, RF, RD, RM), or unidentified objects (UI).

Observing the distribution of these 35,000 objects, 20,000 occupy orbits in Low Earth Orbit (LEO) (altitudes lower than 2,000 km) [2]. If we now look at the distribution of the space debris, we can observe that the highest density is found at altitudes between 800 and 900 kilometers. At lower orbits the density is lower because of atmospheric drag that acts as cleaner and helps to de-orbit space debris [6]. In this way, the highest density is located in orbits around 900 km of altitude and 98º of inclination (see Figure 2.5).

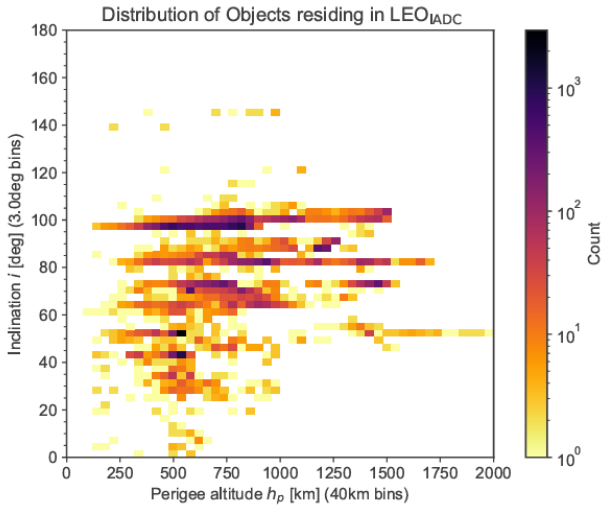


Figure 2.5 Distribution of objects in LEO, [2].

All cataloged objects have a diameter greater than 10 centimeters. However, statistical models have quantified in more than a million the number of objects orbiting Earth with diameter greater than a centimeter and 100 million for objects bigger than a millimeter in diameter as seen in Figure 2.6 [5].

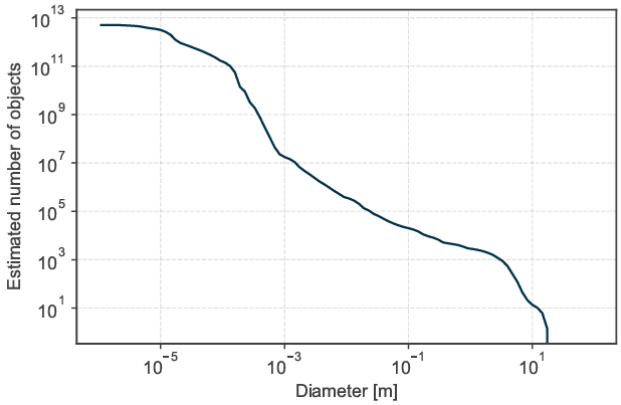


Figure 2.6 Estimated number of space debris objects as function of the object size in Earth orbit. [2]

These smaller objects result from self-ignition of mechanical satellite parts that carry fuel or solid fuel that transforms into particles when fired in space.

Some other sources of space debris are in-orbit collisions. The first-ever accidental collision happened between an American and a Russian satellite. They were destroyed and spread over 2,000 trackable objects [7]. This was known as the second most catastrophic collision. The first one occurred two years earlier when China decided to conduct an antisatellite test, adding more than 3,500 objects to the space debris count [8].

Due to the large number of objects surrounding our planet, one can tend to think that collisions are the main source of space debris, but this is not like that, yet. Throughout the history of the space industry, only 650 in-orbit fragmentations have been detected, of which 7 correspond to in-orbit collision between space operational satellites and space debris [9]. However, while the number of fragmentations is constant over time (10 fragmentations/year), their impact is not constant: their size, mass, and velocity (about 10 km/s) determine the degree of catastrophic effects their impact can cause [5].

Although not a problematic source of space debris, in the future on-orbit collisions are estimated to become the main source [10]. In the next centuries, a cascade of collisions will occur if no further application of space debris mitigation guidelines is applied; see Figure 2.7.

Avoiding on-orbit collisions implies orbital maneuvers (performed when the probability of impact exceeds 0.0001%). These require a propellant, which is a finite source of energy in space.

Most countries that launch satellites have agreed to follow some guidelines to mitigate space debris: limit space debris release during normal operations, prevent collisions and potential collisions in orbit, or remove objects from protected regions Low Earth Orbit (LEO) and Geostationary Orbit (GEO) once their lifetime has ended [2]. In the last five years, over 50% of the Low Earth Orbit (LEO) payloads reaching the end of their useful life successfully attempted to deorbit, this number rounded 90% for Geostationary Orbit (GEO) payloads and rocket bodies in Low Earth Orbit (LEO). However, these efforts are not sufficient and the goal is to get a 100% if space sustainability is the goal.

2.2 Current Space Surveillance and Tracking Systems

Despite the agreement of the space users of following the mitigation guidelines, the space debris that is already there is going to be orbiting space and posing a threat to the active payloads. These objects

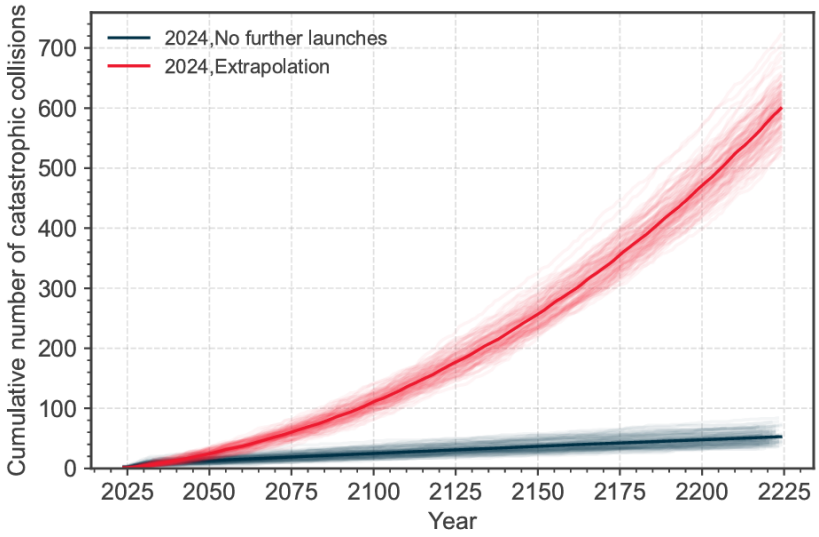


Figure 2.7 Collision trend, [2].

have to be controlled and tracked continuously to calculate the risks of impact and, therefore, take action through avoidance maneuvers.

Space Surveillance and Tracking (SST) is part of Space Situational Awareness (SSA) [11], which also includes Space Weather (SWE) Segment and Near Earth Object (NEO) Segment. It comprises the necessary actions to track space objects in space. The methods range from ground-based survey radar, tracking radar, ground-based optical telescopes, ground-based laser tracking, or space-based optical telescope [11]. These methods can be integrated into single-sensor or multisensor platforms.

This system also is complemented with a data center to process the acquired data and deliver an updated catalog. This is done following some steps: correlation (check if the objects have been already detected), orbit determination (update or add the orbital parameters), monitor the catalog data [12].

In terms of sensors, in line with the tracking methods commented on before, we can differentiate three different types of sensors: optical (passive and active), laser, and radar. Optical sensors are inexpensive and effective, but are limited by weather conditions. Radar systems present high accuracy when determining orbital parameters, with errors of around 10 meters, and lack of limitation in terms of weather. These errors could be lowered to 5 meters due to the use of laser sensors. Today, multi-laser sensor systems, which combine different types of sensor to enhance their capabilities, can collect orbit information with a precision of 2 centimeters [13].

Satellite tracking has been researched, with constellations of satellites such as Geosynchronous Space Situational Awareness Program (GSSAP) [14], Silentbarker [15], the Vyoma Flamingo [16] or the NorthStar Skylark satellites [17]. All these satellites place an optical sensor to carry out their observations. The benefits of placing an optical sensor in a satellite in space are robustness since this will not be limited to weather, atmospheric conditions, and atmospheric effects on light (thus, systems can be built to be diffraction-limited) [18]. The main goal of these technologies is not cataloging new objects in space, but providing further information about the smallest objects to improve the space debris models.

In this context, it is crucial to highlight that the development of Space-to-Space Tracking Systems using event-based sensors is an area of active research.

Satellite	Resolution	Mass [kg]	Volume [cm]	Power [W]
EventSat	15 cm at 500 km	9	6U	7-19
Skylark	5 cm in LEO; 40 cm in GEO	n.a.	16U	n.a.
Flamingo 1	10 cm in LEO	70-150	50x50x65	60-200
Flamingo 2	10 cm in LEO	150-250	50x50x50	50-300
Silentbarker	Classified	Classified	Classified	Classified

Table 2.1 Comparison between EventSat and other SSA satellites [3]

2.3 EventSat Mission

The EventSat mission represents a pioneering effort in nanosatellite technology, employing a compact 6U CubeSat platform in Low Earth Orbit (LEO) to validate the capabilities of event-based vision systems. EventSat aims to improve the precision and efficiency of space-to-space surveillance, taking a significant step forward in orbital surveillance and collision avoidance systems [3].

Although most of the in-space SSA missions mount optical cameras, leading to higher power consumption, EventSat is able to operate at lower power consumption, around 7-19 Watts, thanks to the use of event-based sensors. See Table 2.1

This project contributes to this ongoing effort by developing a sophisticated simulator.

3 Methodology

In this section, we explain the relevant theoretical concepts used in this work. These include: the functioning of an event-based sensor, the coordinate systems employed, the database structure used, and the different orbital propagators considered.

3.1 Event Sensor

An event, neuromorphic, or silicon sensor is a type of imaging sensor that captures changes in brightness in each of its pixels in an independent and asynchronous way from the others. When this change is greater than a preset threshold, the pixel gets activated. Equation 3.1 describes the fundamental principle of how an event sensor decides to generate an "event" at a specific pixel.

$$\log I(\mathbf{x}, t) - \log I(\mathbf{x}, t - \Delta t) = \pm C \tag{3.1}$$

In the Equation 3.1 $I(\mathbf{x}, t)$ is the intensity at pixel **x** at time t, where **x** is the spatial location of the pixel on the sensor (e.g., (row, column)) and t is the current time. The term $I(\mathbf{x}, t - \Delta t)$ is the intensity at pixel **x** at time $t - \Delta t$, where $t - \Delta t$ represents a previous time, a short time interval Δt before the current time t. The $\log$ function (typically natural logarithm, ln, but could be base 10 log as well) is applied to both intensity values. This subtraction represents a measure of the relative change (because $\log(a) - \log(b) = \log(a/b)$) and thanks to it we can manage high dynamic range images.

On-events occur when the logarithmic intensity change is positive and equal to the C threshold, off-events [19], when this is negative, as seen in Figure 3.1. An event consists of a timestamp, the position of the pixel, and the polarity.

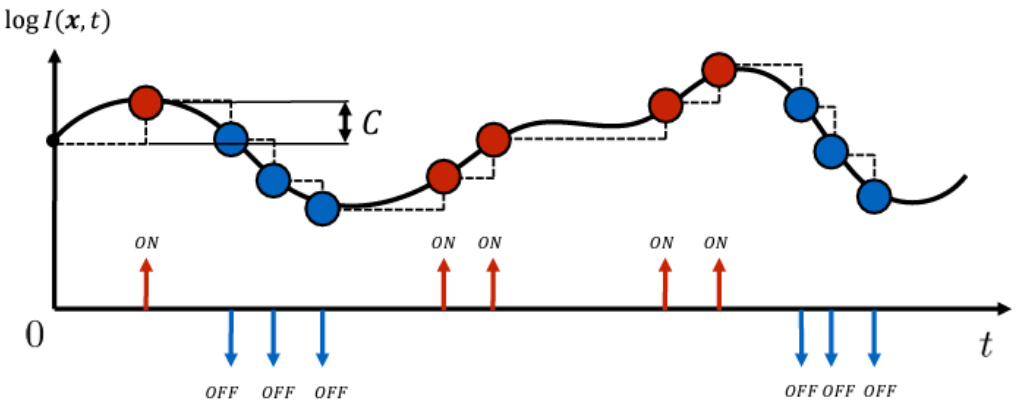


Figure 3.1 Event Trigger. Credits: Davide Scaramuzza

Due to their asynchronous nature, event sensors can capture events with high temporal resolution (around 1 microsecond) [20], effectively reducing motion blur. Furthermore, because of this operational mode, the data transfer rate is not constant, allowing energy and bandwidth consumption to vary. This typically results in lower power consumption. Event cameras also offer a higher dynamic range compared to standard cameras, making them more suitable for high-contrast scenes [21].

These are the features that make this type of cameras stand out among the conventional (frame) cameras that capture absolute brightness values. This is done in a synchronous way: the camera records the entire pixel array at a determined frame rate, even if nothing is happening in front of the sensor, and retrieves data for every of their pixels. In the majority of the cases it results in redundant data, unlike event sensors which are able to store data only when events occur.

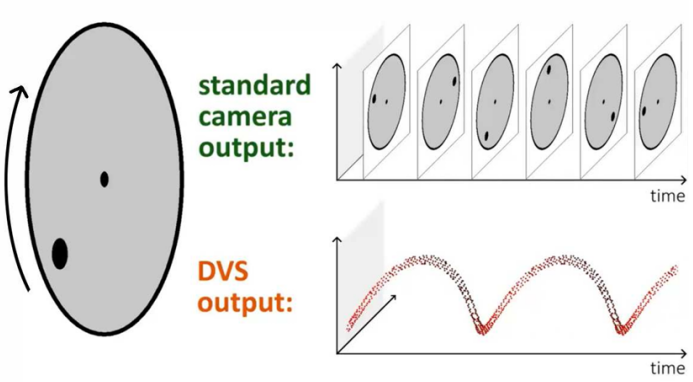


Figure 3.2 Traditional vs Event-based camera sensors. Credits: Davide Scaramuzza

Table 3.1 provides a comparison to further illustrate the differences between high-speed cameras, standard cameras, and event-based cameras in several important performance metrics. It clearly illustrates the trade-offs and advantages of each type. [19].

High-Speed cameras prioritize temporal resolution at the cost of spatial resolution and generate a large amount of data, whereas standard cameras offer a balance of speed and resolution with moderate data rates and power consumption. Event-based cameras, however, stand out by combining high temporal and spatial resolution with remarkably low data rates and power consumption, along with an exceptional dynamic range. These features make them efficient and well-suited for applications that require speed, efficiency, and robustness in conditions where power and bandwidth are constrained, such as in space.

Table 3.1 Camera comparison. Credits: ETH Zürich

	High Speed Camera	Standard Camera	Event-based Camera
Max fps or measurementrate	Up to 1MHz	100-1,000 fps	1MHz
Resolution at max fps	64x16 pixels	>1Mpxl	>1Mpxl
Bits per pixels (event)	12 bits	8-10 per pixel	40 bits/event (t,(x,y),p)
Data rate	1.5 GB/s	32MB/s	1MB/s on average
Meanpower consumption	150 W	1 W	1 mW
Dynamic range		60 dB	140 dB

Despite their advantages, event-based sensors suffer from background noise, even in static scenes. Furthermore, they require specialized hardware for event-by-event processing, and common processors based on Von Neumann architectures are inefficient for this type of data [19]. This type of computers are based on sequential processing, separated computations and memory, code as binary instructions, and synchronicity (clock-driven). All these features lead to bottlenecks when processing event-based data. For solving this problem, the event data is well processed with neuromorphic computers based on spike neural networks [22]. These computers are inspired by brains and are composed of neurons and synapses, which handle both processing and memory, allow highly parallel operations and asynchronicity (event-driven, they only process information when events or spikes arrive).

Event cameras are well-suited for applications where real-time processing is critical, such as robotics and wearable electronics. They are also used for object tracking, surveillance, monitoring, and object/gesture recognition [23].

The benefits of using event-based sensors in space optical instruments include capturing information only when brightness changes within the Field of View (FOV), for example, upon the detection of a satellite or space object. This selective data acquisition reduces power consumption when there is no activity in the Field of View (FOV) and lowers the data rate. This advantage is crucial for space-based optical tracking systems, especially for small satellites where electrical power is a limited resource [24].

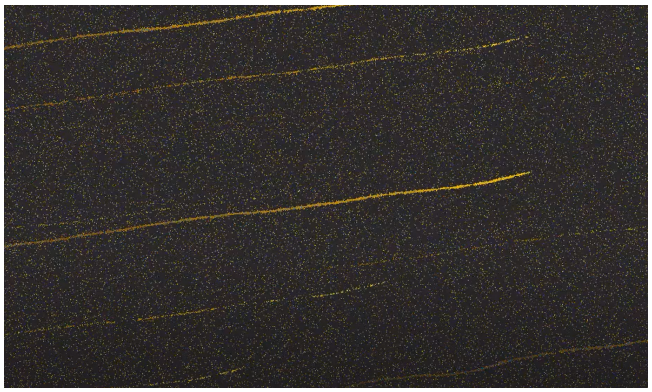


Figure 3.3 Event-based sensor application to SST. Credits: Western Sydney University

Indeed, the application of event-based sensor in space is not new. The Western Sidney University has already implemented this kind of sensor in on-ground mobile telescopes. Traditional space object tracking faced challenges such as huge amounts of discarded data or the ability to make observations only during the night. This new development allows for reduced data capture and daytime sky observation. Consequently, it has enabled direct orbital trajectory prediction: they only gather information when something is happening in the scene, so there is no need for making an active selection of all the space recording and they can perform observations during the day because event sensors work independently of brightness background [25].

3.1.1 IMX636 Event sensor

The IMX636 Sony sensor [26] is the event-based sensor that will work with EventSat. The characteristics are described in Table 3.2.

Table 3.2 Sony IMX636 sensor properties

Pixel array	1280 x 720
Camera resolution (MPixels)	0.9216
Sensor size (mm)	6.2208 x 3.4992
Contrast Threshold (%)	25
Dynamic range (dB)	100, depending on illuminance conditions
Maximum clock frequency (MHz)	100
Power (mW)	5 (standby) - 150 (maximum)

3.2 Reference Coordinate Frames

3.2.1 Earth Inertial Frame (ECI)

The Earth Inertial Frame (ECI) frame is a Cartesian spatial reference system where positions are represented by X, Y, and Z coordinates. See Figure 3.4. The origin of this system is the center of mass of the Earth. Z-axis is the Earth's rotation axis. The X-axis and the Y-axis are in the mean Equator plane (containing the Geocentre) of the epoch J2000. Specifically, the X-axis points towards the mean Vernal Equinox of epoch J2000 (12:00 Terrestrial Time on 1 January 2000). Y-axis forms a right-handed triad with the X-axis and Z-axis. Notably, the axes do not rotate with Earth. This is the system that is used usually in space computation. The classical elements can be considered to refer to this frame with a slight loss in accuracy.

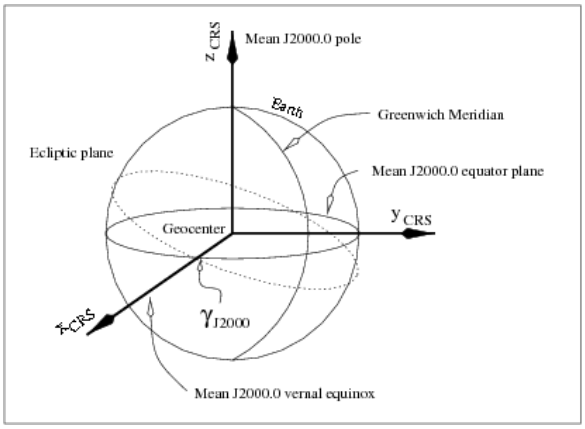


Figure 3.4 ECI Frame. Credits: Wikipedia

3.2.2 Earth-centered Earth-fixed (ECEF)

The Earth-centered Earth-fixed (ECEF) is known as the geocentric coordinate system. It is a Cartesian spatial reference system that represents positions as X, Y, Z coordinates. See Figure 3.5. Similarly to Earth Inertial Frame (ECI), its origin is at the Earth's center of mass. The Z-axis points towards the north pole, whereas the X-axis and Y-axis are in the equatorial plane. The X-axis points to the International Reference Meridian, and the Y-axis completes a right-handed triad with the X-axis and the Z-axis. All of the axes rotate with Earth. This is the system that is usually used to track the orbits of satellites and it is going to be used for the 3d representation of the space environment within the app.

It is important to note that the vector pointing to the North Pole does not perfectly align with the Earth's rotational axis. This discrepancy arises from Earth's rotational axis nutation, precession, and polar motion.

3.2.3 Orbit Frame

The orbit frame used is the Local Vertical Local Horizontal (LVLH) - Earth Pointing frame [27]. See Figure 3.6. The origin of this frame is located at the center of mass of the satellite. The Z-axis points towards the Earth's center, the Y-axis is aligned with the negative orbit normal, and the X-axis is defined by the cross product of the Y-axis and Z-axis, completing a right-handed coordinate system.

3.2.4 Body Frame

The body frame is specifically defined for each satellite application. In the case of an observing satellite, the origin of this frame is positioned in the center of mass of the satellite. X-axis has the direction of the

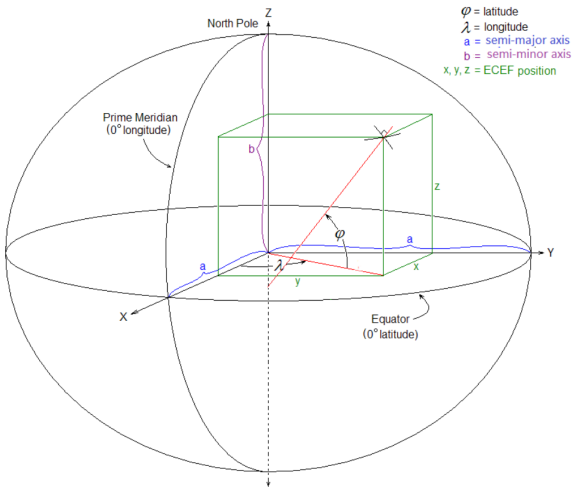


Figure 3.5 ECEF Frame. Credits: Wikipedia

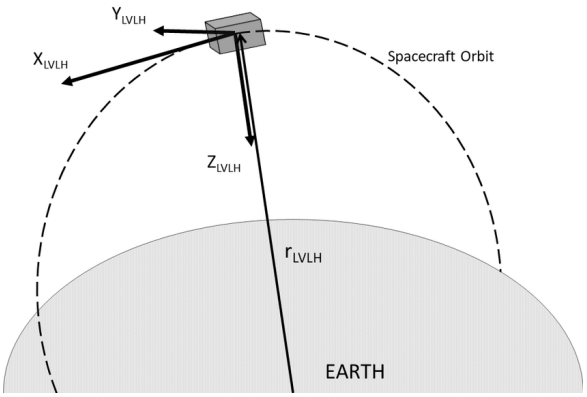


Figure 3.6 LHLV Frame, [**?**]

pointing camera vector, the Y-axis is the up-camera vector, and the Z-axis results from the cross product of the X-axis and the Y-axis. See Figure 3.7.

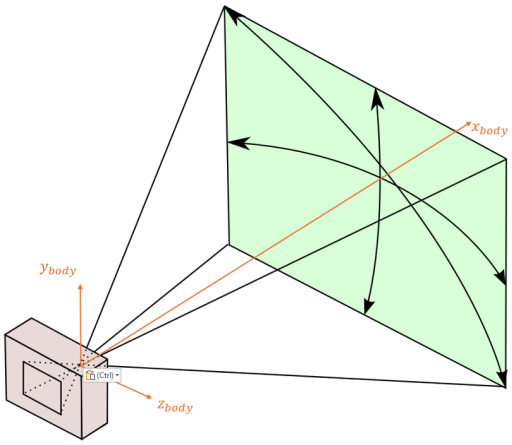


Figure 3.7 Body frame. Credits: Wikipedia

3.2.5 Quaternions

Quaternions are hypercomplex numbers that are used to define rotations between frames. In this context, they will be used to define the orientation of the body-fixed frame relative to the orbit frame. A quaternion consists of a vector representing the axis of rotation and a scalar representing the angle of rotation about that axis. The rotation follows the right-hand convention. The general representation of a quaternion is provided by Equation 3.2.

$$\mathbf{Q} = Q_1\mathbf{i} + Q_2\mathbf{j} + Q_3\mathbf{k} + Q_4 \tag{3.2}$$

It meets the condition: $Q_1^2 + Q_2^2 + Q_3^2 + Q_4^2 = 1$. This formulation is derived from Euler's theorem, which states that any rotation can be achieved from any orientation with a single rotation about one axis.

3.3 Data Base

3.3.1 SpaceTrack

SpaceTrack.org is an organization owned by the United States Space Command (USSPACECOM) and the Combined Force Space Component Command (CFSCC). It is managed by the 18th Space Defense Squadron (18th SDS) and the 19th Space Defense Squadron (19th SDS). Both perform space surveillance and maintain the space catalog. Their activities are classified into different levels of services: basic, emergency, and advanced. The basic service provides public access to a satellite catalog, element sets, reentry predictions, and decay confirmations through their website. Emergency services are tailored for owners and operators of active spacecraft, delivering conjunction data messages and maneuver information. The advanced service includes expanded capabilities, faster response times, and additional data and information.

The Satellite Catalog (SATCAT) they provide is available through a data base for United States (US) and international satellite owners / operators, academia and other entities. The data is updated a minimum of once every 24 hours for all near-Earth objects, all deep-space objects, and other tracked items. The interface of the wabpage is shown in Figure 3.8.

In the Satellite Catalog (SATCAT), space objects greater than 10 centimeters in diameter are listed if their origin is known and have been reliably tracked by the United States Space Command (USSPACECOM).

Each entry in the catalog contains the satellite number or NORAD ID, international designator, object type, and associated orbital parameters, among other characteristics. This information can be downloaded in two formats: Orbit Mean Message (OMM) format or Two-Line Elements (TLE), a compact format of the Orbit Mean Message (OMM) format. Advanced queries can be performed through the API or the Query Builder. For example, a data base that contains only active payloads could be downloaded.

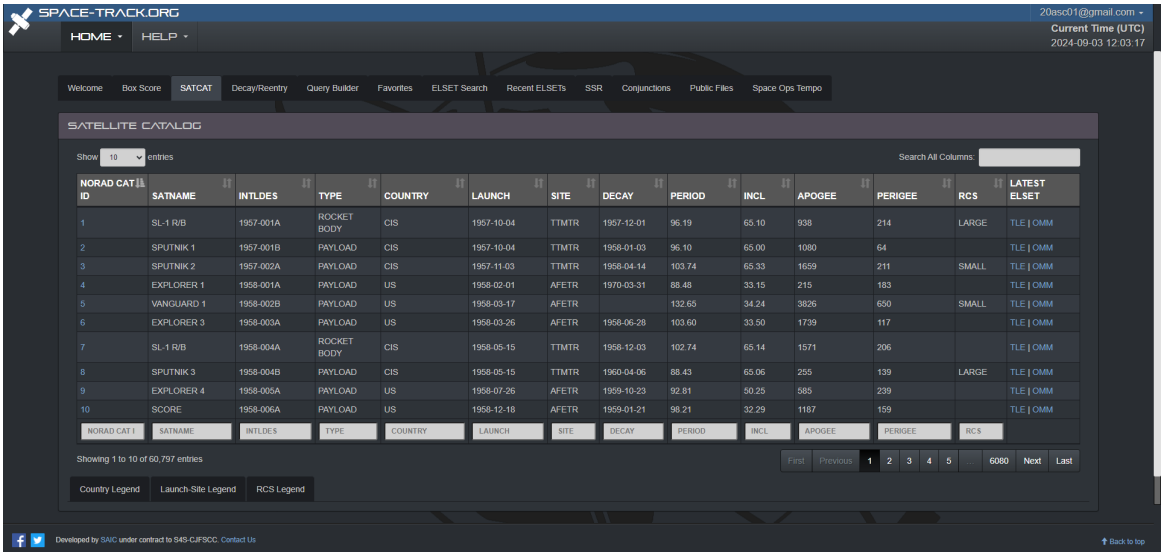


Figure 3.8 SATCAT

Some parameters that are going to be used in this project are the following:

NORAD ID: A sequential number assigned by the United States (US) Space Force to each object as it is cataloged.

Common Name: The generally recognized name associated with the object.

Object ID: Also known as the Committee on Space Research (COSPAR) designation or NASA Space Science Data Coordinated Archive (NSSDCA) ID, this uniquely identifies a space object. It consists of the launch year, a sequential launch number for that year, and up to a three-letter code indicating a component of that launch. For instance, 1990-037A and 1990-037B refer to objects from the 37th successful launch in 1990. Object A is the Space Shuttle and object B is the Hubble Space Telescope, both deployed during the same mission.

Epoch: It is the time at which the position of the object is defined for the given orbital parameters.

Eccentricity (e): Describes the shape of the orbit, ranging from 0 (circular) to just under 1 (highly elliptical). See Figure 3.9.

Semi-major Axis (a): It also gives information about the geometry of the orbit, specifically about its size. It is half of the distance between the periapsis and the apoapsis. See Figure 3.9.

Inclination (i): Informs about the vertical orientation of the orbital plane. It indicates the vertical tilt of the orbital plane with respect to the reference plane, measured at the ascending node and perpendicular to the intersection of these two planes. See Figure 3.9.

Right Ascension of the Ascending Node (Ω): Orients the orbit horizontally, describing the position of the ascending node (where the orbit crosses the reference plane from south to north) with respect to the vernal point of the reference frame, measured in the equatorial plane. See Figure 3.9.

Argument of Periapsis (ω): Describes the orientation of the orbit within the orbital plane. It is the angle between the ascending node and the periapsis (point of closest approach to the central body), measured in the orbital plane. See Figure 3.9.

Mean Anomaly (M): Positions the satellite along its orbit at a specific epoch. It represents the angle from the pericenter that a fictitious satellite would have if it were moving in a circular orbit with the same period. See Figure 3.9.

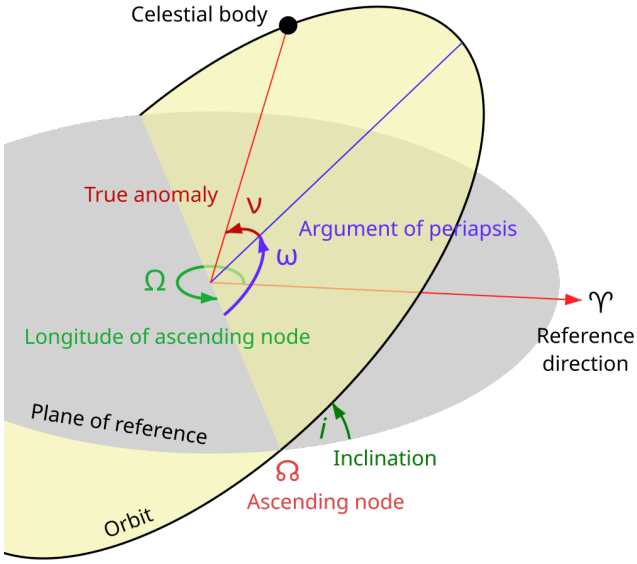


Figure 3.9 Classical Orbital Elements. Source: Wikipedia

However, the classical orbital elements, which we are going to use for the propagation methods, do not consider mean anomaly as one of them but true anomaly.

True Anomaly (ν): Defines the position of the satellite within the orbit at a specific epoch. It is the angle from the pericenter. A comparison with the Mean Anomaly is shown in Figure 3.10.

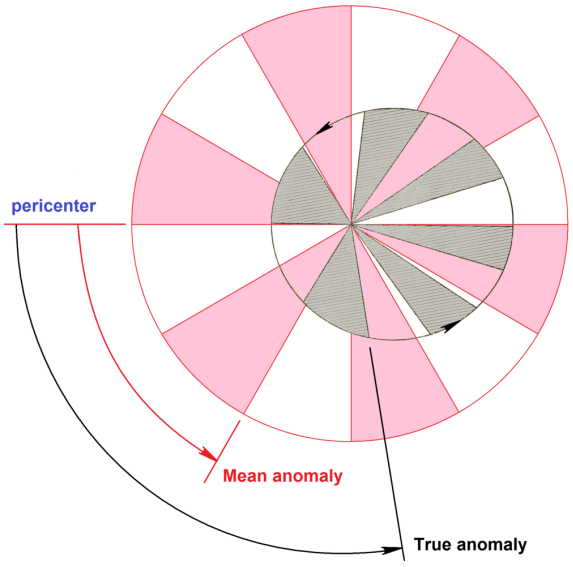


Figure 3.10 Comparison between Mean and True Anomaly. Source: Wikipedia

The way of computing True Anomaly (ν) is by solving Kepler's equation 3.3, which is nonlinear. Thus, a numerical solving method, such as Newton-Raphson, should be used to calculate Eccentric Anomaly (E). Then True Anomaly (ν) can be obtained with 3.5.

$$M = E - e \sin E \tag{3.3}$$

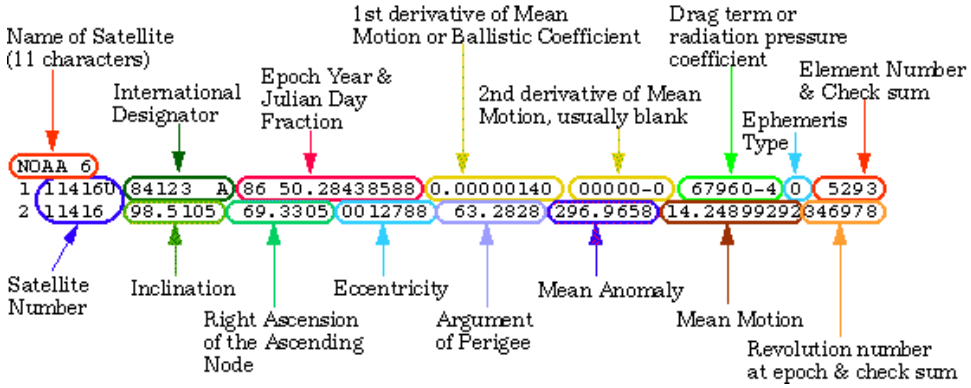


Figure 3.11 Two-Line Elements (TLE). Credits: Moutaman Mirghani

The Newton-Raphson method is an iterative root-finding algorithm. In this case, we want to find the root of the function $f(E) = E - e\sin E - M$. Doing the necessary iterations until $E_{n+1} - E_n$ is small enough with the following algorithm:

$$E_{n+1} = E_n - \frac{f(E_n)}{f'(E_n)}, \tag{3.4}$$

we can obtain E, which using the following equation will allow us to know the True Anomaly, ν:

$$\nu = 2\arctan(\sqrt{\frac{1+e}{1-e}}\tan E/2). \tag{3.5}$$

SpaceTrack also allows users to access historical element sets, conjunction predictions, and other data. However, it does not provide information about the physical size or shape of space objects.

The Two-Line Elements (TLE) set is a way of grouping all the information information descripted before and the parameters needed for the estimation of perturbations. The () data representation is suitable for the () model. The explanation of this representation of the parameters of the position and orbit of an object in space is provided in Figure 3.11.

All elements in Figure 3.11 are mean elements. These are calculated by a least squares estimation from observations of a satellite's orbit, and they remove periodic variations [28].

3.3.2 GCAT: General Catalog of Artificial Space Objects

The General Catalog of Artificial Space Objects (GCAT), maintained by Jonathan C. McDowell [29], complements Satellite Catalog (SATCAT) by providing additional information, especially the physical characteristics of space objects. The following GCAT parameters are relevant to this project:

JCAT number: This is the Jonathan Catalog ID to identify objects in space. It consists of a letter followed by a sequence number with five digits. For example, the letter S refers to the standard catalog that coincides with the Satellite Catalog (SATCAT). The five digits correspond to the NORAD ID.

Length: Longest dimension of the main body of the object in meters.

Diameter: Second longest dimension of the main body of the object in meters. The cross section can be calculated as the length times the diameter.

Span: It is the largest dimension in meters of the object, not the main body. This includes structures such as booms, solar panels, or antennae.

Shape: A qualitative description of the shape of the object. This is provided in an approximate way. Some of the information in this field has been extracted from the Royal Aircraft Establishment (RAE) table of Earth satellites.

3.4 Propagation algorithms overview

Once the classical orbit elements are known, the next step is to compute the positions and velocities of space objects as they move along their orbits over time. This is achieved using orbital propagators. Some propagators use simplified models, considering only the two-body gravitational interaction. More sophisticated propagators account for perturbations such as atmospheric drag, solar radiation pressure, gravitational forces from third bodies (like the Moon and Sun), and the non-spherical shape of the Earth. Generally, there is a trade-off between accuracy and computational cost; simpler propagators are faster but less accurate, while more complex propagators are more accurate but computationally intensive.

Propagation algorithms are used to advance the state (position and velocity) of a spacecraft in time. These algorithms can be broadly classified as analytical or numerical integrators. Analytic propagators model spacecraft motion over time using mathematical equations derived from celestial mechanics, while integrator propagators directly calculate the forces acting on the spacecraft (including perturbations) at each time step and numerically integrate the equations of motion. An example of an analytic propagator is Farnocchia[30]; on the other hand, Cowell's allows one to propagate using integration approaches [31].

The suitability of a propagator depends on the type of orbit and desired accuracy. Table 3.3 summarizes the applicability of different propagators used in this project to different orbit types [32]:

Table 3.3 Orbital Propagators. Y (Yes) if they work with the kind of orbit they make reference to; N (No) if not

Propagator	Elliptical	Parabolic	Hyperbolic
Farnocchia	Y	Y	Y
Cowell	Y	Y	Y
Danby	Y	Y	Y
Pimienta	Y	Y	N
Vallado	Y	Y	Y

3.4.1 Keplerian orbit model

The Keplerian orbit model is the simplest representation of orbital motion, describing the orbit of one body (m_2) around another (m_1). The bodies are supposed to be spherical point masses, and they only interact with each other (no other bodies interact with these two). The result is an equation, the Isaac Newton law of universal gravitation, which describes the gravitational forces F, which each mass applies to the other. The gravitational force, F_{12}, applied by m_2 on m_1 is presented in Equation 3.6 and the force, F_{21}, applied by m_1 on m_2 is shown in Equation 3.7.

$$F_{12} = G\frac{m_1 m_2}{r^2}\hat{\mathbf{u}}_r \tag{3.6}$$

$$F_{21} = -G\frac{m_1 m_2}{r^2}\hat{\mathbf{u}}_r \tag{3.7}$$

In these equations, G is the gravitational constant, r is the distance between the centers of the two masses, and $\hat{\mathbf{u}}_r$ is the unitarian vector that points from m_1 to m_2. Applying Newton's second law to each mass ($\mathbf{F}_{12} = m_1\ddot{\mathbf{R}}_1$ and $\mathbf{F}_{21} = m_2\ddot{\mathbf{R}}_2$) and considering the relative position vector $\mathbf{r} = \mathbf{R}_2 - \mathbf{R}_1$, we obtain the two-body equation of motion:

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^2}\hat{\mathbf{u}}_r = -\frac{\mu}{r^3}\mathbf{r}, \tag{3.8}$$

where $\mu = G(m_1 + m_2)$ is the gravitational parameter. This is a non-linear second-order vector differential equation. When solved we come up with two vectors of constant components, thus, 6 integration constants. If we suppose a fixed axis system in m_1, we can divide the vectorial equation 3.8 into three scalar equations, each for every of its components:

$$\ddot{x} = -\frac{\mu}{r^3}x, \qquad \ddot{y} = -\frac{\mu}{r^3}y, \qquad \ddot{z} = -\frac{\mu}{r^3}z \tag{3.9}$$

These equations characterize the problem of the two bodies. They can be generalized to the n-body problem, considering the gravitational interactions among multiple bodies as seen in Equation 3.10.

$$\ddot{\mathbf{r}}_i = -\sum_{j=1, j\neq i}^{n} \frac{Gm_j(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}^3} \tag{3.10}$$

Another approach to solving equation 3.8 is by supposing that the orbit is a conic section. The formula used is Equation 3.11. Analytical propagators use this approach.

$$r(\nu) = \frac{a(1-e^2)}{1 + e\cos\nu} \tag{3.11}$$

Perturbations

The equations of the n-bodies (Equation 3.10) do not take into account some other effects, apart from the gravitational forces the n-bodies apply among them. These effects are called perturbations and are accounted for in the equation of two bodies (Equation 3.8), including a vector **p**. The result of this addition is Equation 3.12.

$$\ddot{\mathbf{r}} = -\frac{\mu}{r^3}\mathbf{r} + \mathbf{p} \tag{3.12}$$

The effects that other objects apply over the 2 main bodies,third-body gravitational effects, are described with vector **p'**, which is shown in Equation 3.13.

$$\mathbf{p'} = -\sum_{j=1, j\neq i}^{n} \frac{Gm_j(\mathbf{r}_i - \mathbf{r}_j)}{r_{ij}^3} \tag{3.13}$$

In Equation 3.13 i makes references to the perturbed body and j to the perturbing bodies which do not include the main body.

Apart from **p'**, perturbations can include other sources such as the non-perfect spherical Earth shape, which leads to uneven distribution of the gravitational field. Earth's oblateness, for example, causes gravity to vary with latitude and altitude. The vector $\mathbf{p}_o$ represents this perturbation and is calculated with Equation3.14.

$$\mathbf{p}_o = \frac{3}{2}\frac{\mu J_2 R^2}{r^4}\left[\frac{x}{r}\left(5\frac{z^2}{r^2} - 1\right)\hat{i} + \frac{y}{r}\left(5\frac{z^2}{r^2} - 1\right)\hat{j} + \frac{z}{r}\left(5\frac{z^2}{r^2} - 3\right)\hat{k}\right] \tag{3.14}$$

In Equation 3.14 J_2 is the second zonal harmonic gravity constant for oblateness, R is the equatorial radius of Earth. The asymmetry across the equator is considered by J_3 but this is not going to be considered, since it is neglectable for Earth: $J_2 = 0.001082$ and $J_3 = -0.0000025$.

Another source of perturbations is the solar radiation pressure. This perturbation depends on the object geometry, the materials of the surface, and the attitude. Modeling this perturbation can be challenging. However, thanks to the cannonball model, a simplification allows us to model this perturbation with Equation 3.15. In this equation $\hat{\mathbf{u}}_{OS}$ is the unit vector that goes from the object to the Sun, ν is the shadow function (0 if the object is in the shadow zone or 1 if not), $S = 1367W/m^2$ is the solar constant, c is the speed of light, C_R is the coefficient of reflectivity and A_S is the absorbing area.

$$\mathbf{p}_{SR} = -\nu\frac{S}{c}\frac{C_R A_S}{m}\hat{\mathbf{u}}_{OS} \tag{3.15}$$

Atmospheric drag is another perturbation that affects satellites orbiting Low Earth Orbits. The lower the altitude of the satellite, the larger the atmospheric drag. For orbits higher than 600 km, it is not a big problem, because drag is very small. The equation used to model this is Equation 3.16.

$$\mathbf{p}_D = \frac{F_D}{m_2} = -\frac{C_D A_S \rho V^2}{2m_2} \tag{3.16}$$

In Equation 3.16, C_D is the drag coefficient, A_S is the satellite's cross-sectional area projected in the direction of motion, ρ is the density of the fluid at the satellite's altitude and V is the velocity relative to the atmosphere.

Taking into account these sources of perturbation, the total perturbation vector **p** is the sum of individual perturbations shown in Equation 3.17.

$$\mathbf{p} = \mathbf{p'} + \mathbf{p}_{SR} + \mathbf{p}_O + \mathbf{p}_D \tag{3.17}$$

3.4.2 Farnocchia's propagator [1]

Farnocchia's propagator is an analytical method that propagates orbits by advancing the true anomaly. It takes the initial $\mathbf{r}_0$ and $\mathbf{v}_0$, calculates from them the classical orbit parameters. Then, it increases the true anomaly by a value corresponding to a time step Δt (using 3.18) and gets **r** and **v** at that time. This algorithm of propagation has the characteristic that is valid for all types of orbits: elliptic, parabolic, and hyperbolic. The relation between the true anomaly and the time is given by Equation 3.18, where t_P is the pericenter passage time and q is the pericenter distance (for elliptic orbits: $q = a(1-e)$). This equation can be rewritten depending on the value of the eccentricity.

$$\int_0^{\nu} \frac{1}{1+e\cos\nu} d\nu = \sqrt{\frac{\mu}{q^3(1+e)^3}}(t-t_P) \tag{3.18}$$

For the elliptic orbits, $e < 1$:

$$(t-t_P) = (E - e\sin E)\sqrt{\frac{q^3}{\mu(1-e)^3}} \tag{3.19}$$

Starting with initial position $\mathbf{r}_0$ and velocity $\mathbf{v}_0$, we calculate the initial orbital elements $(a, e, i, \Omega, \omega, \nu_0)$. From the initial true anomaly ν_0, we calculate the initial Eccentric Anomaly E_0 using Equation 3.20.

$$E_0 = 2\arctan\left(\sqrt{\frac{1-e}{1+e}}\tan\left(\frac{\nu_0}{2}\right)\right) \tag{3.20}$$

Then, we calculate the initial Mean Anomaly $M_{E,0}$ with Equation 3.21 and the initial time since pericenter passage Δt_0 with Equation 3.22.

$$M_{E,0} = (E_0 - e\sin E_0) \tag{3.21}$$

$$\Delta t_0 = M_{E,0}\sqrt{\frac{q^3}{\mu(1-e)^3}} \tag{3.22}$$

To propagate to a future time $t = t_0 + tof$, we calculate the new time since the pericenter passage $\Delta t_1 = \Delta t_0 + tof$. Now, we need to obtain $M_{E,1}$ from Equation 3.23.

$$M_{E,1} = \Delta t_1\sqrt{\frac{\mu(1-e)^3}{q^3}} \tag{3.23}$$

We then need to solve Kepler's equation for the new Eccentric Anomaly E_1, Equation 3.35. This equation is solved numerically using the Newton-Raphson method, which algorithm is described in Equation 3.25.

$$M_{E,1} = E_1 - e \sin E_1 \tag{3.24}$$

$$E_{k+1} = E_k - \frac{E_k - e \sin E_k - M_{E,1}}{1 - e \cos E_k} \tag{3.25}$$

Once E_1 is obtained, we use the equation 3.26 to get ν_1.

$$E_1 = 2 \arctan\left(\sqrt{\frac{1-e}{1+e}} \tan\left(\frac{\nu_1}{2}\right)\right) \rightarrow \nu_1 = 2 \arctan\left(\sqrt{\frac{1+e}{1-e}} \tan\left(\frac{E_1}{2}\right)\right) \tag{3.26}$$

Finally, with the updated orbital elements (only true anomaly has changed directly in this step), we can compute the new position $\mathbf{r}_1$ and velocity $\mathbf{v}_1$.

Equation 3.25 has the problem that is singular at $(e, E) = (1, 0)$. In this case, the term $1 - e \cos E < \delta$, where δ is a small parameter. The solution can be found considering the problem as a Strong Elliptic problem if $e < 1 - \delta$, being δ around 0.01, and the solution can be obtained with the standard Newton-Raphson method described above.

However, if $1 - \delta \leq e < 1$, we have to compute E_δ, defined in Equation 3.27.

$$E_\delta = arccos\left(\frac{1-\delta}{e}\right) \tag{3.27}$$

Then, we will check if this is Strong Elliptic or Near Parabolic comparing M_{E_δ}, defined in Equation 3.28, and M_E, defined in Equation 3.29 . If it is Strong Elliptic ($M_{E_\delta} \leq |M_E|$), we would apply equations 3.25, while in the case of Near Parabolic ($M_{E_\delta} > |M_E|$) we will proceed as mentioned in the section dedicated to it.

$$M_{E_\delta} = (E_\delta - e \sin E_\delta) \tag{3.28}$$

$$M_E = \Delta t_1 \sqrt{\frac{\mu(1-e)^3}{q^3}} \tag{3.29}$$

For parabolic orbits, e = 1:

$$(t - t_P) = \left(D + \frac{D^3}{3}\right) \sqrt{\frac{2q^3}{\mu}} \tag{3.30}$$

Starting with initial position $\mathbf{r}_0$ and velocity $\mathbf{v}_0$, we calculate the initial orbital elements $(a, e, i, \Omega, \omega, \nu_0)$. From the initial true anomaly ν_0, we calculate the initial Parabolic Anomaly D_0 using Equation 3.31.

$$D_0 = \tan \frac{\nu_0}{2} \tag{3.31}$$

Then, we calculate the initial Mean Anomaly $M_{P,0}$ with Equation 3.32 and the initial time since pericenter passage Δt_0 with Equation 3.33.

$$M_{P,0} = \left(D_0 + \frac{D_0^3}{3}\right) \tag{3.32}$$

$$\Delta t_0 = M_{P,0} \sqrt{\frac{2q^3}{\mu}} \tag{3.33}$$

To propagate to a future time $t = t_0 + tof$, we calculate the new time since the pericenter passage $\Delta t_1 = \Delta t_0 + tof$. Now, we need to obtain $M_{P,1}$ from Equation 3.34.

$$M_{P,1} = \Delta t_1 \sqrt{\frac{\mu}{2q^3}} \tag{3.34}$$

We need to solve the Equation 3.35 for the Parabolic Anomaly D_1. The solution is found analytically with the process described in Equation 3.36

$$M_{P,1} = \left(D_1 + \frac{D_1^3}{3} \right) \tag{3.35}$$

$$B = \frac{3M_{P,1}}{2} \rightarrow A = (B + \sqrt{1 + B^2})^{\frac{2}{3}} \rightarrow D_1 = \frac{2AB}{1 + A + A^2} \tag{3.36}$$

Once D_1 is obtained, we use the equation 3.37 to get ν_1.

$$\nu_1 = \arctan 2D_1 \tag{3.37}$$

For the hyperbolic orbits, $e > 1$:

$$(t - t_P) = (e \sinh F - F) \sqrt{\frac{q^3}{\mu(e-1)^3}} \tag{3.38}$$

Starting with initial position $\mathbf{r}_0$ and velocity $\mathbf{v}_0$, we calculate the initial orbital elements $(a, e, i, \Omega, \omega, \nu_0)$. From the initial true anomaly ν_0, we calculate the initial Hyperbolic Anomaly F_0 using Equation 3.39.

$$F_0 = 2 \tanh^{-1} \left(\sqrt{\frac{e-1}{e+1}} \tan \left(\frac{\nu_0}{2} \right) \right) \tag{3.39}$$

Then, we calculate the initial Mean Anomaly $M_{H,0}$ with Equation 3.40 and the initial time since pericenter passage Δt_0 with Equation 3.41.

$$M_{H,0} = (e \sinh F_0 - F_0) \tag{3.40}$$

$$\Delta t_0 = M_{H,0} \sqrt{\frac{q^3}{\mu(e-1)^3}} \tag{3.41}$$

To propagate to a future time $t = t_0 + tof$, we calculate the new time since the pericenter passage $\Delta t_1 = \Delta t_0 + tof$. Now, we need to obtain $M_{H,1}$ from Equation 3.42.

$$M_{H,1} = \Delta t_1 \sqrt{\frac{\mu(e-1)^3}{q^3}} \tag{3.42}$$

We then need to solve Kepler's equation for the new Eccentric Anomaly E_1, Equation 3.43. This equation is solved numerically using the Newton-Raphson method, which algorithm is described in Equation 3.44.

$$M_{H,1} = (e \sinh F_1 - F_1) \tag{3.43}$$

$$F_{k+1} = F_k - \frac{e \sinh F_k - F_k - M_{H,1}}{e \cosh F_k - 1} \tag{3.44}$$

Once F_1 is obtained, we use the equation 3.45 to get ν_1.

$$F_1 = 2 \tanh^{-1} \left(\sqrt{\frac{e-1}{e+1}} \tan \left(\frac{\nu_1}{2} \right) \right) \rightarrow \nu_1 = 2 \tan^{-1} \left(\sqrt{\frac{e+1}{e-1}} \tanh \left(\frac{F_1}{2} \right) \right) \tag{3.45}$$

Finally, with the updated orbital elements (only true anomaly has changed directly in this step), we can compute the new position $\mathbf{r}_1$ and velocity $\mathbf{v}_1$.

The Equation 3.44 has the problem that is singular at $(e, F) = (1, 0)$. In this case, the term $e \cosh F - 1 < \delta$, where δ is a small parameter. The solution can be found considering the problem as a strong hyperbolic problem if $e > 1 - \delta$, being δ around 0.01, and the steps for the resolution would be the mentioned before.

However, if $1 - \delta \leq e < 1$, we have to compute F_δ, defined in Equation 3.46.

$$F_\delta = arccosh\left(\frac{1+\delta}{e}\right) \tag{3.46}$$

Then, we will check if this is Strong Elliptic or Near Parabolic comparing M_{H_δ}, defined in Equation 3.47, and M_H, defined in Equation 3.48 . If it is Strong Hyperbollic ($M_{H_\delta} \leq |M_H|$), we would apply equations 3.44, while in the case of Near Parabolic ($M_{H_\delta} > |M_H|$) we will proceed as mentioned in the section dedicated to it.

$$M_{H_\delta} = (e \sinh F_\delta - F_\delta) \tag{3.47}$$

$$M_H = \Delta t_1 \sqrt{\frac{\mu(e-1)^3}{q^3}} \tag{3.48}$$

For Near Parabolic:

$$(t - t_P) = \chi_{NP}(e, D)\sqrt{\frac{2q^3}{\mu}} \tag{3.49}$$

Starting with initial position $\mathbf{r}_0$ and velocity $\mathbf{v}_0$, we calculate the initial orbital elements $(a, e, i, \Omega, \omega, \nu_0)$. From the initial true anomaly ν_0, we calculate the initial Parabolic Anomaly D_0 using Equation 3.50.

$$D_0 = \tan\frac{\nu_0}{2} \tag{3.50}$$

Then, we calculate $\chi_{NP,0}(e, D_0)$ with Equation 3.51. This equation allows us to ensure regularity around $e = 1$ and it results from introducing the Taylor expansions for E and $\sin E$ in Equation 3.19 and the Taylor expansions for F and $\sinh F$ in Equation 3.38. Also, we calculate the initial time since the pericenter passage Δt_0 with Equation 3.52.

$$\chi_{NP,0}(e, D) = \frac{\sqrt{2}}{\sqrt{1+e}} D_0 + \frac{\sqrt{2}}{(\sqrt{1+e})^3}\frac{D_0^3}{3}\sum_{k=0}^{\infty}\left(e - \frac{1}{2k+3}\right)x_0^k$$

$$x_0 = \frac{e-1}{e+1}D_0^2 \tag{3.51}$$

$$\Delta t_0 = \chi_{NP,0}(e, D_0)\sqrt{\frac{2q^3}{\mu}} \tag{3.52}$$

To propagate to a future time $t = t_0 + tof$, we calculate the new time since the pericenter passage $\Delta t_1 = \Delta t_0 + tof$. Now, we need to obtain $\chi_{NP,1}(e, D)$ from Equation 3.53.

$$\chi_{NP,1}(e, D) = \Delta t_1 \sqrt{\frac{\mu}{2q^3}} \tag{3.53}$$

We can use the Newton's method to obtain D_1 from the Taylon expansion $\chi_{NP,1}(e, D_1)$. The algorithm is described in Equation 3.54. In this way, the singularity is solved because $\frac{\partial \chi_{NP}(e,D_k)}{\partial D}$ does not go to 0 when $e \to 1$.

$$D_{k+1} = D_k - \frac{\chi_{NP}(e, D_k) - M_P}{\frac{\partial \chi_{NP}(e,D_k)}{\partial D}} \tag{3.54}$$

Once D_1 is obtained, we use the equation 3.55 to get ν_1.

$$\nu_1 = \arctan 2D_1 \tag{3.55}$$

3.4.3 Cowell's propagator

Thanks to this integrator propagator, perturbations that deviate from the trajectory of a satellite could be considered. It is recommended when the perturbation acceleration is the same order of magnitude as the gravity between the two main bodies. In many cases this condition is met: for a satellite in LEO, the Earth oblateness effect is three orders of magnitude smaller and solar radiation, atmospheric drag, or the effect of the Moon over the satellite are even smaller [33].

In this algorithm, the perturbed two-body equation of motion, Equation 3.56, is solved with a numerical solver.

$$\begin{aligned} \ddot{\mathbf{r}} &= -\tfrac{\mu}{r^3}\mathbf{r} + \mathbf{p} \\ \mathbf{p} &= \mathbf{p'} + \mathbf{p}_{SR} + \mathbf{p}_O + \mathbf{p}_D \end{aligned} \tag{3.56}$$

Equation 3.56 is a second-order differential equation. Cowell's method uses a numerical integrator to solve this equation step by step. The integrator takes the current position and velocity of the spacecraft, calculates the total acceleration (including perturbations) at that point, and then uses that acceleration to estimate the position and velocity at a later time. This process is repeated iteratively to propagate the orbit forward in time. The choice of numerical integrator has to be made by weighing between accuracy and computation efficiency. Common choices include:

- **Runge-Kutta 4-5 (RK45):** offers a good balance between precision and computational cost. It is a variable step size method: Automatically adjusts the time step to maintain the desired level of accuracy. It uses a fourth-order method for the main integration step and a fifth-order method for error estimation. The brilliance of this approach is that the 5th-order method reuses many of the same function evaluations (calculations of the derivative) that the 4th-order method needs.
- **Runge-Kutta 2-3 (RK23):** It is a lower order variable step size method. It is faster than RK45 but less accurate.
- **Runge-Kutta 7-8 (RK78):** It is a high-order variable step size. It is more accurate than RK45, but it is slower.
- **Runge-Kutta 8(5,3)** / **Dormand-Prince 8(5,3):** it uses an eight-order method for the main integration step and a five-order method for error estimation and step size adaptation. A sophisticated high-order, variable step size method. It provides very high accuracy, but is computationally more expensive than RK45. The "8(5,3)" notation refers to the orders of the various stages within the method. It is one specific implementation within the Runge-Kutta family, designed for high precision.

The Runge-Kutta approach is described as follows. The first step is to convert the second order differential equation (3.56) to a system of two first order differential equations. This is done using the "state-space" representation. We introduce a new variable, the velocity vector $\mathbf{v} = d\mathbf{r}/dt$. Now we can rewrite in the system presented in Equation 3.57

$$\begin{aligned} d\mathbf{r}/dt &= \mathbf{v} \\ d\mathbf{v}/dt &= -(\mu/r^3)\mathbf{r} + \mathbf{p} \end{aligned} \tag{3.57}$$

This two equations can be combined in a single state vector, $\mathbf{y}$, (Equation 3.58) which can be expressed as a first order vectorial equation (Equation 3.59).

$$\mathbf{y} = [\mathbf{r}, \mathbf{v}] = [x, y, z, v_x, v_y, v_z] \tag{3.58}$$

$$\frac{d\mathbf{y}}{dt} = \mathbf{f}(t, \mathbf{y}) = [\mathbf{v}, -(\mu/r^3)\mathbf{r} + \mathbf{p}] \tag{3.59}$$

Now given $\mathbf{y}$ at time t_n, $\mathbf{y}_n$, a time step Δt_n or h and the number of stages s of the method, we can calculate $\mathbf{y}_{n+1}$ at time $t_{n+1} = t_n + \Delta t_n$ with equation **??**.

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h \sum_{i=1}^{s} b_i k_i \tag{3.60}$$

In this equation k_i are the slope coefficients for Explicit Runge–Kutta methods, they are calculated with Equation 3.61; for Implicit methods, they are calculated with Equation 3.62

$$\mathbf{k}_i = \mathbf{f}\left(t_n + c_i h, \mathbf{y}_n + h\sum_{j=1}^{i-1} a_{ij}\mathbf{k}_j\right) \tag{3.61}$$

$$\mathbf{k}_i = \mathbf{f}\left(t_n + c_i h, \mathbf{y}_n + h\sum_{j=1}^{s} a_{ij}\mathbf{k}_j\right) \tag{3.62}$$

Each method is accompanied by its Butcher table, which has the form of the equation in 3.63.

$$\begin{array}{c|cccc} c_1 & a_{11} & a_{12} & \cdots & a_{1s} \\ c_2 & a_{21} & a_{22} & \cdots & a_{2s} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ c_s & a_{s1} & a_{s2} & \cdots & a_{ss} \\ \hline & b_1 & b_2 & \cdots & b_s \\ & b_1^* & b_2^* & \cdots & b_s^* \end{array} \tag{3.63}$$

The row of b_i^* is added only for adaptive and implicit methods. Explicit methods are those in which the matrix a_{ij} is lower triangular. On the other hand, in implicit methods, the matrix a_{ij} is fully populated, and it is very helpful when solving stiff equations (those for which certain numerical methods are unstable; the step size must be extremely small). Adaptive methods are capable of making and estimating errors by calculating a solution of order p and another of order $p-1$. These two solutions are compared to measure the error and adapt the step size h. If the error is high, this is that the error is above a tolerance limit, and the solution is recalculated with a lower step size. Opposite, if the error is low, this is that the error is below a tolerance limit, the solution is recalculated with a higher step size. This method allows to get the optimal step size, which has the advantage of saving computation time. Examples of these type of Runge-Kutta algorithms are Runge-Kutta 4-5, Runge-Kutta 2-3 or Runge-Kutta 7-8, among others. The butcher table for this method is shown in 3.64. The higher order solution p is calculated with Equation 3.60 and the lower order $p-1$ solution is calculated with Equation 3.65. As a clarification the index s is the number of stages, for instance, the RK45 method uses 6 stages or the Dormand-Prince 8(5,3) uses 13 stages.

$$\begin{array}{c|ccccc} 0 & & & & & \\ c_2 & a_{21} & & & & \\ c_3 & a_{31} & a_{32} & & & \\ \vdots & \vdots & \vdots & \ddots & & \\ c_s & a_{s1} & a_{s2} & \cdots & a_{s,s-1} & \\ \hline & b_1 & b_2 & \cdots & b_{s-1} & b_s \\ & b_1^* & b_2^* & \cdots & b_{s-1}^* & b_s^* \end{array} \tag{3.64}$$

$$\mathbf{y}_{n+1}^* = \mathbf{y}_n + h\sum_{i=1}^{s} b_i^* k_i \tag{3.65}$$

The error is then obtained with Equation 3.66.

$$\mathbf{e}_{n+1} = \mathbf{y}_{n+1} - \mathbf{y}_{n+1}^* = h\sum_{i=1}^{s}(b_i - b_i^*)k_i \tag{3.66}$$

The disadvantages of the Cowell's model are the error accumulation and that it requires force models. Like all numerical integration methods, Cowell's method is subject to error accumulation over time. This can be mitigated by using a high-order integrator and a small time step, but it is always a consideration because this will increase the computational cost. However, modeling perturbing forces such as atmospheric drag or solar radiation pressure can be challenging.

3.4.4 Simplified General Perturbation (SGP) model propagator

The Simplified General Perturbation (SGP) model is an analytical model. This method calculates the coordinates of the satellite and the velocity at any instant including the most important natural perturbation. Its main advantage is that it is very fast, more than its equivalent numerical model. It has an error of about one kilometer and grows between one and three kilometers per day [34].

3.5 Observability algorithm

In module $observability.py$ the necessary computations are performed to count the number of observed satellites that the observant satellite or satellites is able to capture. This is based on an algorithm that consists of a process of nested computations, organized in steps. Here, the steps and simplifications needed are shown:

3.5.1 First step

The first step is to determine whether an object is observable by its size, the distance to the observant satellite, and the camera sensor and lens properties. In this process, we will get the minimum size an object should have to be visible depending on the distance between the camera and the observed object through the different steps of the discretized time domain. If the actual size of this object is greater than the minimum observable size for any of the steps of the discretized time domain, then we will proceed to the second step.

For this, the distance between the observed satellite and the observer satellite is calculated for each step of the discretized time.

Another parameter to consider is the size of the pixels. The data we have are the camera resolution in pixels, the sensor width in meters, and the sensor height in meters. The camera resolution is defined in Equation 3.67:

$$camera\,resolution = resolution\,width \cdot resolution\,height \tag{3.67}$$

being resolution width and resolution height measurements in pixels. Aspect ratio can be calculated with either formula 3.68 or 3.69:

$$aspect\,ratio = \frac{sensor\,width}{sensor\,height}, \tag{3.68}$$

$$aspect\,ratio = \frac{resolution\,width}{resolution\,height}. \tag{3.69}$$

The approach that we will follow is getting the aspect ratio using equation 3.68. Then we can obtain the resolution width (see Equation 3.70) from Equation 3.69.

$$resolution\,width = aspect\,ratio \cdot resolution\,height \tag{3.70}$$

Equation 3.70 can be substituted in equation 3.67, so we get the resolution height (see Equation 3.71).

$$resolution\,height = \sqrt{\frac{camera\,resolution}{aspect\,ratio}} \tag{3.71}$$

The resolution width can be computed with equation 3.70. Now, we can get the pixel size using Equation 3.72.

$$pixel\,size = \frac{sensor\,height}{resolution\,height} = \frac{sensor\,width}{resolution\,width}. \tag{3.72}$$

Now, that we know the pixel size of the sensor, we have to calculate the diameter of the airy disk. The airy disk is a phenomenon that occurs due to the wave nature of light. When light travels through the aperture

of an optic system, the light tends to bend around the edges of the aperture. The light rays diverge and move out of phase, so they begin to interfere with each other. This causes the amplitude to increase in some places because the waves add to each other and decrease in other areas because the waves cancel out with others (see Figure 3.12. This is called diffraction. A point source of light will then be imaged not as a perfect point but as a blurry spot called the airy disk. This airy disk is the fundamental limit to how sharp an image can be, and dictates the finest detail a lens can resolve.

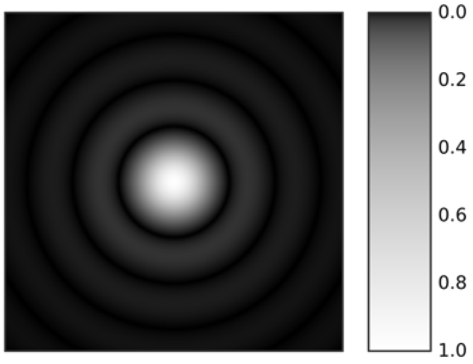


Figure 3.12 Computer generated image of an Airy disk. Credits: Wikipedia

The diameter of the airy disk, d, is approximated by the diameter of the first null of this, which is the area that contains approximately 84% of the total power [35]. The diameter is calculated with Equation 3.73:

$$\frac{d}{2} = 1.22\lambda N, \tag{3.73}$$

where N is the f-number of the lens and λ is the light wavelength which is considered to be the average of the visible spectrum, $550nm$.

The size of the airy disk in relation to the pixel size is crucial for understanding image resolution. It is not about the Airy disk hitting just one pixel, but rather about comparing their relative sizes to determine the limiting factor on resolution.

This airy disk is supposed to hit just one pixel in the camera sensor. So, if the airy disk diameter of the light gets to be similar in size to a pixel of the sensor matrix, that pixel gets excited and represents that amount of light in an image (see Figure 3.13).

If the airy disk is smaller than the pixel size, the light from a point source is concentrated within a small area on the pixel and it would not be as excited as if the airy disk were bigger. In this case, the pixel will represent a smaller intensity of light, a weaker signal. The pixel will not be differentiated from their neighbors (see Figure 3.14. The limiting factor would be the size of the pixels; the resolution is pixel-limited. [36]. In this case, the size of an object can be as small as possible. The smaller the object, the lower the light that is going to emit and the fainter the brightness of the pixel will be. In order to obtain the

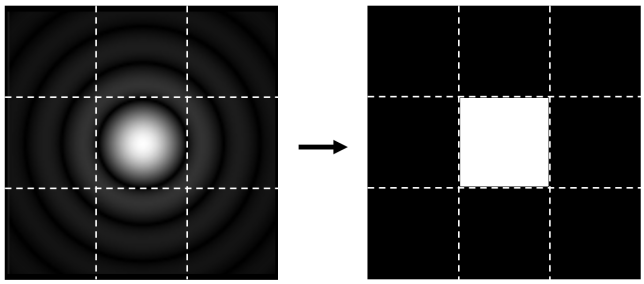


Figure 3.13 Airy disk is similar in size to the pixel size.

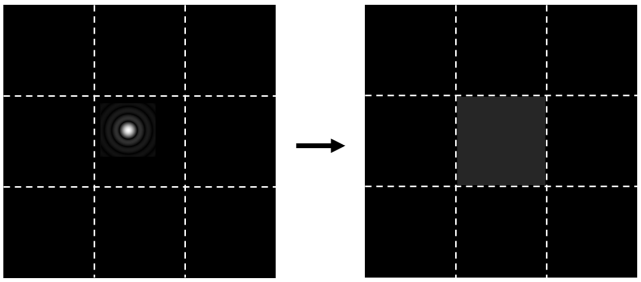


Figure 3.14 Airy disk is smaller than pixel size

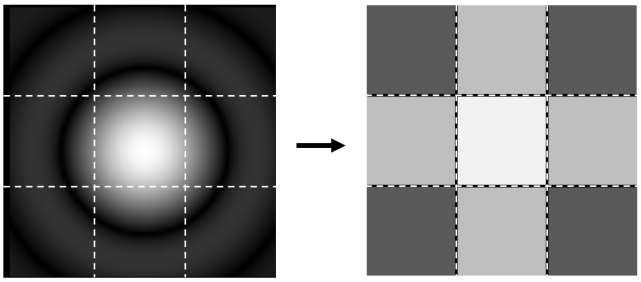


Figure 3.15 Airy disk is bigger than pixel size

light of the object well represented in the pixel, we would need an object in space whose projection on the sensor matrix occupies the size of a pixel.

If the airy disk is larger than the pixel size, the light from a point source will spread over multiple pixels (see Figure 3.15). In this case, the resolution is primarily limited by diffraction (the airy disk size), and the pixel size does not fully limit the resolution. For this reason, since the diameter of the airy disk is normally larger than the pixel size, the diameter of the airy disk is said to be the theoretical maximum resolution for an optical system. In other words, the diameter of the airy disk corresponds to the size of the smallest object that a lens can resolve. [37]

Other factors that may reduce the maximum resolution are imperfections and aberrations, but these are due to manufacturing errors. They are not going to be considered because a satellite usually mounts high-accuracy lenses.

Considering both the Airy disk diameter and the pixel size, we choose the larger of these two values as the effective resolution limit at the sensor level. Then, taking the value of the different distances between the observed and observer satellites over the propagation time, the minimum size that an object should have is obtained with Equation 3.74.

$$size_{min,i} = \frac{max(d, pixel\ size) \cdot distance_i}{focal\ length} \tag{3.74}$$

If the actual size of the object is bigger or equal to the minimum size an object should have to be resolvable by the image sensor for any of the steps of the discretized time domain, then we will continue to the following step.

3.5.2 Second step

In this stage of the process, we consider the pointing of the camera (the x axis of the body-fixed frame) and the up direction (the negative y axis of the body-fixed frame). With this we determine if the target is within the vertical and horizontal Field of View (FOV). To begin with, we will determine the coordinates of the

observed object with respect to the body-fixed frame of the observant satellite, $[x_{target}, y_{target}, z_{target}]_{body}$. See Figure 3.16

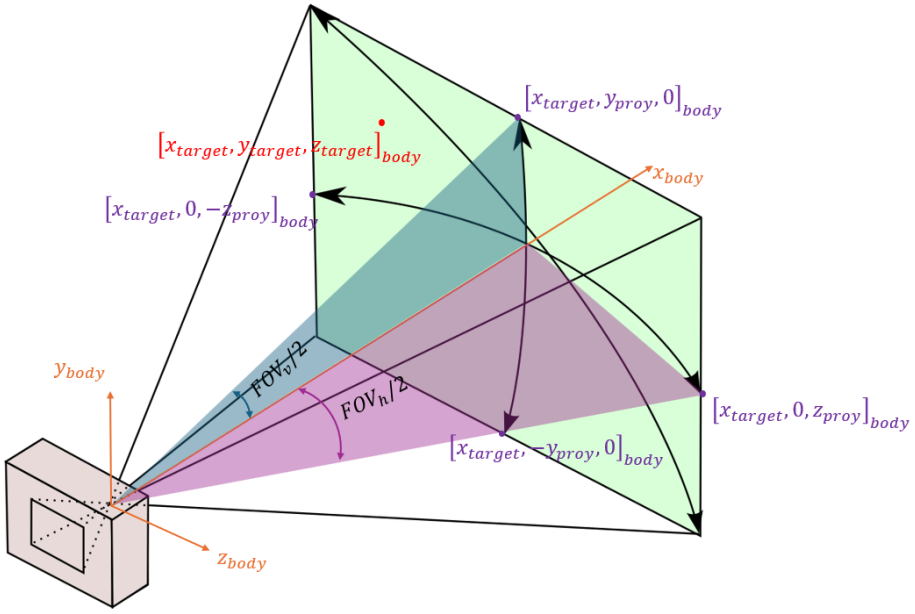


Figure 3.16 Vertical and horizontal field of view

Then, the formulas to obtain the vertical and horizontal Field of View (FOV) are the ones describes in Equation 3.75 and 3.76 [38]:

$$FOV_v = 2\arctan\left(\frac{\frac{sensor\ height}{2}}{focal\ lenght}\right) \tag{3.75}$$

$$FOV_h = 2\arctan\left(\frac{\frac{sensor\ width}{2}}{focal\ lenght}\right) \tag{3.76}$$

Now, we calculate the range of values that y_{body} would have in a plane perpendicular to the pointing vector, oriented in the upward direction and containing the observed object. This threshold is $[-y_{proy}, y_{proy}]$. The same process is repeated in the z_{body} direction, where the threshold is $[-z_{proy}, z_{proy}]$. See Figure 3.16). If the conditions described in Equations 3.77 and 3.78 are met, then the observed object is within the vertical and horizontal Field of View (FOV).

$$[-y_{proy} \leq y_{target} \leq y_{proy}] \tag{3.77}$$

$$[-z_{proy} \leq z_{target} \leq z_{proy}] \tag{3.78}$$

Due to the computer nature, a continuous spectrum has to be discretized. When analyzing data in the discreet domain, we only get the information in the discreet points, but the information between two discreet points gets lost. See Figure 3.17.

It could happen that between one point of the discretization and the next the object has passed through the field of view, but the algorithm fails to detect it in both the first and second points of the discretization (see Figure 3.17). Thus, it would not be counted as observable. To overcome this problem, a comparison is made between the current discretization point and the next. In this comparison, the result is a vector that joins the current position of an observed object in an instant of the discreet time and the position in the next instant of the discreet time (see Figure 3.18). If the vector crosses the pyramid that determines the field of view (see Figure 3.18), the algorithm counts this object as observable and proceeds to the last step.

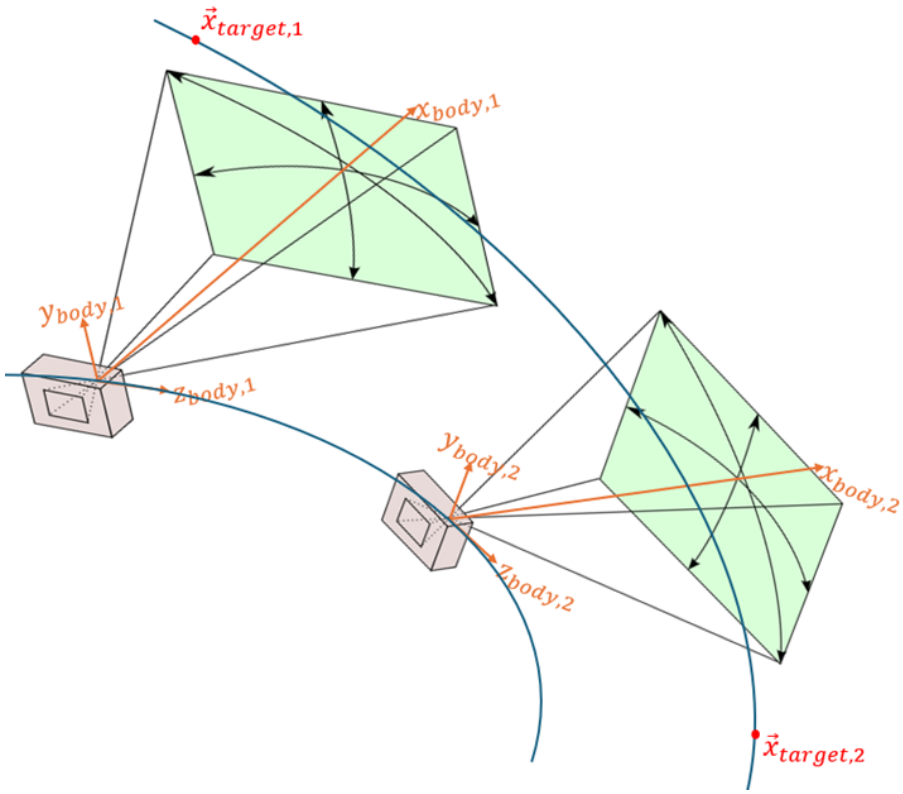


Figure 3.17 Representation of two discretization points of the orbit.

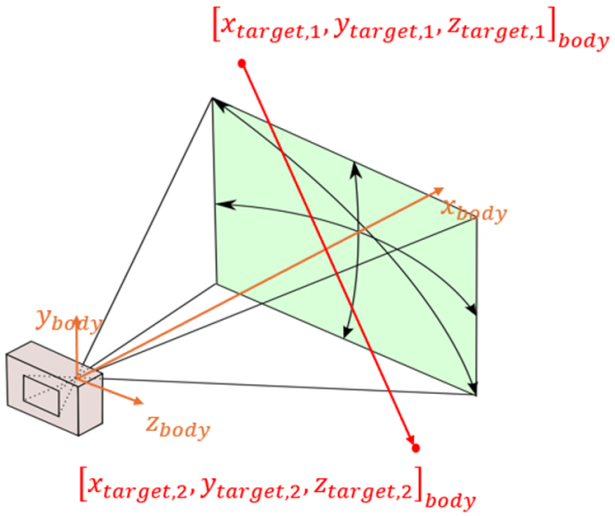


Figure 3.18 Vector joining two consecutive point of discretization

3.5.3 Third step

In this step, we consider the position of the Sun, the Earth and the observed satellite. With these data, we can determine if the observed satellite is in the shadow zone of the Earth. The shadow of the Earth has two parts: the umbra and the penumbra. See Figure 3.19.

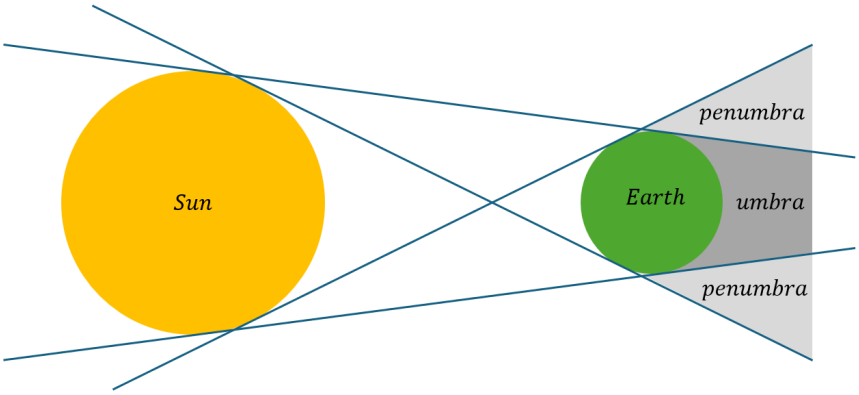


Figure 3.19 Shadow of the Earth: umbra and penumbra

First we will calculate the umbra cone that is tangent to both the Sun and Earth and has its axis in the vector that joins the center of the Sun and the Earth. α is the angle that the axis forms with the tangent (see Figure 3.20), and β is the angle between the axis and the internal tangent (see Figure 3.21.

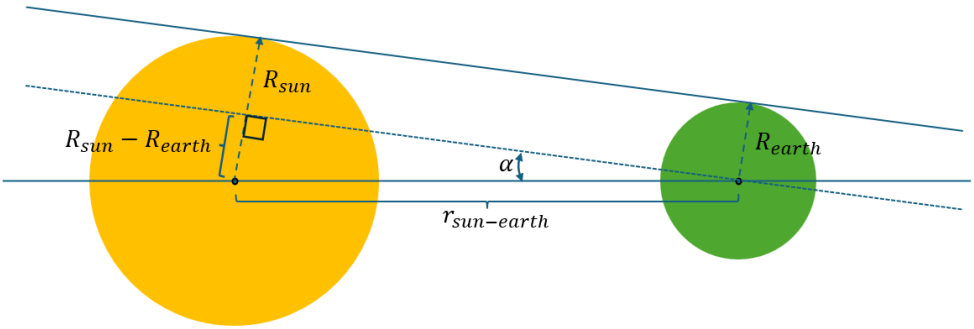


Figure 3.20 Calculation of α

$$\alpha = \arctan\left(\frac{R_{sun} - R_{earth}}{r_{sun-earth}}\right) = 0.2633°$$

$$\beta = \arctan\left(\frac{R_{sun} + R_{earth}}{r_{sun-earth}}\right) = 0.2682°$$

Since the angles α and β are very small, for this project the shadow will be considered to form a cylindrical dark zone with radius equal to the Earth's and the axis will be the vector that joins the center of the Sun and the center of the Earth. See Figure 3.22.

The next step is to know if the observed object is within the shadow or outside the shadow of the Earth. If it is outside, the object will be hit by the light somewhere, and that light can be captured by the observant

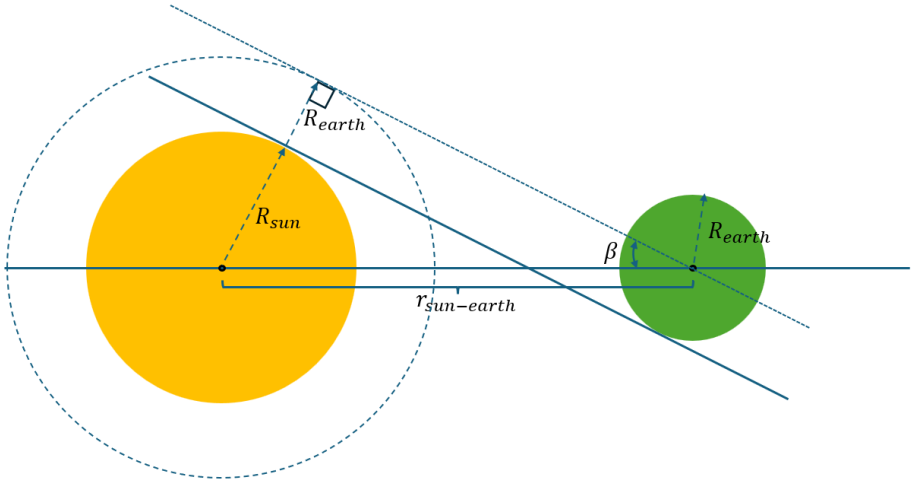


Figure 3.21 Calculation of β

Figure 3.22 Shadow of the Earth with simplifications

satellite. The calculation is as follows: first the vector $\mathbf{r}_{Sun-observed}$ is projected into $\mathbf{r}_{Sun-Earth}$, so the angle γ is obtained (see Figure 3.23). Then the height h is calculated with Equation 3.79.

$$h = \sin\gamma \cdot r_{Sun-observed} \tag{3.79}$$

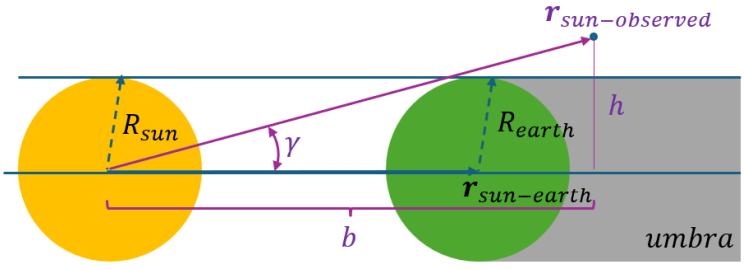


Figure 3.23 Computing h

Then three case scenarios have to be considered:

- $h > R_{Earth} \Longrightarrow$ The observed object is observable.
- $h < R_{Earth}$ and $b < r_{Sun-Earth} \Longrightarrow$ The observed object is observable.
- $h < R_{Earth}$ and $b > r_{Sun-Earth} \Longrightarrow$ The observed object is not observable, it is in the shadow zone.

3.5.4 Fourth step

This step is carried out if the object is in the light zone and it has been captured between two steps of the discretized propagation, so it has not fallen in either of the Field of View (FOV) of the camera in any of the two discretized points but has been captured in a time in between. In this step a recursion process starts: the propagation is repeated between the two discretized points with a smaller time step. If the object is not capture in the Field of View (FOV) of the camera in any of the discretized times but in between two, the process is repeated until it is captured.

If after these four steps the object stays observable, it can be concluded that the object is likely to be observed by the observant satellite.

4 State of the art

4.1 Orbital propagators

The development of orbital simulators has been an active research field in recent decades: efforts have been focused on delivering more and more accurate solutions that can help stakeholders plan their space missions and optimize their satellite resources as much as possible so that they can span their life. This kind of software is very flexible and offers many options and features. Some examples of these software are: Systems Tool Kit (STK), Orekit or General Mission Analysis Tool (GMAT). These simulators have the characteristic of being very rich and optimized for orbital calculations, but they lack the ability to perform simulations of camera views of the satellite, especially event-based cameras. In this way, custom software needs to be designed.

Systems Tool Kit (STK) is a software distributed by Ansys [39]. It allows modeling complex systems of ground, sea, air, and space platforms in three-dimensional simulation. However, this tool is limited by simulating event-based camera sensors, and they do not include modules to perform calculation on how many objects they are able to capture within a propagation.

The General Mission Analysis Tool (GMAT) is an open source project initially developed by NASA and the private sector [40]. It can model, optimize, and estimate spacecraft trajectories in Low Earth Orbit (LEO), lunar applications, interplanetary or deep space missions (see Figure 4.1. However, it lacks the ability to simulate cameras, including event-based cameras.

4.2 Event-based Camera Simulator

Additionally, this bachelor's thesis needed a simulation of the vision of the main sensor that is going to carry: an event-based camera. Currently, several event camera simulators are available, but none of these focus on space applications. Most of these tools work in a similar way: they require a video as an input and retrieve an event file as an output.

Examples of these are, among others, v2e [41], which is a Python software tool. It uses an interpolation to increase the frame rate of the video. Also, it includes characteristic features of event data: temporal noise or leak events. However, this simulator ignores the optical process of a camera, unlike PECS (Physical-based Event Camera Simulator) [42].

In this project, a simple event sensor simulator is going to be developed: the simulator is going to compare the corresponding pixels from one frame and the previous and based on the change in brightness, provide or not an on-event or off-event.

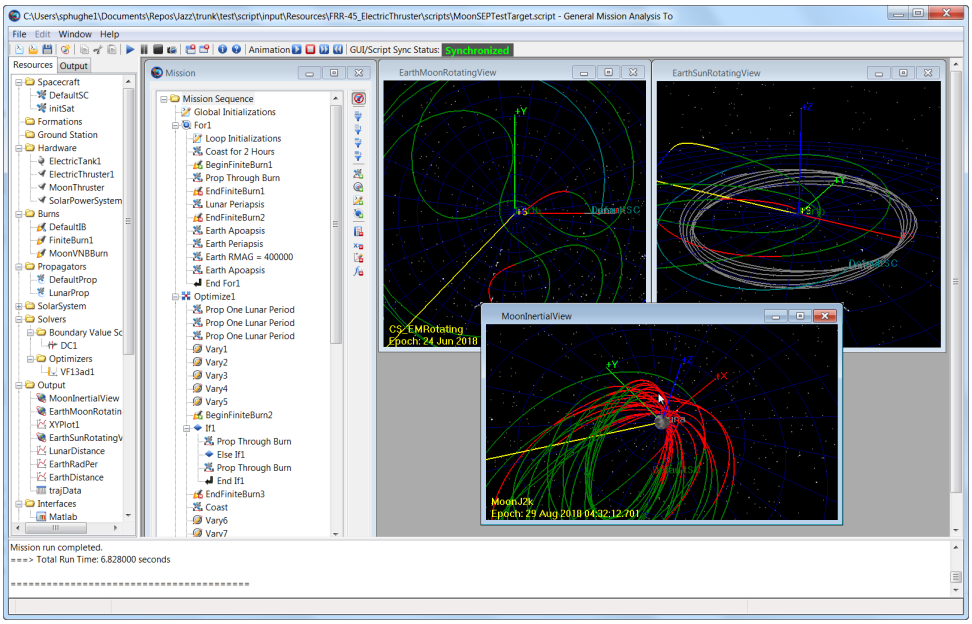


Figure 4.1 Cube-sat for a lunar mission. Source: GMAT Wiki

5 Tool Guide

5.1 Tool introduction

Once the theoretical background has been explained, the next step is to present the tool developed to achieve the objectives of this project.

The interested user can find a copy of the project in the following link: `https://github.com/alberto-sanchezcal/satsimulator`

When a user first compiles the app, the browser is opened. There, the graphical user interface is rendered (see Figure 5.1).

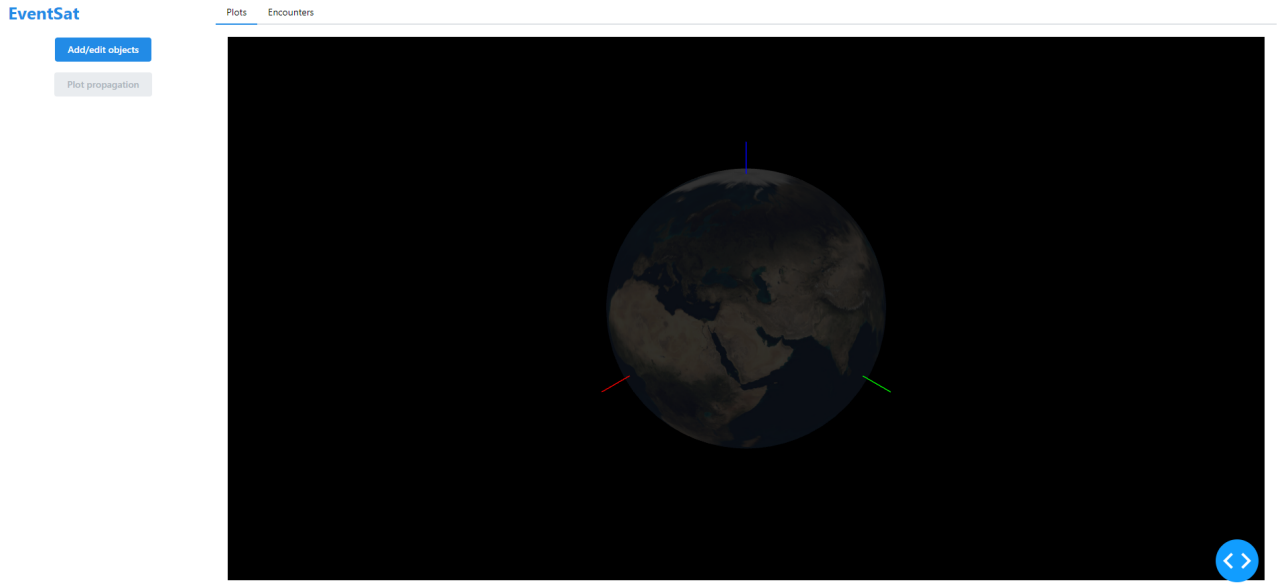


Figure 5.1 Home Page

If the user clicks on the "Add/edit objects" button, a modal is opened (see Figure 5.2). Here, the user can see the preprocessed satellite database. Several options are available: filter objects based on parameters such as Semi-major Axis (a), Eccentricity (e), Right Ascension of the Ascending Node (Ω), Argument of Periapsis (ω), True Anomaly (ν) or look for specific objects by their names or object ID.

In addition, multiple satellites can be selected by clicking on their respective checkbox, or all objects in the catalog can be selected by clicking on the check box in the header of the table. The user can quickly query the orbital parameters by hovering the mouse over the eye symbol of the respective satellite (see Figure 5.3). Furthermore, they can be deleted by clicking on the delete button next to the aye symbol.

The user can also create their own satellite. For this, they can click on the "Create object" button, or they can click on the upload symbol button.

When the button "Create object" is pressed, another modal is opened (see Figure 5.4). In this modal the user can fill out a form. In this form, the user is asked to create a name, to introduce the classical orbital elements that are going to characterize the orbit and initial position of the object. Furthermore, since this

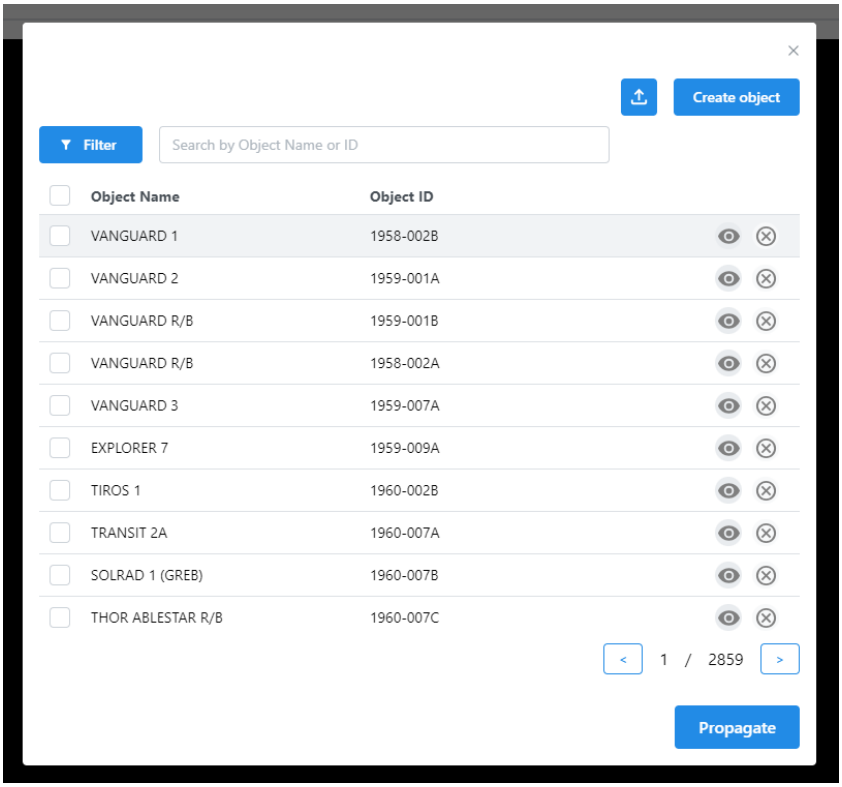


Figure 5.2 Modal that is opened when the "Add/edit objects" button is clicked

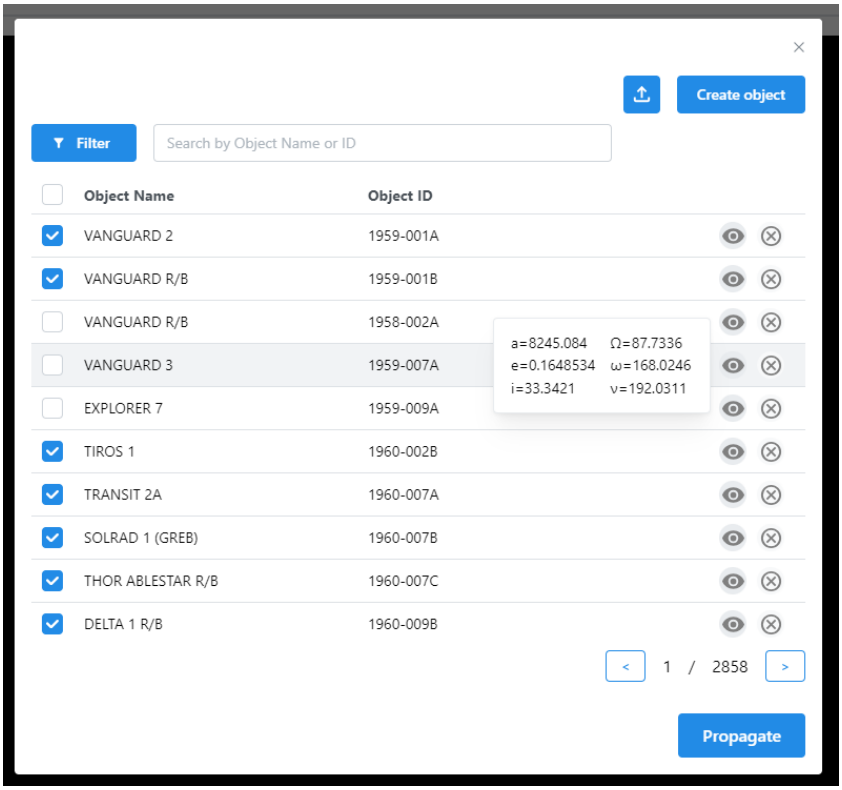


Figure 5.3 Several satellites have been selected, VANGUARD 1 satellite has been deleted from the database and the mouse is over the eye button

created satellite is supposed to mount an event-sensor camera, some parameters to model the view are required. The initial attitude of the satellite is defined by the quaternion nomenclature: the rotation axis which can be not unit vector and the rotation angle. All the fields have to be filled in the units indicated; otherwise the propagator will not work as expected. The tool rejects the "Save" action if any of the fields are not completed or if the format is not desirable.

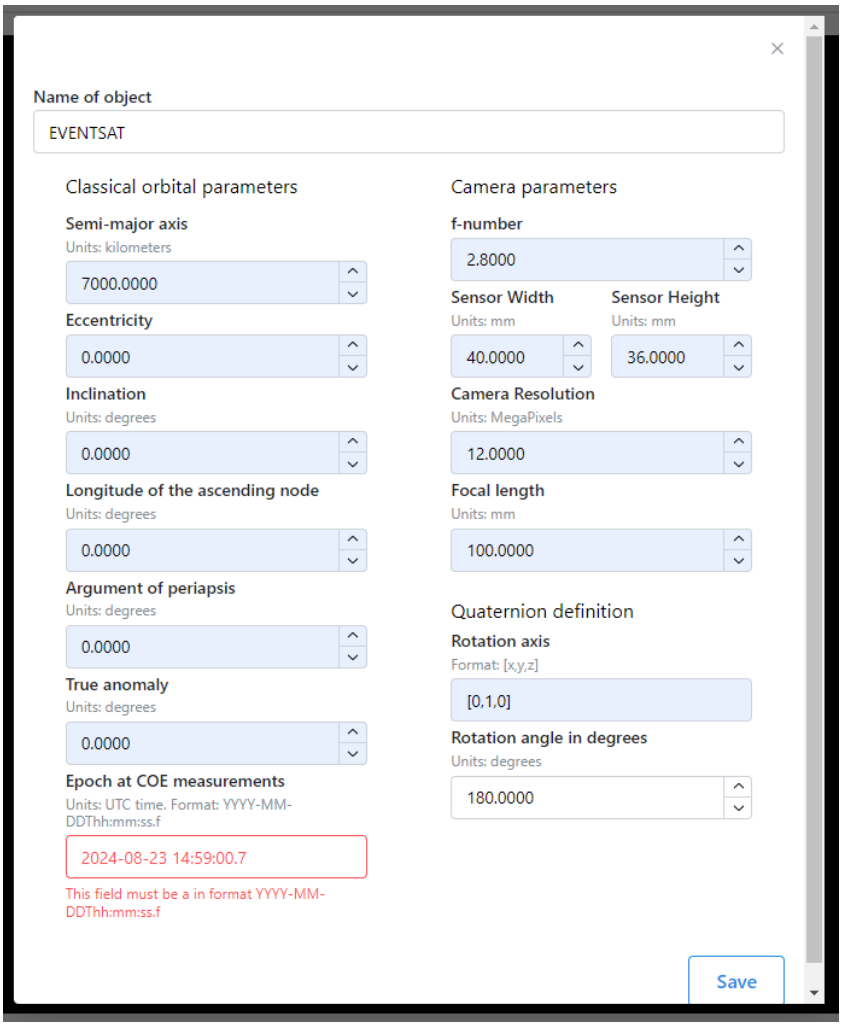


Figure 5.4 The object cannot be saved in this case. The epoch has not been provided in the desirable format: YYYY-MM-DDThh:mm:ss.f ("T" is missing)

Another way of creating an object is by clicking the upload symbol button. A modal appears (see Figure 5.5) and the user will find a box to drag a JSON file with the created satellites they wish to upload to the platform. In addition, some instructions on the JSON file format are provided.

Once the object has been created, the user can see that a new button has been activated in the object row, an edit button. If the user wanted to edit the object after saving it, they have the option to do so. This button is not activated for the rest of the objects in the catalog, just for the objects crafted by users.

Now, if the "Propagate" button is clicked in the right bottom part of the initial modal (see Figure 5.2), a new modal is opened (see Figure 5.6). The user is asked to select the propagator they prefer to use, the initial epoch of the propagation, and the propagation time.

Once the propagation is done, the user has the possibility of clicking on the "Plot propagation" button or going to the Encounters tab.

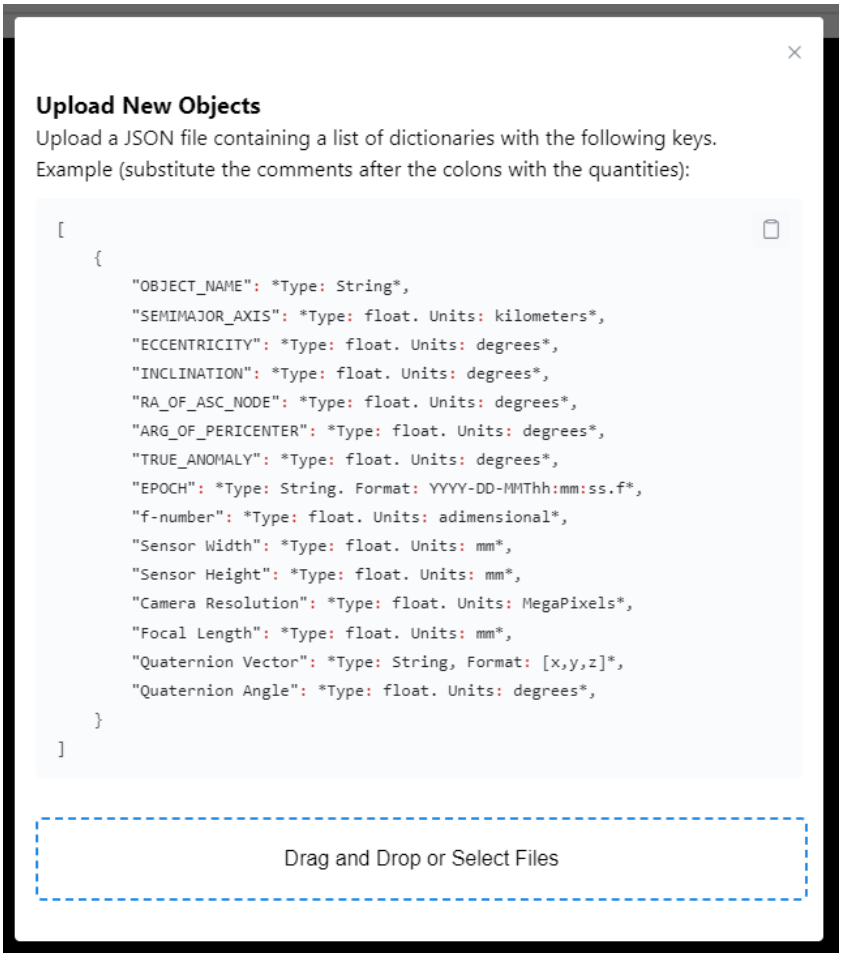


Figure 5.5 JSON file upload modal

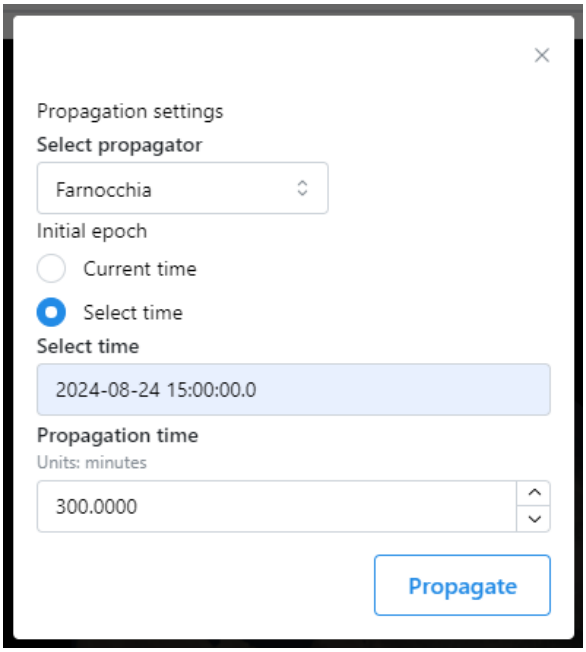


Figure 5.6 Propagation setting modal

When plotting the propagation, it can be highlighted that the position of the sun is also simulated along with the created observant satellites and the database objects. Also, Earth-centered Earth-fixed (ECEF) axis are included to help the user orient in the 3D environment. The simulation (see Figure 5.7) is animated in real time; the user can stop the animation, speed it up, go forward in time with the help of the slider, magnify the size of the objects for better visualization (see Figure 5.8), enable or disable the body and orbit axes of the created satellites. In the lower right part, the epoch is shown in every instant.

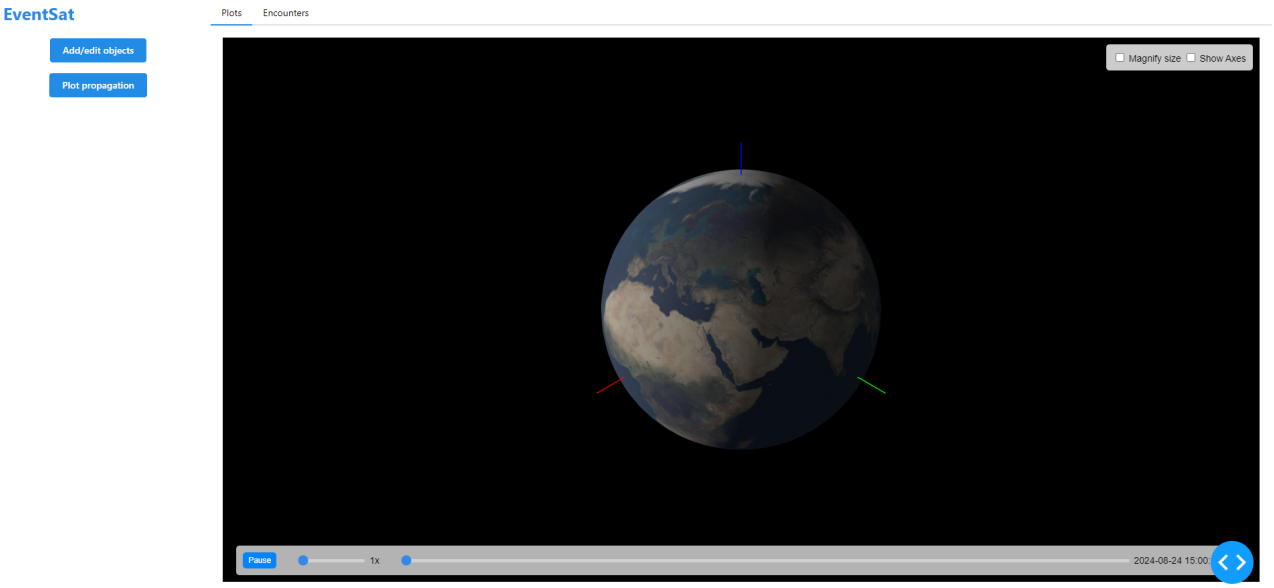


Figure 5.7 Simulation Plot

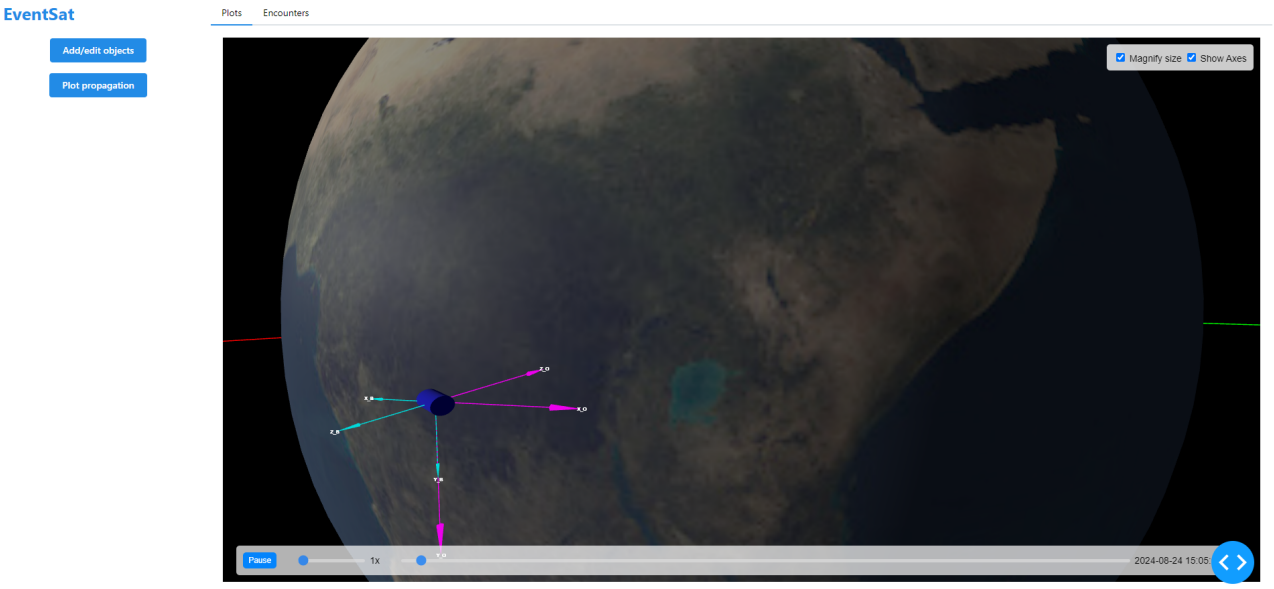


Figure 5.8 Simulation zoom with the activation of some functionalities

"Plot propagation" button is not recommended for long-time propagation or propagation of a large number of objects because the browser gets a lot of data in the client side and this slows down the rendering of the Web.

Now, going to the "Encounters" tab (see Figure 5.9), the user can look for information about the observability of the observant satellites with the rest of the selected objects that have been included in the

propagation. It is important to note that the created satellites cannot be observed among them, they can just see satellites from the catalog.

For this example, a fictitious object was created and introduced to the satellite database, and it is part of the catalog group; it does not belong to the created objects group. This dummy object also has big dimensions, so the observation was assured.

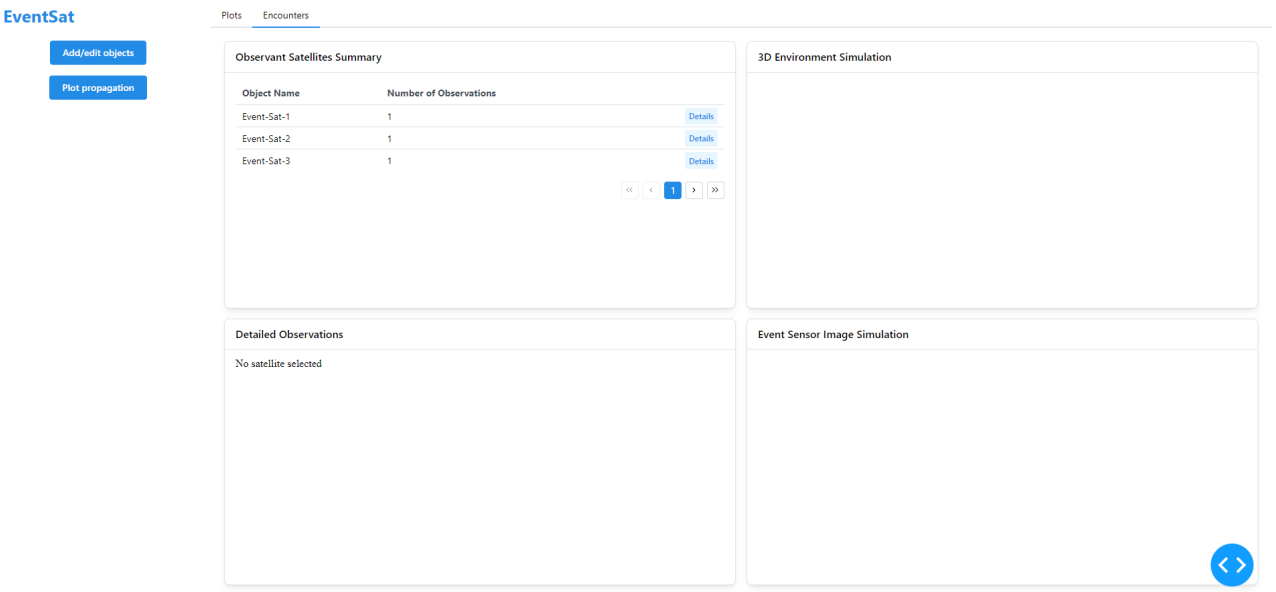


Figure 5.9 Encounters Tab

In this page, we can see "Observant Satellites Summary", a table in which all the created satellites appear no matter if they have gathered observations or not. They are in descendant order of the number of observations. Now, if the user clicks on the "Details" button, another table is loaded in the Detailed Observations box (see Figure 5.10).

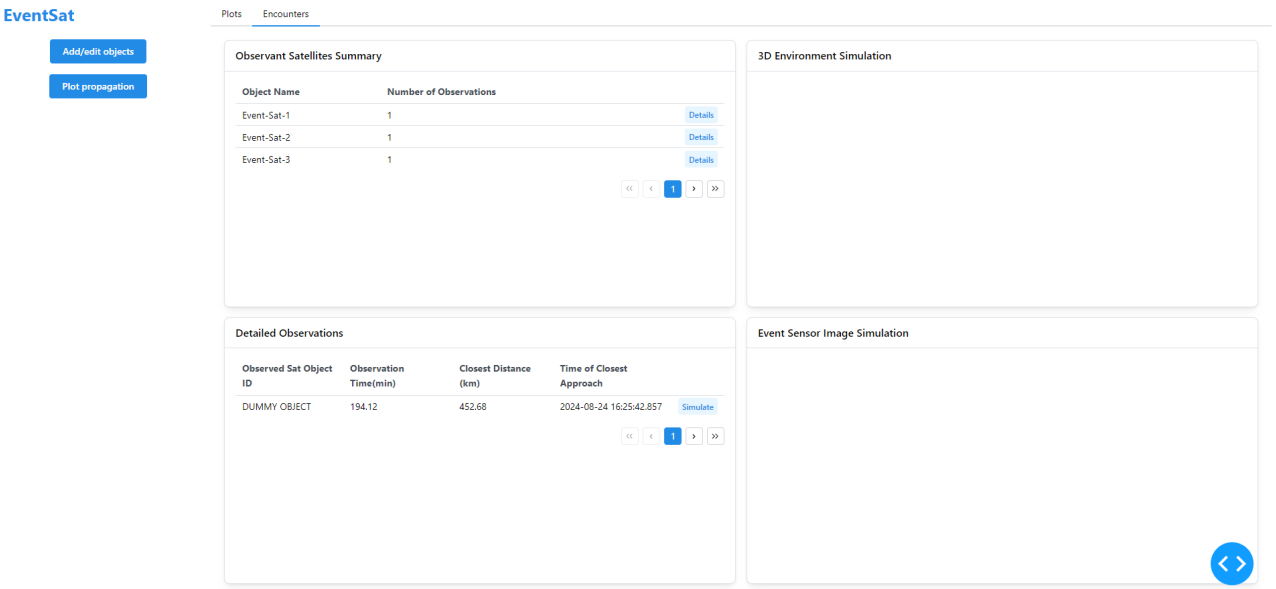


Figure 5.10 Encounters Tab

In the Detailed Observations box, we can see information about all of the observations that have happened. Some information provided in this table is the total approximated minutes of observation, the closest dis-

tance of the two satellites in which they are observable, and the time this closest approach occurs. If one wants to simulate the camera view, the "Simulate" button must be triggered.

The simulations comprehends a 3D Environment simulation and an Event Sensor Image Simulation (see Figure 5.11) for 1 second before and 1 second after the time of closest aproach. This simulation is rendered at 1,000 frames per second, which tries to emulate an event-based camera sensor. Since the event sensor image is downscaled in order to meet the resolution of the box assigned in the Graphical User Interface (GUI), the user can download the result and this will have the resolution of the sensor. Also, the evolution of the sight of the camera during the propagation can be watched by pressing the "Play" button, and other parts of the animation can be consulted by sliding the slider along the bar.

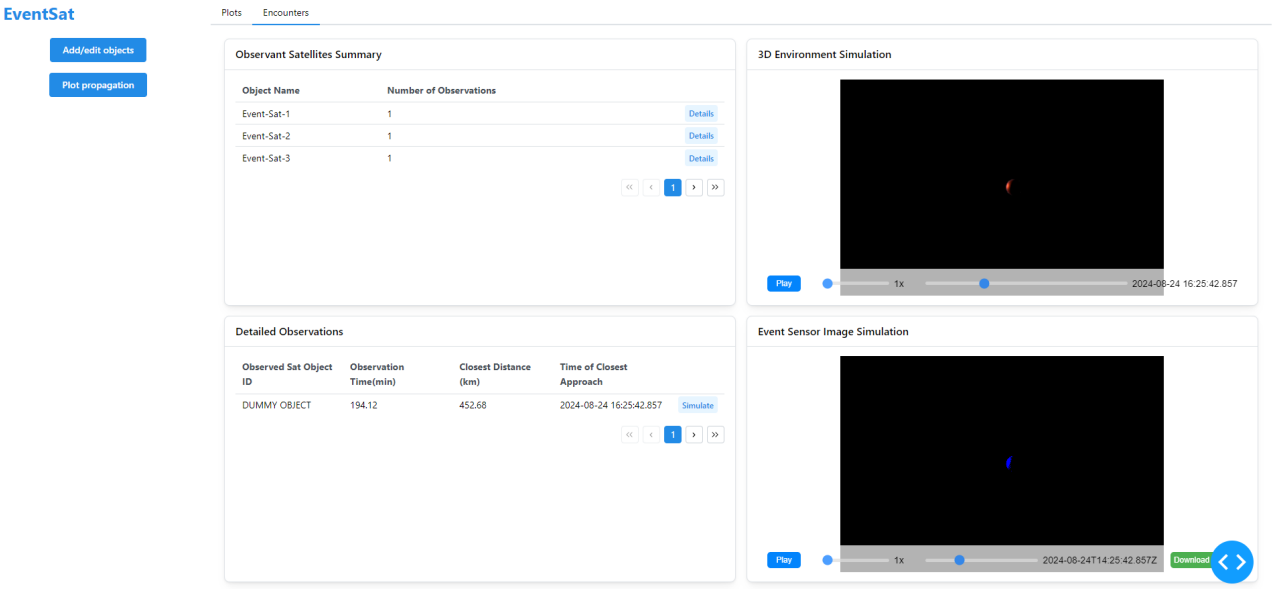


Figure 5.11 Encounters Tab

5.2 Technologies involved and project structure

The project is written in Python and JavaScript, employing a client-server architecture, to deliver a comprehensive and interactive experience. Python serves as the back-end, handling the logic of the Graphical User Interface (GUI) and the computational tasks such as orbital propagation and data analysis. JavaScript, specifically within a web-based framework, is used for the front-end to create interactive visualizations of the simulation results.

Python was chosen as the main programming language because of the growing community that supports the language. Also, it is a very easy to write and read language, thanks to its dynamic nature, simplifying the development process, and making it easier to rapid prototyping and iterative improvements. However, this contrasts with the lower performance compared to statically typed compiled languages and the increased difficulty of debugging the code.

Poliastro [43] is selected as the core library for orbital mechanics calculations due to its specialized functionality and ease of integration within the Python ecosystem. Other libraries on which Poliastro relies are Astropy [44] for coordinate frame transformations and time handling, and Numba for performance optimization.

Python is an interpreted language, and because of this nature, some mathematical operations are slow compared to compiled languages. In computationally heavy code, Numpy helps overcome this problem, because it implements operations in low-level languages like C/C++ or Fortran [45], which are faster than

pure Python. A key reason for this increase in speed is that NumPy works with homogeneous, multi-dimensional arrays, which are stored more efficiently in memory and allow for vectorized operations. This avoids the overhead of Python dynamic typing and the associated type-checking and dispatching that occur in standard Python loops, which are particularly slow when dealing with numerical calculations [46].

For the Graphical User Interface (GUI), the choice is a technology based on the Web. In this approach, the server primarily hosts the logic, and the client hosts the user interface. The reasons for this choice are varied. One of them is that the computing capabilities can be increased by upgrading the web server in which the calculations are performed to meet future demands. In this way, the web application can increase in capabilities, and it will not be constrained by the limits of the personal computer. Another perk is that it can be accessed through multiple locations and devices without the need to install the device memory. One of the main disadvantages of web applications is that these applications must be connected to a server through the Internet to perform data transfer. Thanks to current internet high speeds, this limitation is mitigated, to the point in most cases the data transfer is barely noticiable, and the other perks can compensate the cases in which it could be.

In this sense, the library chosen for the Graphical User Interface (GUI) is Dash [47]. It is a low-code framework for building applications in Python. It is open source and is mainly used for data applications. Due to its widespread use and suitability for web application development, it has been selected as the primary tool for creating the project's graphical interface.

The project written in Dash has a layer of styles to make it more attractive visually. This layer is Dash Mantine Components, a component library inspired in the Mantine React-library. React is a JavaScript user interface library developed by Facebook, which follows a component-based architecture. Mantine is a user interface component library that allows faster developments and is built on top of React.

To visualize orbital trajectories and simulate encounters, the project uses Three.js [48], a powerful JavaScript library to create and display animated 3D computer graphics in a web browser. It was chosen for its diverse features, large community support, and integration with the web-based front-end.

The project is structured in a main script *app.py* and four folders: assets, database, modules, and templates (see Figure 5.12).

5.3 app.py

This script is split into 3 main parts: the global variable declaration, another that loads the database into a global variable, and the last one that initializes the app and contains the logic of the app.

The block which loads the database makes queries to the functions defined in the data.py file (this file is inside the modules folder). In the initialization and logic part of this file, the app layout is loaded, and then a collection of callbacks follows. Callbacks are pieces of code or functions that are activated in response to a change in a specific input. It allows the interaction between the user interface and the underlying code, allowing to update the information screened in the app.

A callback is composed of inputs, outputs, and states. The inputs are components that the user interacts with, for example buttons, and trigger a callback. The outputs are the components that are updated when a callback is triggered. States are components that serve to pass data or information to the callback, but do not trigger the callback directly. A callback can be activated when one of their inputs is triggered but can also be triggered because that input is the output of another callback, forming a cascade of callbacks. For example:

```
@app.callback(
  Output("trajectory_data", "data"),
  State("propagation_id", "data"),
  Input("plot_propagation_button", "n_clicks"),
  prevent_initial_call=True,
)
def update_visualization(propagation_id, plot_propagation_button):
```

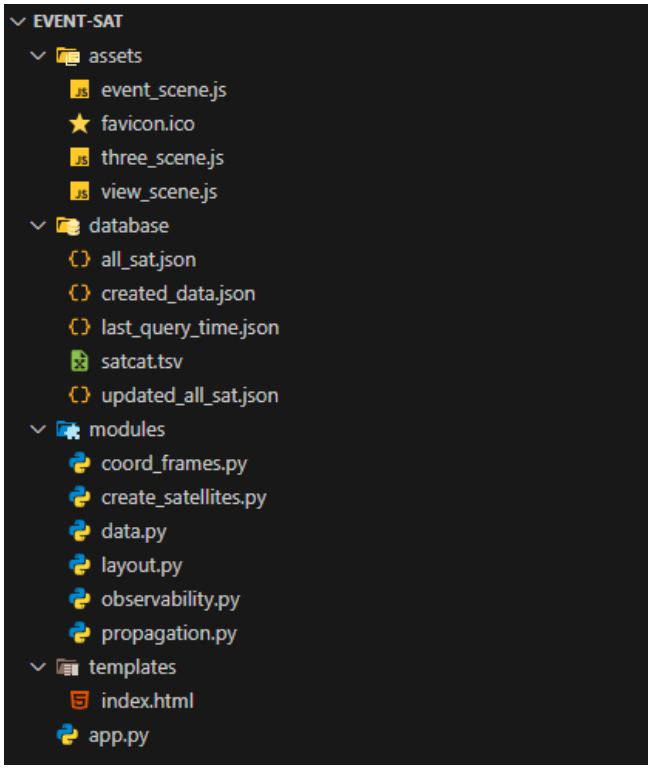


Figure 5.12 Project code structure

```
8   if propagation_id is None:
9     return None
10
11  data = propagated_data_store[propagation_id]
12  trajectory_data = data["trajectory_data"]
13
14  return trajectory_data
```

This callback takes as an input the clicks of the "Plot propagation" button. If this is clicked, the callback is activated and takes the `propagation_id`, which is a store component in the code. It saves the information about the last propagation that has been made. The callback then retrieves the necessary data from the global variable `propagated_data_store` and returns the result to the store component `trajectory_data`, which changes. As the latter storage component changes, it triggers another callback that has this component changes as trigger. In this case, it is a callback that renders the desired simulation of the trajectory.

A description of the functionality of this app.py file is shown in the flow chart in Figure 5.13:

5.4 Modules

In this section a brief explanation about the different modules is provided.

5.4.1 layout.py

This module provides the different Mantine Dash components to the Graphical User Interface (GUI) of the app. See Figure 5.14, where the structure of this layout is depicted.

At the beginning of this module, several `dcc.Store` components are declared. They store information on the client side of the app. Some of them act as triggers and do not store important information, some others store information that results from the interaction of the user and computations within the app.

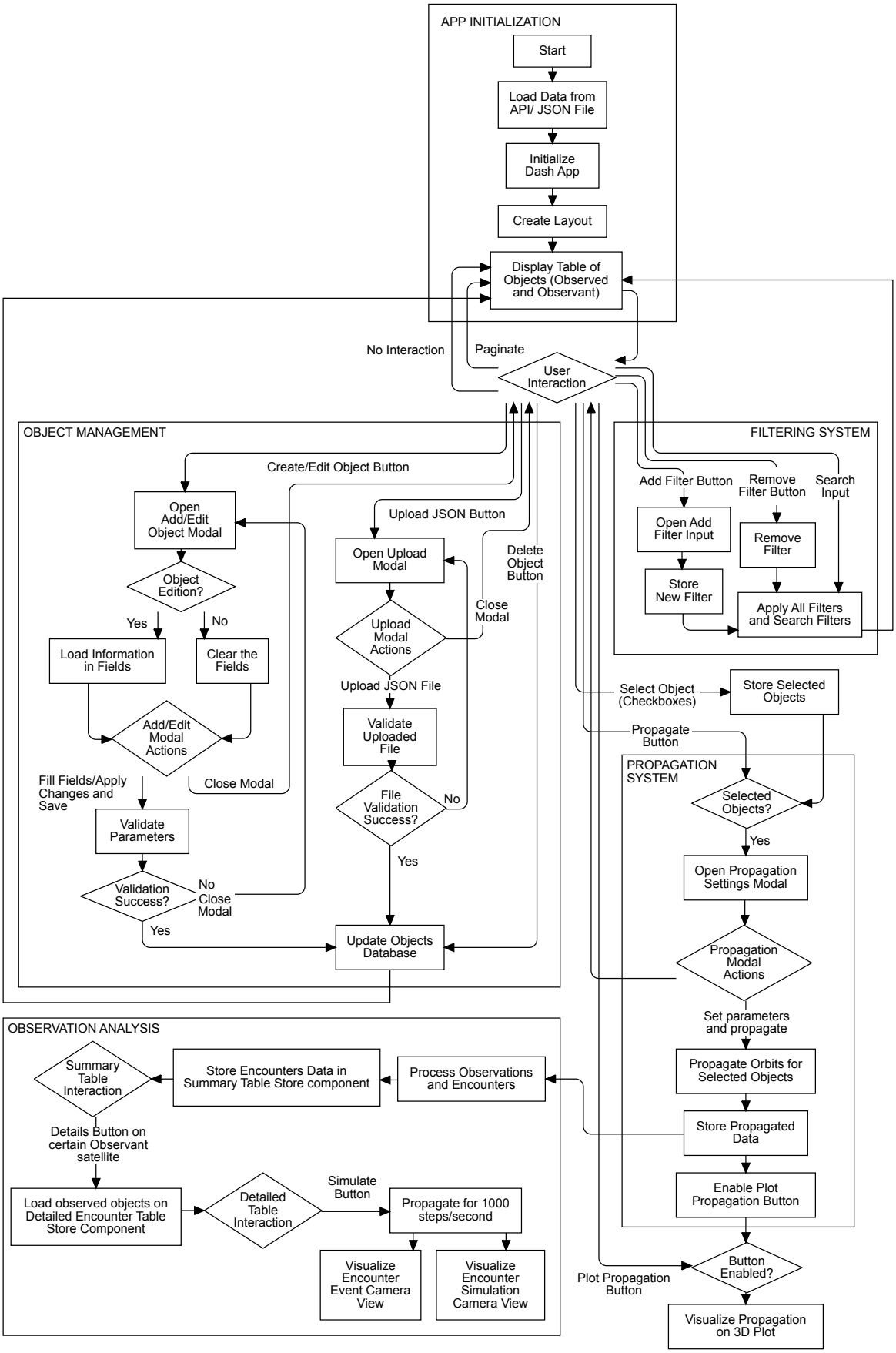


Figure 5.13 app.py flow chart diagram

`trajectory-data` and `trajectory-data-simulation` are not the same. While the first stores the information about the trajectories of all the checked objects (when the `plot_propagation_button` is triggered), the second is specifically designed to hold trajectory data only for the observant satellite and the observed object relevant to a particular encounter simulation, triggered by the "Simulate" button in the "Encounters" tab. This separation is a deliberate design choice to optimize client-side performance. This is done in this way because the `JavaScript initViewScene` and `initEventScene` functions requires data in the client side. By limiting the data volume passed to these functions, especially when dealing with extensive propagation datasets, the application avoids bottlenecks and ensures a smoother user experience.

The Store components declaration is followed by the Grid Layout. In this Layout we can see 2 columns: the left column contains the main title and some buttons, the right columns has a two-tab section, "Plot" and "Encounters". The "Plot" tab has a 3D visualization container, while the "Encounters" tab contains 4 cards: "Observant Satellites Summary", "Detailed Observations", "3D Environment Simulation", and "Event Sensor Image Simulation" (see Figure 5.14).

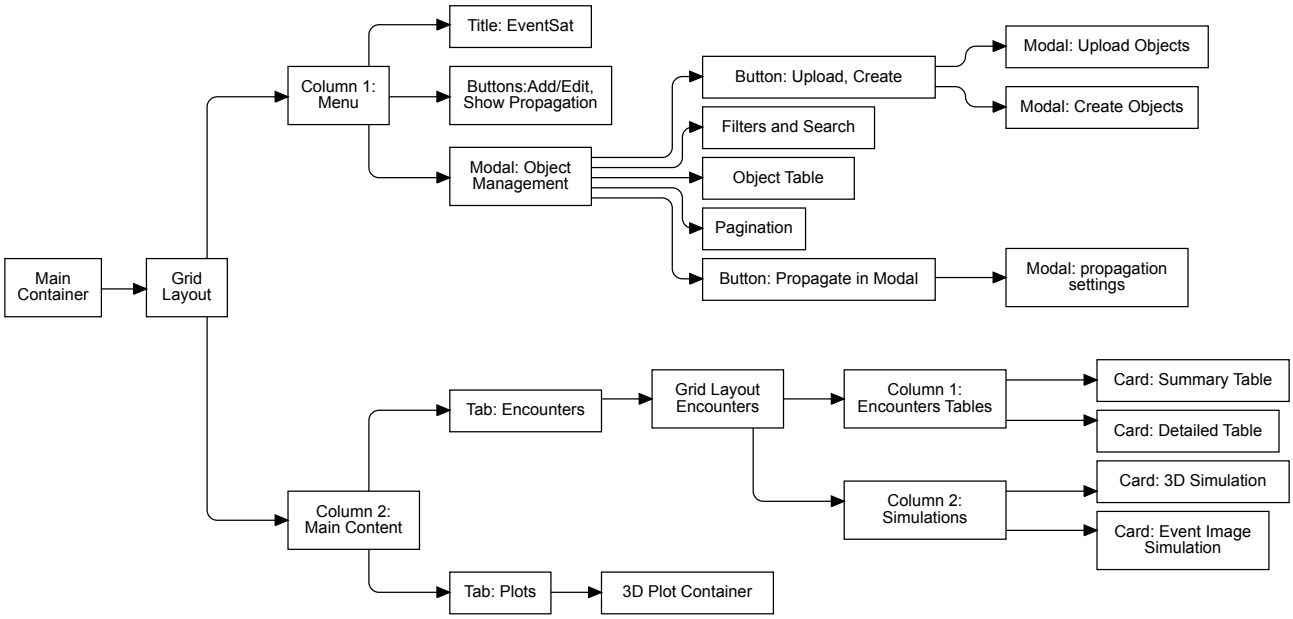


Figure 5.14 layout.py flow chart diagram

5.4.2 data.py

This module is used at the beginning of the `app.py` script. Its main mission is to gather the information from the space-track.org database and from planet4589.org. The functionality of this module is explained in the flow chart in Figure 5.15

As an explanation for the processing of the GCAT data, the value of shape in the GCAT is a string that can be the combination of several basic geometries. Performing an analysis, we find that there are a total of 365 unique geometries: *Disk + Ant*, *Sphere + 12 Ant*, or *Cyl + 4 Pan + Ant*, among others. It is impossible to model in 3D all of them so this script filters these geometries into boxes, spheres, cylinders, cones, and domes. These are the main geometry of the space objects they refer to, which usually is the first word of the string. Additionally, a total of 57 geometries cannot be classified into the previous main geometries, so they are assigned a default geometry: sphere.

When processing the GCAT data, dimension values are assigned to the object if it is not provided by GCAT. This value is 10 centimeters for objects classified as debris and 1 meter for objects that are not debris. In

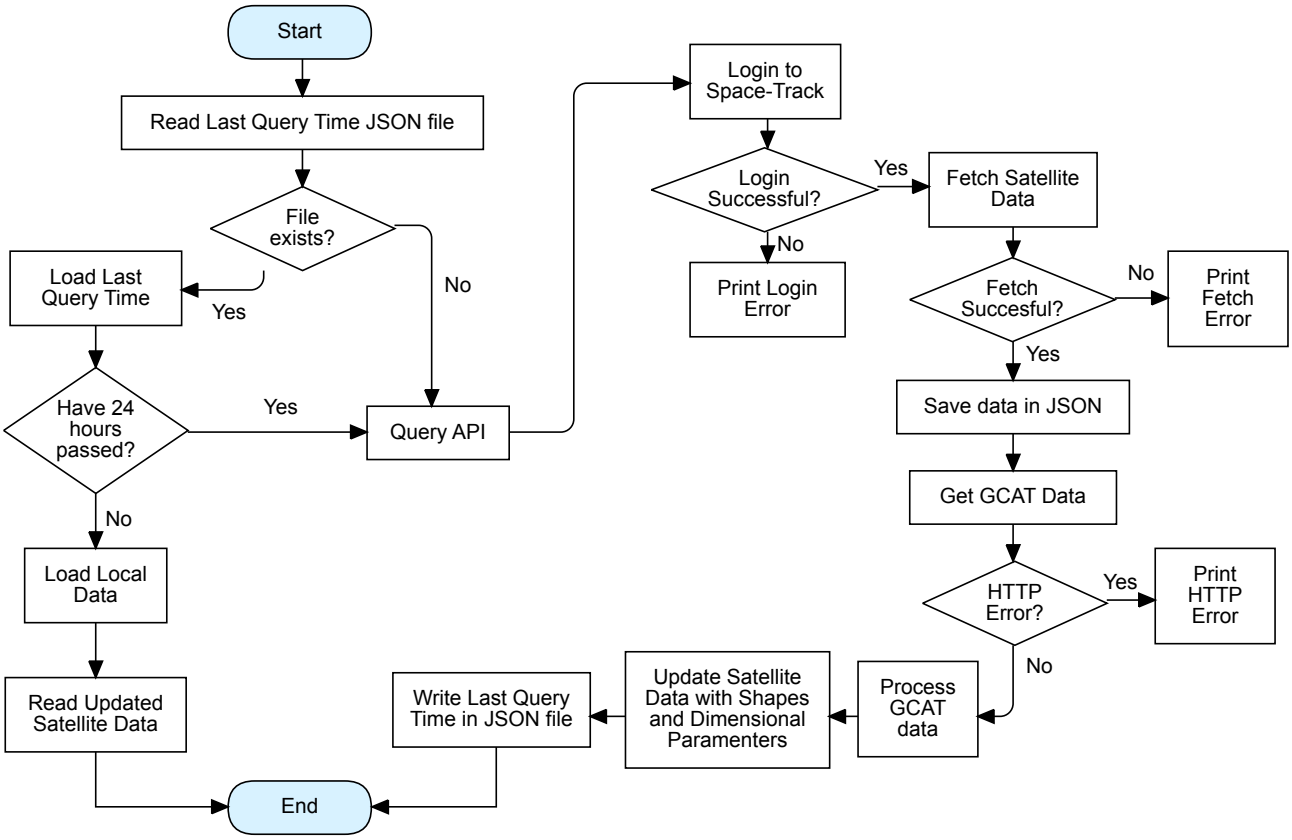


Figure 5.15 data.py flow chart diagram

addition, it computes the true anomaly of the position of the satellite in the orbit at the measurement epoch. This is because the data provided by the Star-Track does not include true anomaly but mean anomaly.

5.4.3 propagation.py

In this module there is a function called `propagate`, which takes as arguments the orbit, the time discretization, and the method chosen by the user. This module is called in the callback in the callback `propagate_selected_orbits`, which takes the first steps of the propagation. First, the initial values of the radius and velocity from the satellite are calculated, and with these two, the propagation occurs with Farnocchia's, Cowell's (with or without perturbations), Danby's, Pimienta's, Vallado's or SGP4's propagator. In Figure 5.16 the reader can see how the propagation system works.

5.4.4 coord_frames.py

When the the satellite is created by the user, the orbit coordinate system and the body coordinate system are computed, as can be seen in Figure 5.16. The latter defines the attitude of the satellite in the orbit, as was commented on in Section 3.2. This is useful when the algorithm for calculating the observability is called.

5.4.5 observability.py

This module is called whenever a propagation is done, see Figure 5.13. It follows the instructions given in Section 3.5. The instructions are summarized in the flow chart presented in Figure 5.17.

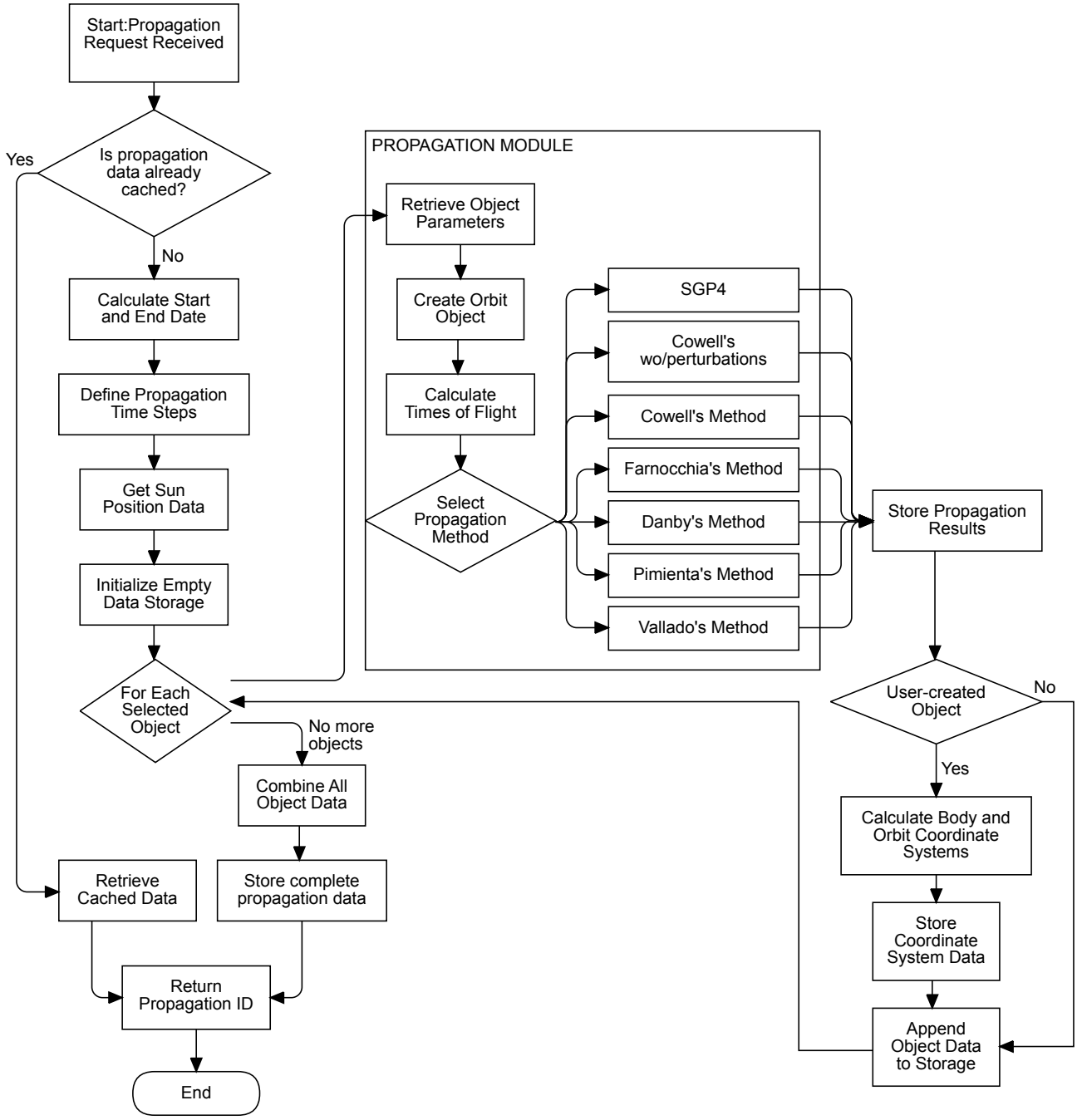


Figure 5.16 propagation.py flow chart diagram

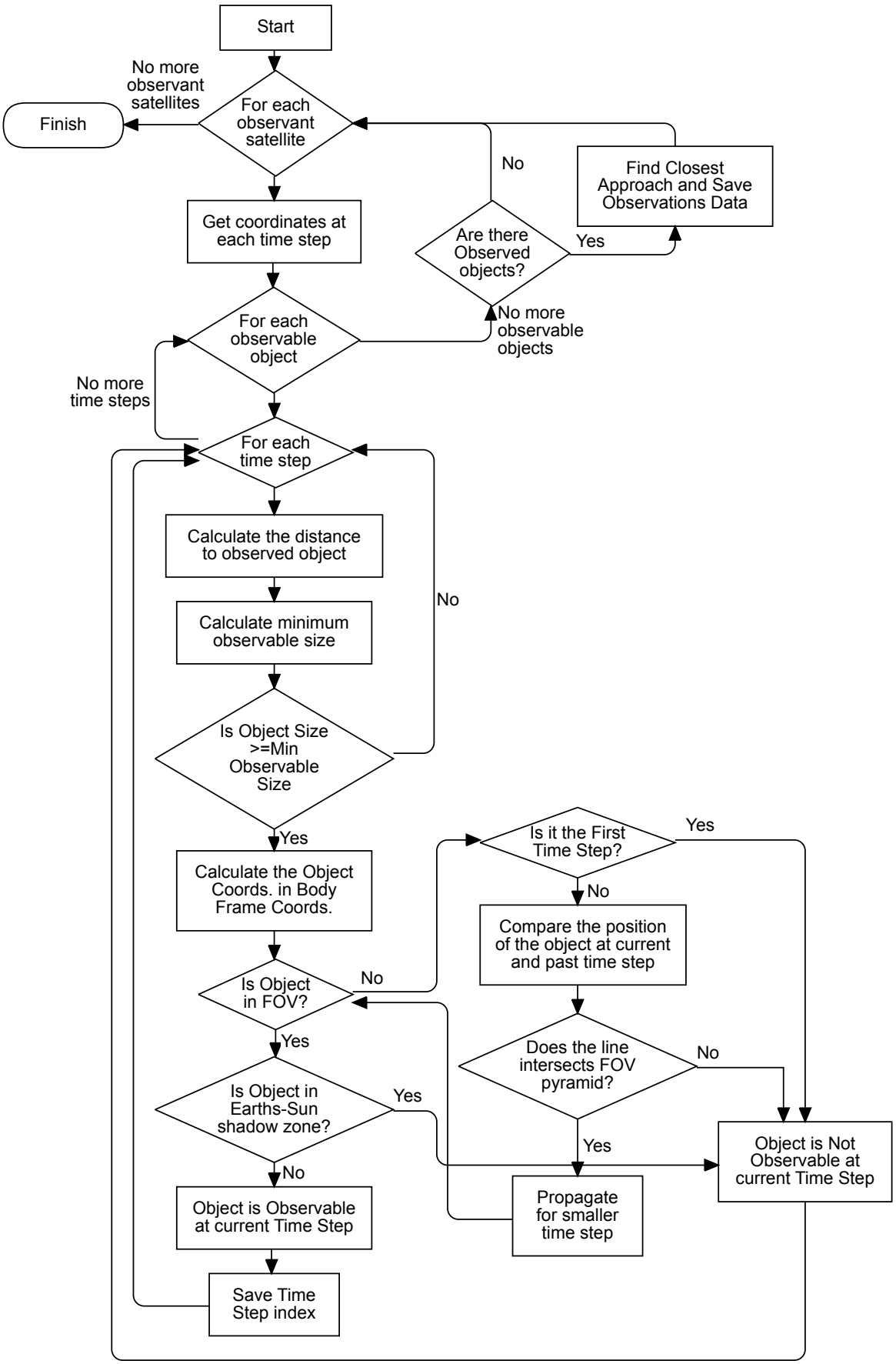


Figure 5.17 observability.py flow chart diagram

5.4.6 create_satellites.py

This Python code must be executed independently from the rest. It saves in a JSON file a set of random configurations for different satellites. This file can be loaded into the app later. It will be useful when analyzing a use case.

5.5 Assets

In this folder several JavaScript Three.js scripts are contained. three_scene.js contains the function initThreeScene which renders the 3D environment into the threejs-container, view_scene .js contains the function initViewScene which renders the 3D environment camera view into the 3d_camera_sight_sim container and event_scene.js contains the function initEventScene which renders the event camera vision simulation into the event_cam_image container.

initThreeScene includes the Earth, the Sun (as a light source), Earth-centered Earth-fixed (ECEF) axes, and the different objects with their different shapes, dimensions and a color code: red for debris, blue for payloads, orange for rocket bodies, green for unknown and white if none of the before is mentioned.

The simulation has the ability to simulate the movement of the objects along their corresponding trajectories and the sun. If the resolution in the time domain is low, the simulation is not smooth. To solve this issue, an interpolation for the coordinates was done. The simulator also contains controls for the animation: they allow users to play/pause the animation and adjust its speed. It also displays the current epoch of the animation.

Some actions that the user can do are: hovering over an object will highlight it and will show the trajectory, clicking on it will show the coordinates in the console, checking the Magnify checkbox will magnify the size of the objects and checking the Show Axes checkbox will show the orbit and body axis for the created satellites.

The differences of initViewScene are that in this the camera is set in the position of the satellite of the orbit. The pointing direction is equal to the direction of the x-body axis and the up direction of the camera is equal to the negative direction of the y-body axis. Furthermore, when this simulator is initialized, the camera is position in the closest-distance of observation to the observed object. In this code some functionalities of initThreeScene have not been included: Earth-centered Earth-fixed (ECEF) -axis, hover and click actions, the "Magnify" and "Show Axis" check boxes.

initEventScene works the same way as initViewScene but in this code a filter is applied. Furthermore, the scene is rendered at the resolution of the camera (and then is down sampled for fitting in the app container).

The filter is defined in the EventCameraShader, this is a first approach to the visualization of events. The frame rate is the same all the time, so the asynchronous characteristic of event cameras cannot be modeled.

For this approach 2 consecutive frames are compared, if there is a significant change in pixel brightness (calculated in the variable diff), more than the absolute value of the threshold, then that pixel change its color to red (off-event) or to blue (on-event).

Additionally, persistence is simulated with the decayRate variable, the lower, the longer that pixel stays activated.

Also, noise is emulated. noise is a random variable that depends on time and is multiplied by noise Strength, this is added to the diff variable. The noise slightly perturbs the change detection process, making it more likely to trigger "events" even in areas where the actual change might be below the threshold.

6 Validation

In this section, we will demonstrate the implementations work correctly. For this, we will compare some tests with already proved and existing software.

6.1 Orbital mechanics tests

With this test the position of the satellite will be checked. The tool chosen to compare with is General Mission Analysis Tool (GMAT), an open source software developed by NASA to perform space mission design and optimization optimization.

The object that will be used as an example is called STARLINK30104. This satellite presents the Classical Orbital Elements (COE) and epoch shown in 6.1:

Table 6.1 STARLINK30104 Classical Orbital Elements (COE)

Semimajor Axis (km)	6937.189
Eccentricity	0.00023600
Inclination (deg)	43.0050
Argument of Periapsis (deg)	301.2015
Right Ascension of the Ascending Node (deg)	319.1954
True Anomaly (deg)	58.8828
Epoch at measures	2024-09-05T13:17:21.016608

The propagation method that has been selected was Cowell. No perturbations were considered in either test, the simulator test and the GMAT test.

As the reader can see by Figures 6.1 and 6.2, the propagation is visually very similar.

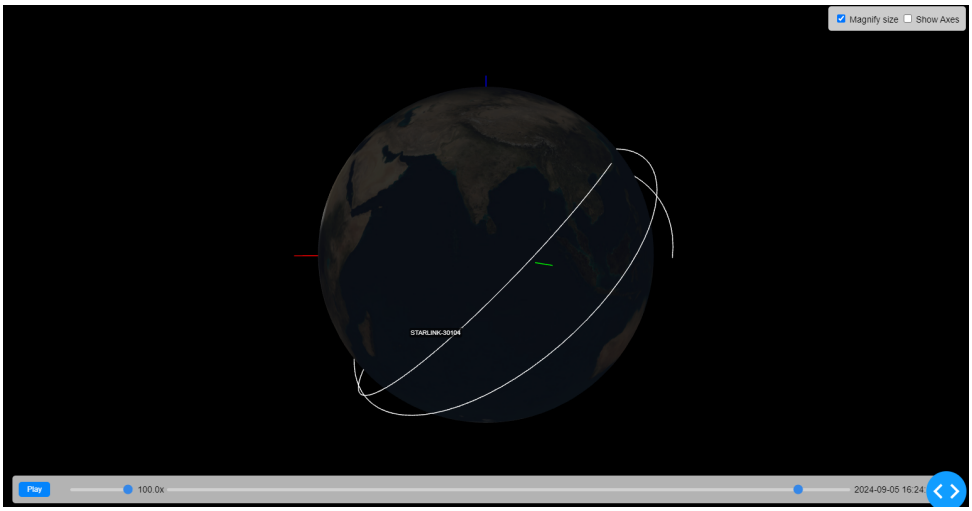


Figure 6.1 Starlink simulated in simulator

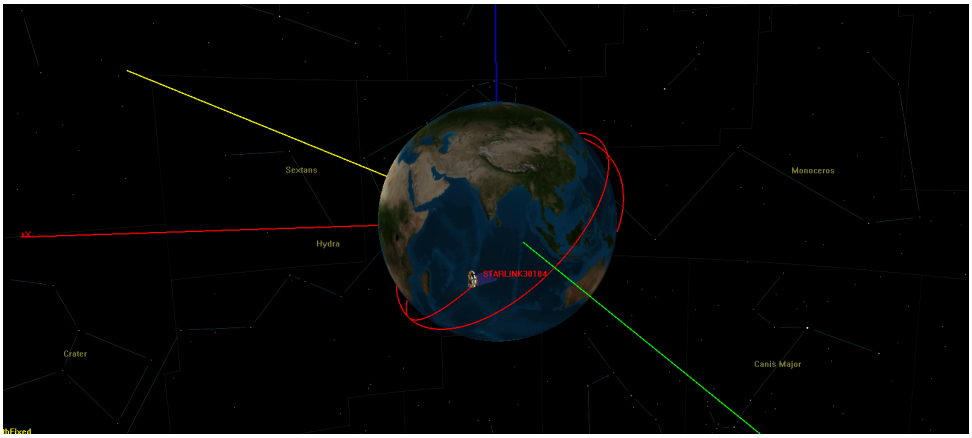
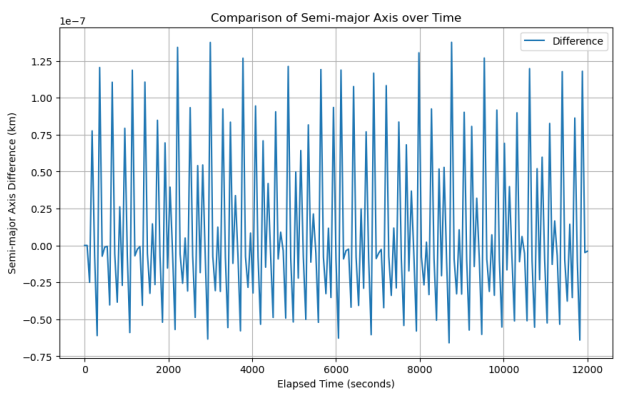


Figure 6.2 Starlink simulated in GMAT

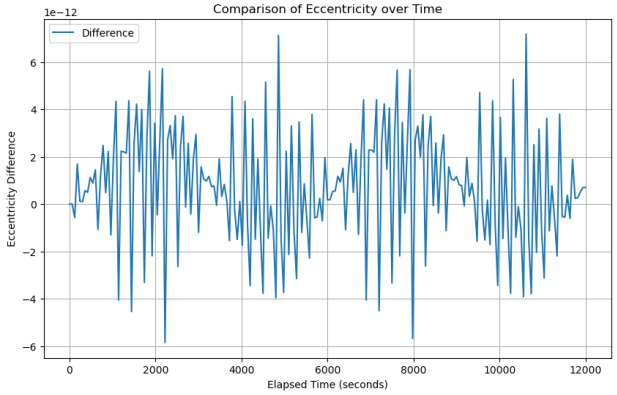
This data can be analyzed in detail. In the following graphics, Classical Orbital Elements (COE) have been compared for the orbit propagation of 200 minutes in GMAT and in the custom-made simulator. The propagation method has been the same for both cases (Cowell- PrinceDormand853).

The comparison shows that the simulation is accurate and correct: Semi-major Axis (a), Eccentricity (e), Inclination (i), Right Ascension of the Ascending Node (Ω) and Argument of Periapsis (ω) have variations in the order of magnitude of minus 13 to minus 6.

A remark should be made with respect to the graphic referencing the True Anomaly (ν): () frames in Poliastro are rotated 90º with respect to the () frames defined in GMAT. However, this discrepancy does not affect the accuracy of the simulation, since a conversion to Earth-centered Earth-fixed (ECEF) frames is made afterward and then the satellite gets positioned correctly, as can be seen by a visual comparison.



(a) Semi-major Axis Difference (km)

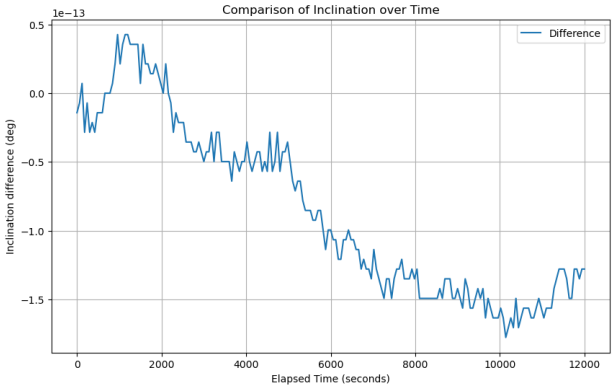


(b) Comparison of Eccentricity over Time

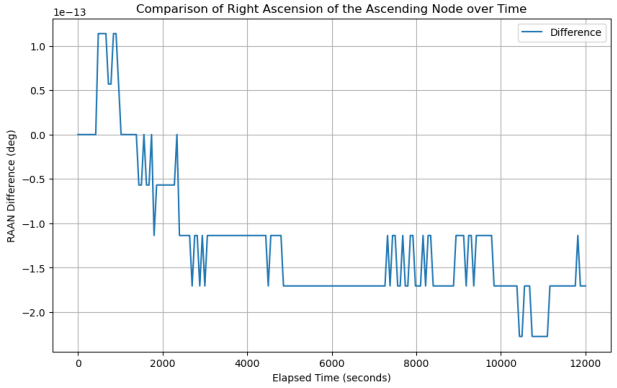
Figure 6.3 Semi-major Axis and Eccentricity Comparisons

6.2 Event camera simulator tests

For the verification of this part of the tool, a test has been developed to force an encounter: a dummy spherical object with exaggerated dimensions of 30 km diameter has been introduced in the satellite data catalog, with the orbital data presented in Table 6.2, and the observed satellite has been created with the orbital parameters shown in Table 6.3. These two objects have the characteristic of crossing each other in perpendicular trajectories, which ensures relative movement between them (see Figure 6.6).

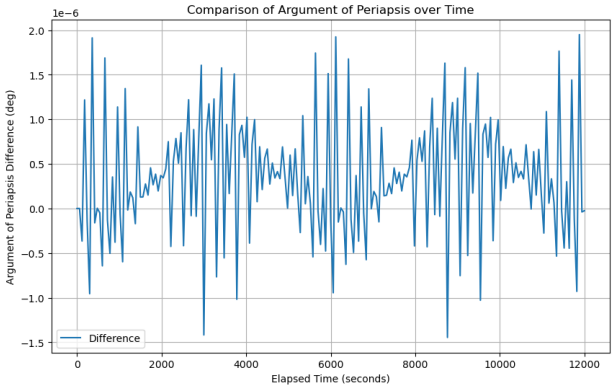


(a) Comparison of Inclination over Time

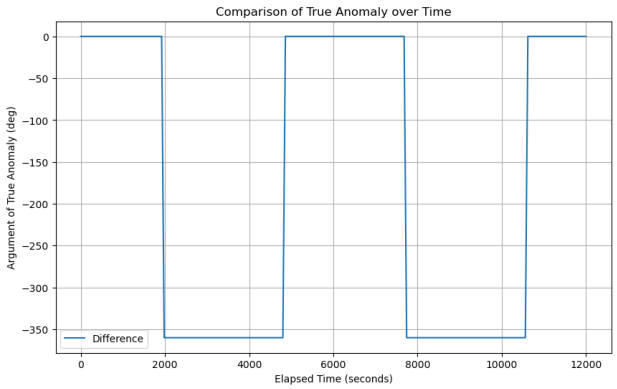


(b) Comparison of Right Ascension of the Ascending Node over Time

Figure 6.4 Inclination and RAAN Comparisons



(a) Comparison of Argument of Periapsis over Time



(b) Comparison of True Anomaly over Time

Figure 6.5 Argument of Periapsis and True Anomaly Comparisons

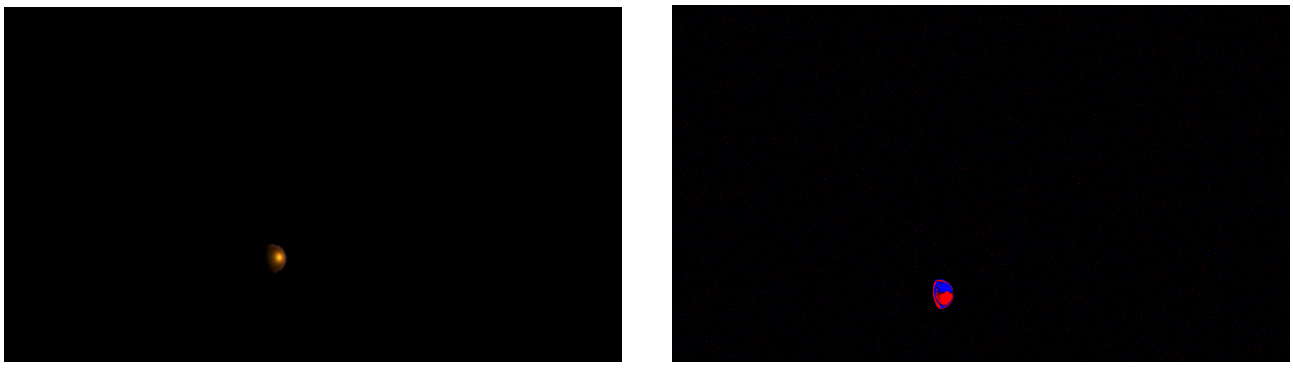
Table 6.2 Dummy object Classical Orbital Elements (COE)

Semimajor Axis (km)	7000
Eccentricity	0
Inclination (deg)	90
Argument of Periapsis (deg)	0
Right Ascension of the Ascending Node (deg)	319.19540
True Anomaly (deg)	179.5
Epoch at measures	2024-08-24T15:00:30.0

Table 6.3 Event Sat 1 Classical Orbital Elements (COE)

Semimajor Axis (km)	7000
Eccentricity	0
Inclination (deg)	0
Argument of Periapsis (deg)	0
Right Ascension of the Ascending Node (deg)	0
True Anomaly (deg)	170
Epoch at measures	2024-08-24T15:00:00.0

Figure 6.6 Three dimensional view of two EventSat's and a dummy object (orange sphere)



(a) Event-Sat Camera View

(b) Event Camera Simulation

Figure 6.7 Event-Sat Camera View and Event Camera Simulation

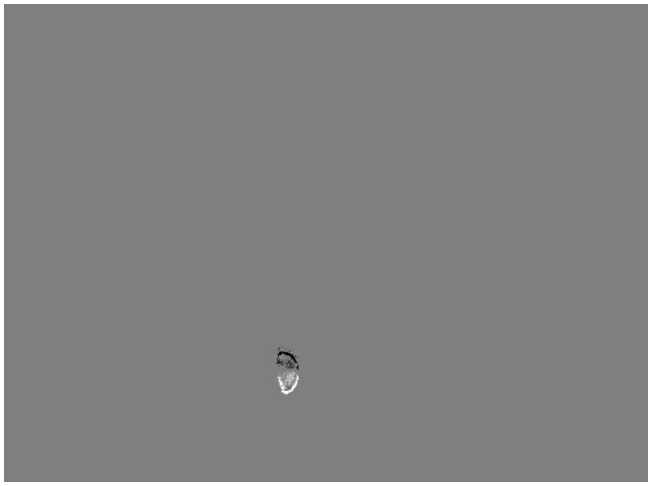


Figure 6.8 Event output using the tool v2e

Now we have a screen recording of the animation and using the tool v2e [41], we can get what a simulation of events could look like (see Figure 6.8. In this case, we can conclude that the Figures 6.7b and 6.8 look very similar.

7 Use Case

Now that the app has been validated and the users have been taught in how to use the app, we will focus on a practical case.

In order to do this, the Python script `create_satellites.py` will be used. This code will allow the user to create a JSON file that contains a set of modeled satellites. The file structure is a list of libraries. Each of these libraries will contain the necessary parameters for configuring a satellite. Then, this JSON file will be fed into the app system. The system will then calculate the observability of each of these satellites in the set with the rest of objects in space. The data output will be analyzed.

The `create_satellites.py` script consists of a declaration of thresholds. The script fills in the necessary data for every parameter in each of the libraries with random numbers within that threshold. However, the number of configurations cannot be infinite, so a smart decision about the threshold has to be made. For this, the distribution of objects will be analyzed depending on the inclination of the orbit and the height.

First, the distribution of objects in space is provided [2] in Figure 7.1.

	PL	PF	PD	PM	RB	RF	RD	RM	UI	Total
LEO	10136	5747	115	227	953	3274	40	579	83	21154
GEO	800	3	3	9	64	0	0	0	33	912
EGO	521	1	1	49	203	82	3	4	1889	2753
GTO	51	28	1	10	235	202	12	51	626	1216
NSO	282	0	0	1	96	0	0	2	38	419
MEO	77	0	4	49	25	42	1	4	415	617
LMO	83	138	5	46	246	590	22	215	955	2300
MGO	68	65	1	3	177	1968	4	0	1178	3464
HEO	29	13	0	1	54	113	0	0	1101	1311
Other	45	0	0	5	5	0	0	0	97	152
Total	12092	5995	130	400	2058	6271	82	855	6415	34298

Figure 7.1 Number of objects in space

As seen in Table 7.1, around 62% of the objects in space reside in the Low Earth Orbit (LEO). Now, the distribution of satellites in Low Earth Orbit (LEO) in a more graphical way can be summarized as in Figure 7.2.

The graphic 7.2 shows that the most populated orbits in Low Earth Orbit (LEO) are the ones situated in inclinations of around 95º and heights of 400-1400 km, 70º and 500-1000 km, 45-55º and 500-700 km. Definitely, polar orbits are the most congested places in the Low Earth Orbit (LEO) environment.

We will choose as a threshold for the inclination values between 40 and 100 degrees. For the height we will range from 400 km to 1000 km, in spite of being the altitudes 400-600 km the most common for CubeSats. In terms of eccentricity, it will be close to zero since cubesats usually occupy circular orbits. In terms of camera configurations, the sensor size is fixed, the f-number, and the focal length can vary between a set of [1.4, 2.8, 4.0, 5.6] and [100, 200, 300, 400, 500, 600], respectively.

Once the file is created, we will open the application and upload it. Additionally, we will apply thresholds to the database before checking objects, so we can reduce unnecessary computational load.

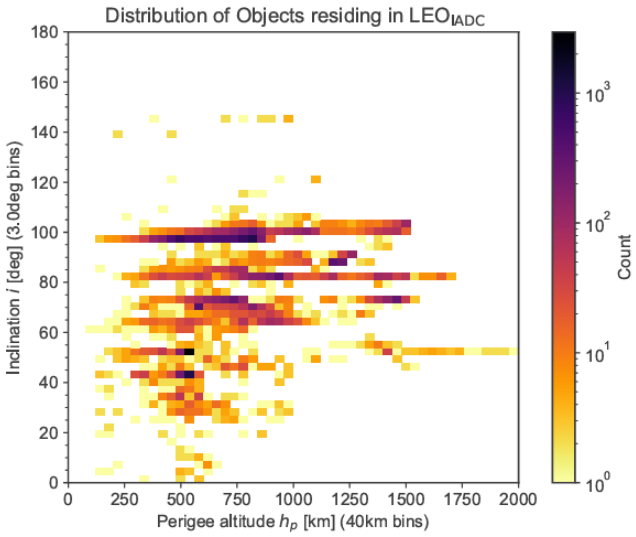


Figure 7.2 LEO distribution

The results for this set of satellites are shown in Figure 7.3. This outcome is obtained after a propagation of a trajectory of 30 minutes. The discretization used was with a resolution of 2 points in the orbit trajectory every minute of propagation time. This resolution is very poor. As commented in previous sections, the object can cross the field of view between two points of discretization. When this happens, the software takes the second point as the one in which an observation occurred, and then it initializes a process of recursion in which the application takes smaller steps between that two points, so it end up with a point in which the observation has occurred.

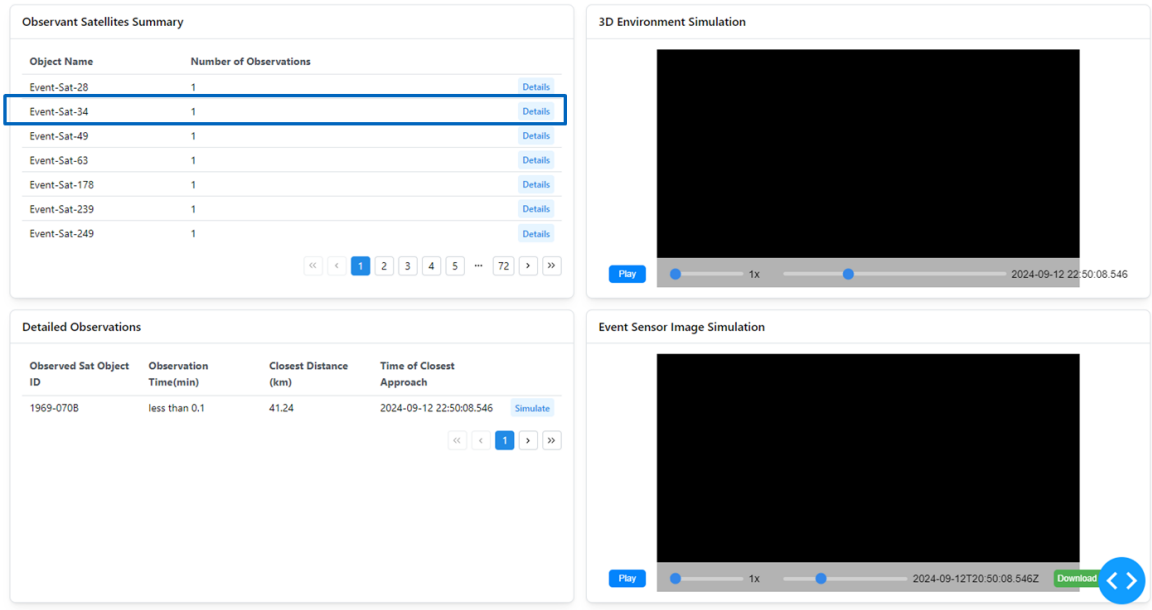


Figure 7.3 Results for the use case

Focusing on Event-Sat-34 one can simulate its view (see Figure 7.4) and notice a small set of blue pixels that get highlighted in Figure 7.5.

Figure 7.4 Event-Sat-34 view

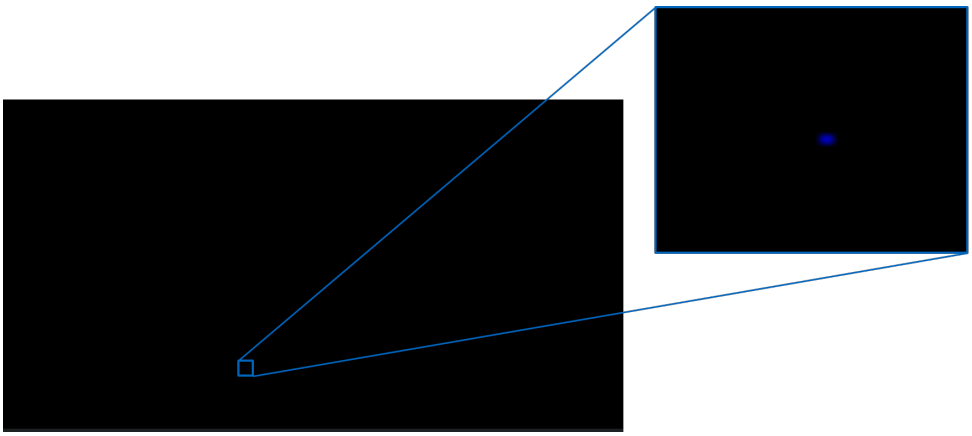


Figure 7.5 Event-Sat-34 view detail

8 Conclusions and future work

The idea of developing an event camera simulator has been a challenge: a broad theoretical background is involved because satellite missions require several engineering and scientific fields.

As commented in the Introduction section, no simulator counted with the ability to propagate orbits of space objects, calculate the observability between custom satellites, and simulate the output of an event-base sensor camera. Now, we can say that a solution that meets the characteristics mentioned earlier is available. However, the journey has not been easy and some complications and challenges have arisen.

The first challenge was getting familiar with web app development, especially when dealing with a large amount of data and the necessity of considering managing it on the server side or on the client side. Another dear was the long time of calculations, especially when trying to obtain results by brute force. A clever way of addressing this problem could be distributing parallel computing in the cloud so calculations can be achieved in less time. In this sense, the project could be complemented by adding a feature: a cloud of objects distributed as described in the statistical models commented on before. This cloud could help the simulator analyze its viability and the best orbit to position the satellite. Additionally, it could help the in the training of artificial intelligence that performs automated maneuvers for debris avoidance.

More enhancements can be applied. Ideas that have not been included in this version are: the possibility of creating an event format file with the simulation of the event camera; with this file it could be possible to get an estimation of the power budget needed. Or, the other way around, simulate how the result is affected by the power budget.

Additionally, during the project, several modules have suffered evolution from their first concepts. First, the visualization components were deployed in Plotly, but this library did not support camera positioning or orientation, nor the addition of filters for simulating event sensors. However, although several improvements have been made, the code is heavy and requires a powerful machine, especially for the observations module and some propagators. It has been identified that the root of the problem could be on the Astropy unit conversion functionality, which is redundant, and when huge iterations are deployed, this is a penalization. In future work, this issue could be solved by making use of more efficient libraries.

When talking about validation, this has been done carefully, so future users can be sure that they will get reliable data.

Without no doubt, an Event-camera in space would be very beneficial to the scientific community. It could allow scientists to better characterize the Low Earth orbit and the impact of space debris. It will allow better statistical models to predict the possibilities of collisions and prevent them.

Bibliography

[1] Davide Farnocchia, Davide Bracali Cioci, and Andrea Milani. Robust resolution of Kepler's equation in all eccentricity regimes. *Celestial Mechanics and Dynamical Astronomy*, 116(1):21–34, May 2013.

[2] ESA Space Debris Office. *ESA's Annual Space Environment Report*. ESA, 2024.

[3] Lara Schuberth, Vincenzo Messina, Ramón María García Alarcia, Jaspar Sindermann, Kian Bostani Nezhad, Roberto Aldea Velayos, Sofia Lennerstrand, Julie Rollet, Federico Sofio, Alessandro Tinucci Monibas, Leonhard Kessler, and Alessandro Golkar. Leveraging event-based cameras for enhanced space situational awareness: a nanosatellite mission architecture study, 2024.

[4] Loren Grush. A future with tens of thousands of new satellites could 'fundamentally change' astronomy: report. https://www.theverge.com/2020/8/26/21401455/satellite-mega-constellations-astronomy-spacex-amazon-oneweb-bright-internet-space, 2024. Accessed: 2024-09-01.

[5] ESA. About space debris. https://www.esa.int/Space_Safety/Space_Debris/About_space_debris, 2024. Accessed: 2024-09-01.

[6] DLR. Space debris. https://www.dlr.de/en/ar/topics-missions/space-safety/space-debris, 2024. Accessed: 2024-09-01.

[7] NASA Orbital Debris Program Office. Orbital debris quarterly news, july 2011. 15(3), 2011.

[8] Asher Isbrucker. Kessler syndrome: What happens when satellites collide. https://asherkaye.medium.com/kessler-syndrome-what-happens-when-satellites-collide-1b571ca3c47e, 2024. Accessed: 2024-09-01.

[9] ESA Debris User Portal. Space environment statistics. https://sdup.esoc.esa.int/discosweb/statistics/, 2025. Accessed: 2025-02-02.

[10] Donald Kessler, Nicholas Johnson, J.-C Liou, and Mark Matney. The kessler syndrome: Implications to future space operations. *Advances in the Astronautical Sciences*, 137, 01 2010.

[11] Tim Flohrer and Holger Krag. Space surveillance and tracking in esa's ssa programme. In *Proceedings 7th European Conference on Space Debris, Darmstadt, Germany, https://conference. sdo. esoc. esa. int*, volume 1, 2017.

[12] ESA. Space surveillance and tracking - sst segment. https://www.esa.int/Space_Safety/Space_Surveillance_and_Tracking_-_SST_Segment, 2024. Accessed: 2024-09-01.

[13] Kingsley Kwadwo Oteng-Amoako. Space surveillance networks. 2016.

[14] United States Space Force. Geosynchronous space situational awareness program. https://www.spaceforce.mil/About-Us/Fact-Sheets/Article/2197772/geosynchronous-space-situational-awareness-program/, 2025. Accessed: 2025-02-02.

[15] Gunter's Space Page. Silentbarker. ttps://space.skyrocket.de/doc_sdat/silentbarker.htm, 2025. Accessed: 2025-02-02.

[16] New Space. Vyoma space (flamingo). https://www.newspace.im/constellations/vyoma-space, 2025. Accessed: 2025-02-02.

[17] New Space. Northstar (skylark). https://www.newspace.im/constellations/northstar, 2025. Accessed: 2025-02-02.

[18] Jayant Sharma, Grant H Stokes, Curt von Braun, George Zollinger, and Andrew J Wiseman. Toward operational space-based space surveillance. *Lincoln Laboratory Journal*, 13(2):309–334, 2002.

[19] Davide Scaramuzza. Tutorial on event-based cameras.

[20] Hamid Sarmadi, Rafael Muñoz-Salinas, Miguel A. Olivares-Mendez, and Rafael Medina-Carnicer. Detection of binary square fiducial markers using an event camera. *IEEE Access*, 9:27813–27826, 2021.

[21] Arnon Amir, Brian Taba, David Berg, Timothy Melano, Jeffrey McKinstry, Carmelo Di Nolfo, Tapan Nayak, Alexander Andreopoulos, Guillaume Garreau, Marcela Mendoza, Jeff Kusnitz, Michael Debole, Steve Esser, Tobi Delbruck, Myron Flickner, and Dharmendra Modha. A low power, fully event-based gesture recognition system. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7388–7397, 2017.

[22] Catherine Schuman, Shruti Kulkarni, Maryam Parsa, J. Mitchell, Prasanna Date, and Bill Kay. Opportunities for neuromorphic computing algorithms and applications. *Nature Computational Science*, 2:10–19, 01 2022.

[23] Guillermo Gallego, Tobi Delbruck, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jorg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, January 2022.

[24] Prophesee. Space situational awareness with event-based vision. https://www.prophesee.ai/space-situational-awareness-event-based-vision/, 2024. Accessed: 2024-09-02.

[25] Western Sydney University. Astrosite. https://www.westernsydney.edu.au/icns/research_projects/current_projects/astrosite, 2024. Accessed: 2024-09-02.

[26] Sony. Sony imx636 sensor webpage. https://www.sony-semicon.com/en/products/is/industry/evs.html, 2025. Accessed: 2025-15-02.

[27] Nicoletta Bloise, Elisa Capello, Matteo Dentis, and Elisabetta Punta. Obstacle avoidance with potential field applied to a rendezvous maneuver. *Applied Sciences*, 7:1042, 10 2017.

[28] Poliastro. Loading omm and tle satellite data¶. https://docs.poliastro.space/en/stable/examples/Loading%20OMM%20and%20TLE%20satellite%20data.html, 2025. Accessed: 2025-24-02.

[29] Jonathan C. McDowell. General catalog of artificial space objects. https://planet4589.org/space/gcat, 2020. Accessed: 28.08.2024.

[30] Juan Luis Cano Rodríguez and the poliastro development team. Farnocchia's propagator. https://docs.poliastro.space/en/stable/autoapi/poliastro/core/propagation/farnocchia/index.html, 2025. Accessed: 2025-15-02.

[31] Juan Luis Cano Rodríguez and the poliastro development team. Cowell's propagator. https://docs.poliastro.space/en/stable/examples/Propagation%20using%20Cowells%20formulation.html, 2025. Accessed: 2025-15-02.

[32] Juan Luis Cano Rodríguez and the poliastro development team. Poliastro's propagator. https://docs.poliastro.space/en/stable/autoapi/poliastro/twobody/propagation/index.html, 2025. Accessed: 2025-15-02.

[33] Tom Keates. Analytical mechanics of space systems – second edition, h. schaub and j. l. junkins, american institute of aeronautics and astronautics, 1801 alexander bell drive, suite 500, reston, va 20191-4344, usa. 2009. distributed by transatlantic publishers group, unit 242, 235 earls court road, london, sw5 9fe, uk, (tel: 020-7373 2515; e-mail: richard@tpgltd.co.uk). 793pp. illustrated. £76. (10 *The Aeronautical Journal*, 113:747, 11 2009.

[34] David Vallado, Paul Crawford, Richard Hujsak, and T.S. Kelso. Revisiting spacetrack report 3: Rev. 08 2006.

[35] Max Born and Emil Wolf. *Elements of the theory of diffraction*, page 412–516. Cambridge University Press, 2019.

[36] Karel Fliegel. Modeling and measurement of image sensor characteristics. *Radioengineering*, 13, 12 2004.

[37] CVI Melles Griot. *Fundamental Optics*, page 1–58. CVI Melles Griot, 2024.

[38] Nicholas James Gregory Hollows. Understanding focal length and field of view. www.edmundoptics.com/knowledge-center/application-notes/imaging/understanding-focal-length-and-field-of-view/, 2024. Accessed: 28.08.2024.

[39] Ansys. Ansys' stk home page. https://www.ansys.com/products/missions/ansys-stk, 2025. Accessed: 2025-24-02.

[40] NASA. General mission analysis tool (gmat) v.r2016a. https://software.nasa.gov/software/GSC-17177-1, 2025. Accessed: 2025-24-02.

[41] Y Hu, S C Liu, and T Delbruck. v2e: From video frames to realistic DVS events. In *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. IEEE, 2021.

[42] Haiqian Han, Jiacheng Lyu, Jianing Li, Henglu Wei, Cheng Li, Yajing Wei, Shu Chen, and Xiangyang Ji. Physical-based event camera simulator. In Aleš Leonardis, Elisa Ricci, Stefan Roth, Olga Russakovsky, Torsten Sattler, and Gül Varol, editors, *Computer Vision – ECCV 2024*, pages 19–35, Cham, 2025. Springer Nature Switzerland.

[43] Juan Luis Cano Rodríguez. poliastro/poliastro: poliastro 0.17.0 (SciPy US '22 edition), July 2022.

[44] Astropy Collaboration, Adrian M. Price-Whelan, Pey Lian Lim, Nicholas Earl, Nathaniel Starkman, Larry Bradley, David L. Shupe, Aarya A. Patil, Lia Corrales, C. E. Brasseur, Maximilian Nöthe, Axel Donath, Erik Tollerud, Brett M. Morris, Adam Ginsburg, Eero Vaher, Benjamin A. Weaver, James Tocknell, William Jamieson, Marten H. van Kerkwijk, Thomas P. Robitaille, Bruce Merry, Matteo Bachetti, H. Moritz Günther, Thomas L. Aldcroft, Jaime A. Alvarado-Montes, Anne M. Archibald, Attila Bódi, Shreyas Bapat, Geert Barentsen, Juanjo Bazán, Manish Biswas, Médéric Boquien, D. J. Burke, Daria Cara, Mihai Cara, Kyle E. Conroy, Simon Conseil, Matthew W. Craig, Robert M. Cross, Kelle L. Cruz, Francesco D'Eugenio, Nadia Dencheva, Hadrien A. R. Devillepoix, Jörg P. Dietrich, Arthur Davis Eigenbrot, Thomas Erben, Leonardo Ferreira, Daniel Foreman-Mackey, Ryan Fox, Nabil Freij, Suyog Garg, Robel Geda, Lauren Glattly, Yash Gondhalekar, Karl D. Gordon, David Grant, Perry Greenfield, Austen M. Groener, Steve Guest, Sebastian Gurovich, Rasmus Handberg, Akeem Hart, Zac Hatfield-Dodds, Derek Homeier, Griffin Hosseinzadeh, Tim Jenness, Craig K. Jones, Prajwel Joseph, J. Bryce Kalmbach, Emir Karamehmetoglu, Mikołaj Kałuszyński, Michael S. P. Kelley, Nicholas Kern, Wolfgang E. Kerzendorf, Eric W. Koch, Shankar Kulumani, Antony Lee, Chun Ly, Zhiyuan Ma, Conor MacBride, Jakob M. Maljaars, Demitri Muna, N. A. Murphy, Henrik Norman, Richard O'Steen, Kyle A.

Oman, Camilla Pacifici, Sergio Pascual, J. Pascual-Granado, Rohit R. Patil, Gabriel I. Perren, Timothy E. Pickering, Tanuj Rastogi, Benjamin R. Roulston, Daniel F. Ryan, Eli S. Rykoff, Jose Sabater, Parikshit Sakurikar, Jesús Salgado, Aniket Sanghi, Nicholas Saunders, Volodymyr Savchenko, Ludwig Schwardt, Michael Seifert-Eckert, Albert Y. Shih, Anany Shrey Jain, Gyanendra Shukla, Jonathan Sick, Chris Simpson, Sudheesh Singanamalla, Leo P. Singer, Jaladh Singhal, Manodeep Sinha, Brigitta M. Sipőcz, Lee R. Spitler, David Stansby, Ole Streicher, Jani Šumak, John D. Swinbank, Dan S. Taranu, Nikita Tewary, Grant R. Tremblay, Miguel de Val-Borro, Samuel J. Van Kooten, Zlatan Vasović, Shresth Verma, José Vinícius de Miranda Cardoso, Peter K. G. Williams, Tom J. Wilson, Benjamin Winkel, W. M. Wood-Vasey, Rui Xue, Peter Yoachim, Chen Zhang, Andrea Zonca, and Astropy Project Contributors. The Astropy Project: Sustaining and Growing a Community-oriented Open-source Project and the Latest Major Release (v5.0) of the Core Package. , 935(2):167, August 2022.

[45] Travis E. Oliphant. *Guide to NumPy*. 2006.

[46] Ayoosh Kathuria. Nuts and bolts of numpy optimization part 1: Understanding vectorization and broadcasting. https://blog.paperspace.com/numpy-optimization-vectorization-and-broadcasting/, 2021. Accessed: 2025-09-02.

[47] Dash. Dash plotly home webpage. https://dash.plotly.com/, 2025. Accessed: 2025-02-07.

[48] Threejs. Three.js home webpage. https://threejs.org/, 2025. Accessed: 2025-09-02.