Foundations for a Digital Twin of the CHESS CubeSat

Master Semester Project

Author

Mathilde Simoni
EPFL
Lausanne, Switzerland
mathilde.simoni@epfl.ch

Professor

Jean-Paul Kneib
jean-paul.kneib@epfl.ch

Supervisors

Andrew Price
andrew.price@epfl.ch

Mathieu Udriot
mathieu.udriot@epfl.ch

January 10th, 2025

Contents

Acronyms and Abbreviations

Acronym or Abbreviation	**Meaning**
ADCS	Attitude Determination & Control System
AOP	Argument Of Perigee
CHESS	Constellation of High-Energy Swiss Satellites
CONOPs	Concept of Operations
CROC	Cross section of Complex bodies
DRAMA	Debris Risk Assessment and Mitigation Analysis
ECC	Eccentricity
EPFL	École Polytechnique Fédérale de Lausanne
EPS	Electrical Power System
ESA	European Space Agency
EST	EPFL Spacecraft Team
GNSS	Global Navigation Satellite System
GS	Ground Station
HK data	HouseKeeping data
INC	Inclination angle
LEOP	Launch and Early Orbit Phase
OBC	On-Board Computer
RAAN	Right Ascension of the Ascending Node
SMA	Semi-Major Axis
SSO	Sun-Synchronous Orbit
TA	True Anomaly
UHF	Ultra High Frequency

Table 0.1: Acronyms and abbreviations

1. Introduction

This report details the development of a new simulation framework in Python to support the EPFL Spacecraft Team's CHESS mission. The application created serves two purposes:

1. **Simulation tool for mission design**: The tool aims to assist in the mission design of the CHESS CubeSat, helping to inform decisions across different subsystems. The simulations carried out are essential for determining trajectory, Sun exposure, and other critical parameters, in order to accelerate development, testing, and validation tasks. It follows the principles of Model-Based Systems Engineering (MBSE), defined as "the formalized application of modeling to support system requirements, design, analysis, verification, and validation activities"[1] in ESA's Technology Harmonisation Dossier on *Model-Based Systems Engineering*[1].

2. **Foundations for a digital twin**: The framework also lays the groundwork for a future digital twin of the CHESS CubeSat. This digital twin will be capable of simulating satellite operations, including subsystem anomalies to observe their impact on the overall system. A digital twin can be defined as "a computerized representation that serves as the real-time digital counterpart of a physical object or process"[1].

As outlined in the ESA harmonization report, simulation frameworks can be utilized to "drive the construction of high-fidelity digital twins"[1], which aligns with the goal of this project.

The Constellation of High-Energy Swiss Satellites (CHESS) is a EPFL Spacecraft Team's mission, involving two 3U CubeSats in collaboration with other Swiss institutions, including the universities of Bern and Zurich. The mission's main scientific objective is to improve the understanding of the Earth's upper atmosphere with in-situ measurements. With this goal in mind, the 3U CubeSat mission CHESS Pathfinder 1 is equipped with an instrument to study the chemical composition of the terrestrial exosphere and its density. The satellite will be placed in a circular sun-synchronous orbit at an altitude of 500 km. While the simulations described in this report are specific to the CHESS mission, the framework is adaptable to other missions by updating the parameters.

Other Available Resources

- Project GitHub repository (access regulated by the EPFL Spacecraft team):
  - For instructions on how to use the simulation framework, refer to the README file
  - Developer documentation is available in HTML format in the *docs/html/* folder
  - All constants specific to the CHESS mission utilized in the framework are stored in an Excel document "parameters.xlsx" in the *docs/* folder .
- *02_DigitalTwin/* folder in the EPFL's spacecraft team google drive:
  - Meeting notes and slides can be found in the *Semester_Project_Fall_2024/meetings/* folder
  - Available documents included in the bibliography of this report are stored in the *Literature/* folder.

[1]Shared with the consent of the Swiss delegate to the ESA Harmonisation Team

2. Framework Overview

This application was developed for simulations of the CHESS satellite orbiting Earth in an elliptic orbit. It does not simulate the LEOP nor the end-of-life phases.

2.1. Programming Language and Libraries

The framework was implemented from scratch using Python. This programming language was chosen for its universality and ease of understanding, making it particularly suitable for the EPFL Spacecraft Team. Given that students rotate every 6 months, the steep learning curve associated with specialized software (such as GMAT or STK) can pose significant challenges. Python's readability and extensive library ecosystem address these concerns. Two libraries are particularly useful for this project:

- *Poliastro*: This library is used for orbit propagation and associated functionalities, including plotting, accessing ephemeris data, generating ground tracks, and converting between different orbital representations.

- *Astropy*: It provides the `astropy.units` module, which is useful for handling arithmetic operations involving physical quantities. Additionally, it includes `astropy.constants`, which offers a comprehensive database of physical constants and `astropy.coordinates` to facilitate the management of various coordinate frames.

2.2. User Parameters

The framework allows users to configure simulations using JSON files. JSON was chosen as the format because it's widely used for configuration in Python applications and is easy to edit. User parameters are divided into five categories, each with its own configuration file:

- **Spacecraft**: Defines general parameters like mass, cross-sectional area, initial operating mode, and subsystem-specific details.
- **Orbit**: Specifies initial orbital elements, epoch, and orbit type (e.g., SSO).
- **Simulation**: Includes general simulation settings, such as duration, timestep, and the atmospheric model to use.
- **Ground station**: Lists ground stations with their respective locations and elevation angles.
- **Mission design**: Gathers parameters to generate a report (e.g., data to save, figures to generate) and additional user input, such as commands to initiate SAFE mode at specific times. This file could also be expanded to stop simulations when certain conditions are met.

The **spacecraft**, **orbit**, and **ground station** configuration files are intented to be used by the engineers of the CHESS CubeSat as the parameters are mission-specific. The two other files, **mission design** and **simulation** are intended to be used by developers and users.

At the end of a simulation, users can choose to save the current state (specified in the "mission_design.json" configuration file). If enabled, new "orbit.json" and "spacecraft.json" configuration files are generated and can be reused for subsequent simulations. The other three configuration files are not tied directly to the simulation state and therefore are not saved.

2.3. Code Structure and Flow

The UML diagram in Figure 2.1 summarizes the general structure of the code.

2.3.1. `Simulation` Class

The `Simulation` class acts as the central manager for orchestrating the various components of the simulation, including:

- Parsing user arguments
- Initializing times parameters (duration…)

- Gathering the `Spacecraft`, `GroundStation`, `SwitchAlgo`, and `Propagator` class instances as attributes
- Running the main simulation loop
- Calling the necessary functions to produce the desired output

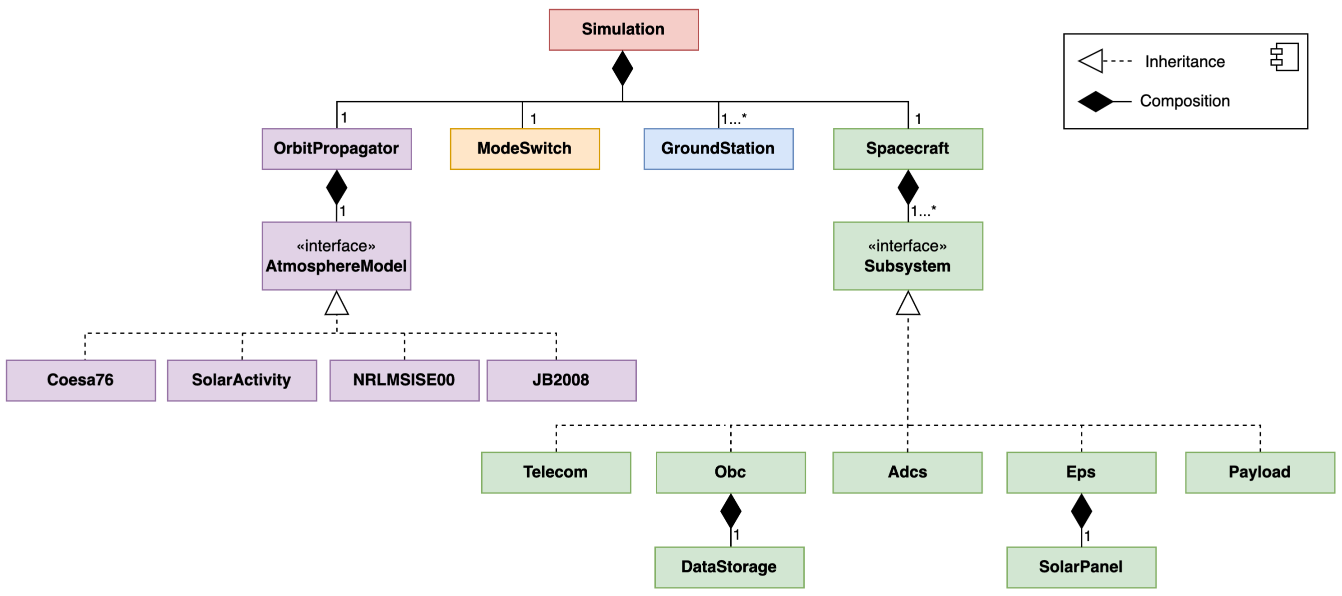


Figure 2.1: UML diagram of the framework

At each timestep, the main simulation loop carries out these actions:

1. Propagate the spacecraft to the next position using the `Propagator`
2. Ask the `Propagator` to calculate position-based variables, including:
   - Eclipse status
   - Visibility window (using a list of `Groundstation` instances)
3. Consider other user-defined parameters (such as planned measurement windows)
4. Check for potential safe flags raised by `Obc`
5. Ask `SwitchAlgo` to find the new operating mode
6. Ask `Spacecraft` to update all its subsystems
7. Save the data at the current timestep

Note: The ground stations are required by the `Simulation` for ground-track plots, by the `Propagator` to calculate the visibility windows, and by the `Telecom` subsystem to do the data rate budget. Hence, it is stored in the `Simulation` class.

2.3.2. `Spacecraft` Class

The `Spacecraft` class stores subsystems and organizes their update at every timestep as follows:

1. Update each subsystem individually by invoing their `update()` function. These update processes operate independently of one another, hence the execution order has no impact
2. Compute the power consumed by every subsystem
3. Ask `Eps` to update the batteries based on data from the other subsystems at the current timestep
4. Update the data storage based on the data generated or transmitted at the current timestep
4. Check if a safe flag has been raised by any subsystem

2.3.3. Subsystems

The organization of the spacecraft in subsystems follows the architecture of the CHESS CubeSat, highlighted in Figure 2.2.

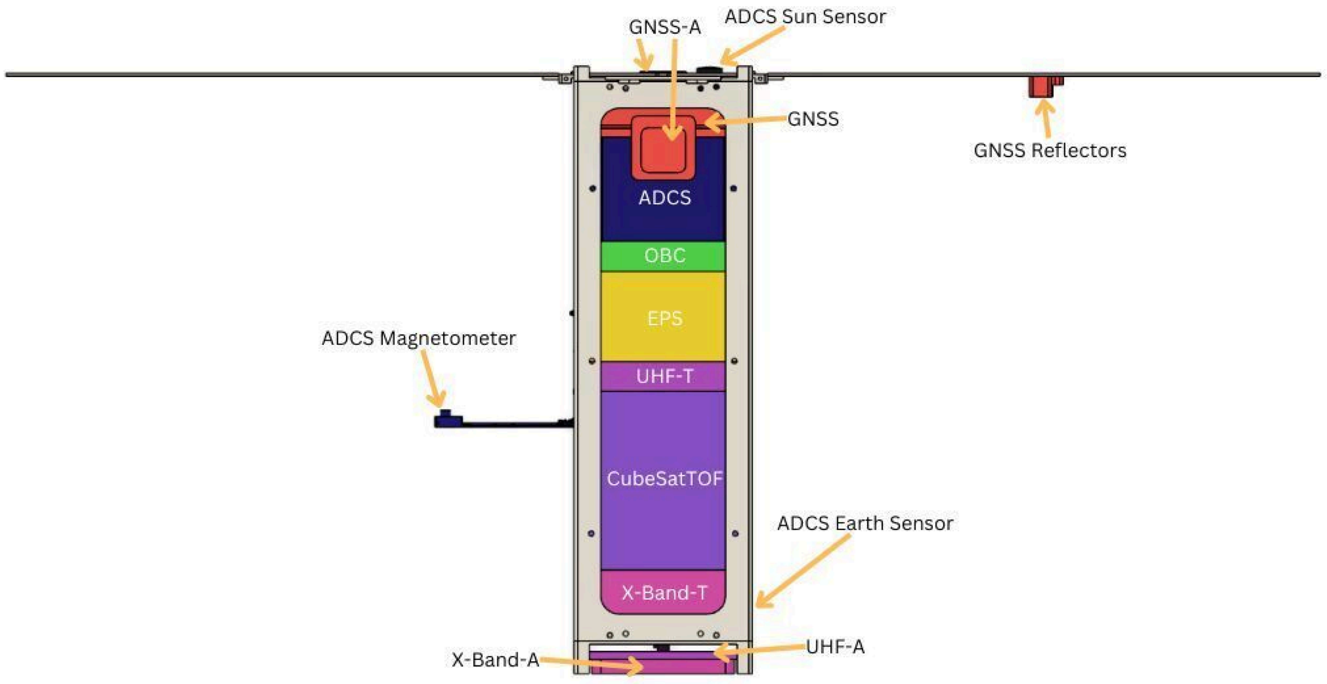


Figure 2.2: CHESS CubeSat architecture

There are 5 subsystem classes: `Eps`, `Adcs`, `Telecom` (gathering *X-band-T* and *UHF-T* modules), `Payload` (gathering *CubeSatTOF* and *GNSS* modules), and `Obc`. These all have a common parent class called `Subsystem` which forces all its children to implement the following functions:

- `update()`: Called by the `Spacecraft` at every timestep and enables each subsystem to update its attributes independently
- `raise_safe_flag()`: Returns a boolean set to `True` if SAFE mode should be triggered (based on the subsystem's state)
- A dictionary of mean consumption rates depending on the operating mode (each subsystem consumes a specific amount of energy depending on the current operating mode, as defined in the power budget of the EST)
- `computePowerConsumed()`: Returns the current consumption rate

The `Subsystem` class acts as an interface which makes the application modular as new subsystems can be added easily. The detailed implementation of each subsystem will be described in Section 4.

2.3.4. Mode Switch Algorithm

The mode switch algorithm, which determines the satellite's current operating mode, is a core component of the simulation. Understanding the active mode is crucial for calculating power output, attitude, data generation, transmission, and other key metrics.

The algorithm is implemented in the `ModeSwitch` class, following the decision tree defined by the EPFL Spacecraft Team and represented in Figure A.1 in the Appendix. Table 2.1 briefly presents the operating modes.

Operating Mode	**Main Goal**
IDLE	Transition between other operating modes
SAFE	Save battery and communicate with the ground station in case of an unpredicted event
CHARGING	Charge the batteries
MEASUREMENT	Perform scientific measurements
UHF-COM	Communicate with the ground station
XBAND-COM	Downlink scientific data

Table 2.1: Spacecraft operating modes

Some additional assumptions were made during the implementation to address specific scenarios not fully defined in the original decision tree. These assumptions are listed below:

- **Measurement window limitation**: Measurements are restricted to n windows per day (user defined, 1 by default), even when multiple measurement opportunities exist. This additional requirement was introduced to prevent excessive measuring, which would leave insufficient time for data to be downlinked and induce low battery energy level. While the initial decision tree didn't impose this restriction, it aligns with real-world mission planning, where measurement windows are usually scheduled.
- **Measurement termination**: Measurements stop when either the maximum measurement time is reached or the storage is full.
- **Communication windows**: Communication windows are considered similar as visibility windows. The satellite communicates whenever it is visible from a ground station, without any specific schedule.
- **Housekeeping data and X-band priority**: The framework assumes no requirement to downlink all housekeeping (HK) data before switching to XBAND-COM mode. During a communication window, if there is scientific data to downlink:
  - The satellite first establishes a handshake in UHF-COM mode,
  - It then switches to XBAND-COM mode to downlink all data,
  - If all data is successfully downlinked and the satellite remains visible, it switches back to UHF-COM mode to downlink any remaining HK data.

  This approach was chosen because HK data is already typically transmitted in regular 10-second beacon intervals (which can reach any ground station on Earth and is subsequently sent to the main ground station), so it is not a priority to downlink it to the ground station during a visibility window.
- **Communication termination**: Communication stops when either the maximum communication time is reached or the satellite is no longer visible. As uplink data functionality has not yet been implemented, no additional conditions for terminating communication have been defined.
- **X-band downlink termination**: Downlink stops when all data has been successfully transferred, provided the satellite remains visible.

2.3.5. SAFE Mode Handling

The current implementation of SAFE mode handling is preliminary and limited, as many of the triggers are not well-defined. Below is the list of potential SAFE mode triggers:

- Tumbling rate > `tumb_max`
- **Battery level < `batt_min`**

- **No communication for `t` $>$ `t_max`**
- HK data failure
- Power failure
- UHF antenna deployment failure
- Solar panel deployment failure
- Software failure
- Temperature `T` $>$ `T_max` or `T` $<$ `T_min`
- **Reception of command from GS to initiate SAFE mode**
- X-band failure
- Payload failure
- OBC failure

Out of these, only the triggers in **bold** have been implemented. Since the `t_max` parameter is not yet defined by the team, it has been temporarily set to 1 day as an upper bound.

The remaining triggers require further definition and refinement before they can be implemented. This will likely involve additional input from the team to clarify thresholds, conditions, and expected behaviors for SAFE mode activation.

3. Propagation

The propagator is the foundation for an effective simulation. It defines the orbit scenario, which in turn determines the satellite's thermal and power environments, as well as its capability to communicate with ground stations. The CHESS Pathfinder 1 mission is planned to have as Circular Sun-Synchronous Orbit (SSO) with an initial altitude of 500 km. The orbital elements are as follows:

SMA[km]	**INC[deg]**	**ECC**	**RAAN[deg]**	**AOP[deg]**	**TA[deg]**
6878	97.4065	0.0	110	0.0	0.0

This section draws primarily from two sources: *Fundamentals of Astrophysics and Applications* [2] and *Spacecraft Dynamics and Control* [3].

3.1. Modeling

As stated in *Spacecraft Dynamics and Control* [3], the motion of a spacecraft can ideally "be regarded as a two-body motion around a central body, which means that the spacecraft is affected by the gravity of only the central body"[3]. However, in order to simulate reality closely, we need to consider additional orbital perturbations which lead to deviations from the orbit determined by Kepler's law. These comprise J2 perturbations, atmospheric drag, third-body perturbations, solar radiation pressure, and many others. The considered orbit for the CHESS Pathfinder 1 mission is classified as a Low-Earth Orbit (LEO) for which some perturbations can be neglected. Section 2.6 of [3] discusses necessary perturbation forces to consider:

- J2 perturbations: "for most orbits around the earth, it is sufficient to only consider J2"[3].
- Atmospheric drag perturbations: "atmospheric drag is considered the dominant perturbing force below 200 km. However, above 1,000km, it can be neglected in most cases"[3].
- Gravitational forces of the Sun and Moon: "below 1600 km, these perturbations can be neglected"[3].
- Solar radiation pressure: "perturbations due to solar radiation pressure can usually be neglected"

Therefore, we choose to only consider J2 and atmospheric drag perturbations. While J2 perturbations only require a constant J2 coefficient, drag force calculations are more complex. Indeed, drag is the dominant source of errors for LEO satellites [4], which are mostly caused by either inaccurate ballistic coefficient or incorrect atmospheric density modeling.

3.1.1. Ballistic Coefficient

The ballistic coefficient is calculated using the drag coefficient and the satellite's cross section. Regarding the drag coefficient, it is common to assume it constant and equal to 2.2 for LEO satellites [5].

Regarding the satellite cross-section, it was chosen to keep it constant throughout the simulation. While this prohibits dynamic drag calculations, it would require a 6 d.o.f solver which is very expensive and does require to update the full propagation framework for a negligible difference. We assume that the satellite is tumbling and that its orientation varies over time. To simulate this, we used the ESA public software DRAMA. Through its tool CROC, it provides an estimation of the average cross-section if there was a loss of control and it would then be in a randomly tumbling motion. In this scenario, the average cross section is $0.084\,\text{m}^2$.

3.1.2. Atmospheric Model

A critical aspect of atmospheric modeling is the influence of solar activity. The upper layers of the atmosphere interact with solar winds, causing atmospheric upwelling. This shifts denser air to higher altitudes, leading to increased air density during periods of high solar activity. A visualization of key solar activity indicators—the solar radio flux index (F10.7) and geomagnetic index (Ap)—starting from the planned launch year (2028) is shown in Figure 1.

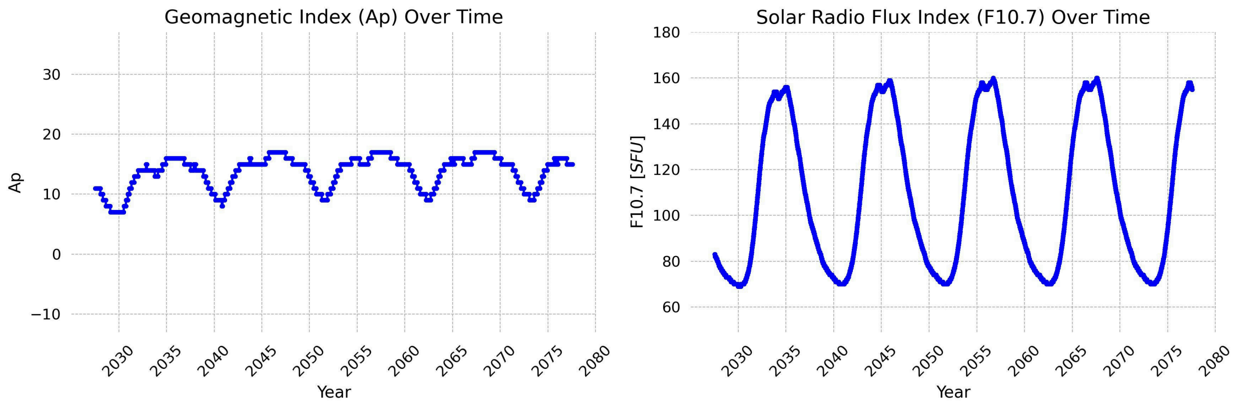


Figure 1: Solar activity variables

The satellite's launch is planned during a period of low solar activity, which implies a potential for an extended satellite lifetime due to lower atmospheric density (hence reduced drag forces).

The different atmospheric models considered in this project are presented in Table 3.2. In order to implement the JB2008 and NRLMSISE00 models, we use a library named *ATMOS* which unfortunately does not accept inputs after a certain date. Hence, we use the periodicity of the solar cycle (11 years, see Figure 1) to go back in time. This might introduce errors, which could be resolved by implementing the model manually. However, due to time constraints, this option was not considered.

Name	**Description**	**Solar Activity**	**Efficiency**
COESA76	• Standard reference model published in 1976	constant: moderate	vert fast
Solar Activity	• Analytical model extracted from [6] • Used by the company *Beyond Gravity* • Assumes an exponential model for atmospheric density and takes into account variations from Solar radio flux index F10.7 and the geomagnetic index Ap	varying	fast
JB2008	• "Jacchia-Browman" model, version 2008 • Uses "analytical expressions for determining exospheric temperature as a function of position, time, solar activity and geomagnetic activity" [2] • Calculates density using the computed temperature and based on empirical temperature profiles or the diffusion equation [2]	varying	medium speed
NRLMSISE 00	• Introduced after the "Jacchia-Browman" model • Calculates the precise composition of the atmosphere	varying	slow

Table 3.2: Atmospheric models

The air density calculated from each of these models is displayed in Figure 3.2 for low, medium and high solar activity. While the COESA76 model stays constant with solar activity, the other three models have an increasing density with solar activity (the y-axis range is the same in all 3 figures). In addition, the "solar activity" model behaves very differently than the other models for lower altitude due to its exponential shape which makes it look linear on a y log scale.

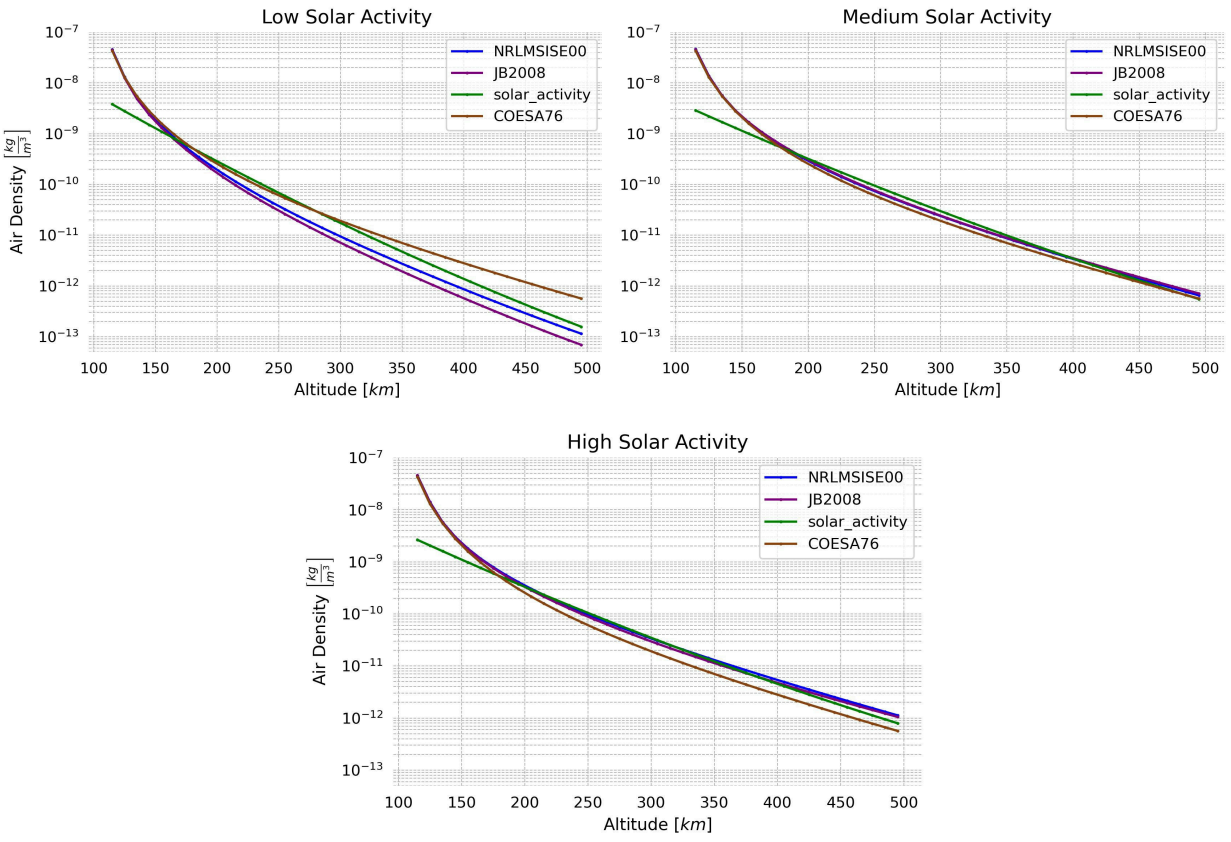


Figure 3.2: Air density depending on altitude for different solar activity intensities

3.2. Implementation

We use *Poliastro* library for orbit propagation, selected based on the following requirements: ease of use, comprehensive documentation with accessible source code, support for J2 and atmospheric drag perturbations, and the ability to propagate by timestep or invoke a step-handler function to execute code at each timestep (e.g., updating all subsystems). Among three evaluated options (*Poliastro*, *Orekit* python wrapper, and *Tudatpy*), *Poliastro* emerged as the best choice, with insights from [7]. The *Orekit* python wrapper was excluded due to its Java-based origin, which complicates understanding for a Python project and raises concerns about potential inefficiencies when converting between the programming languages. While *Tudatpy* offered good documentation, it was discarded as the source code is difficult to understand compared to *Poliastro*, not aligning with the initial choice of Python for clarity and accessibility.

The propagation algorithm uses Cowell's formulation to easily add perturbing accelerations to the two-body equation:

$$\ddot{r} = \frac{-\mu}{\|r\|^3} r + a_d,$$

where a_d is the total acceleration caused by J2 and drag forces, r the vector from the center of the Earth to the satellite, and $\mu = GM$ the product of the gravitational constant G with the mass of Earth M. This formulation permits to add each perturbation linearly [2]. It is then integrated using a Runga Kutta scheme DOP853 of order 8. This is a symplectic scheme, hence well suited to physics simulation to ensure that no energy is added to the system when solving the equation.

3.2.1. Timestep Study

We conducted a study to determine the optimal timestep for propagation. As shown in Figure 5, the two key orbital elements—RAAN and altitude, which are critical for estimating satellite lifetime and maintaining a sun-synchronous orbit—remain unaffected by the choice of timestep. Since no significant impact was observed on other orbital elements either, we conclude that timestep selection does not influence propagation accuracy. This is likely due to hidden optimizations performed by the *Poliastro* library during propagation.

However, the timestep must still be sufficiently small for subsystem updates. This is particularly crucial for the telecom module, as satellite passes can sometimes be as brief as 20 seconds. A potential improvement could involve decoupling the timestep for subsystem updates from the timestep used for propagation, enabling a faster and more efficient simulation. For simulations presented in this report, we used a timestep of $\Delta t = 600s$ for propagation-only simulation (no subsystem update) and $\Delta t = 10s$ otherwise.

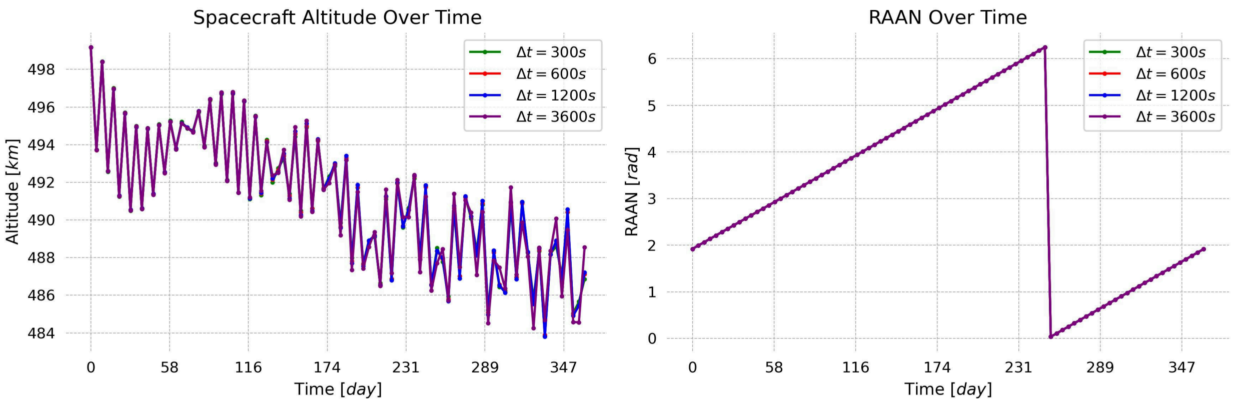


Figure 5: Variation of orbital elements depending on propagation timestep

3.3. Validation

We use DRAMA [8] to validate the propagator through a lifetime analysis. Figure 3.6 displays the satellite's altitude for the different atmosphere models presented in Table 3.2.

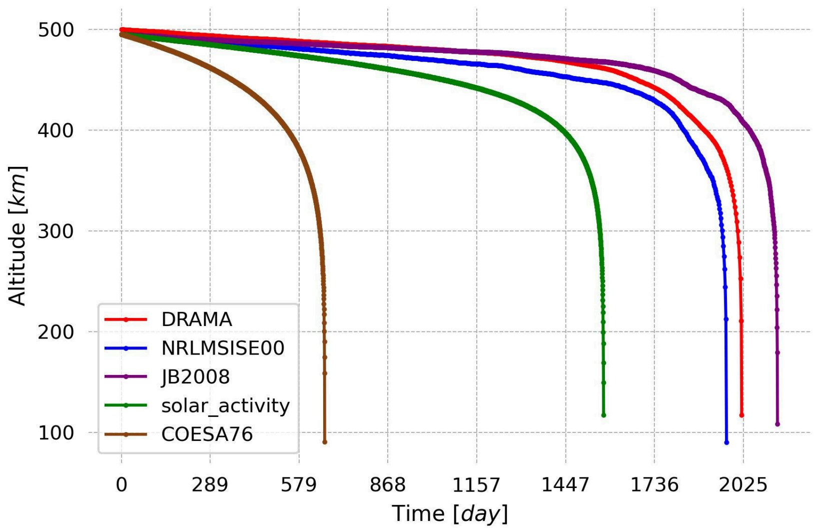


Figure 3.6: Satellite's lifetime with different atmospheric models

We observe that the NRLMSISE00 model is the closest to DRAMA's results. Therefore, we select it for our simulations. Using this model, the expected lifetime is of 1971 days, which compared to DRAMA result of 2020 days, represents a difference of 2.45%. This is considered satisfactory. In order to reduce computation time, we calculate air density every day (by default, user parameter to be specified) instead of every timestep, since the air density change in a period of 24 hours is minimal.

This study also clearly highlights that solar activity needs to be considered since the satellite lifetime is lower than the solar cycle of 11 years. The small 2.45% difference might be because of small perturbations such as the influence of the Moon, Jupiter and the Sun on the satellite or additional zonal coefficients (e.g., J3). It is important to also note that the solar activity values F10 and Ap are predicted and not measured, hence can also fluctuate [8].

Finally, it is important to remember that the cross section used can play a major role in satellite lifetime. Experiments were computed with minimum, average and maximum cross section, which resulted in lifetimes from 4.7 to 7.2 years, as displayed in Figure 3.7. The cross section used in this project based on "tumbling motion" is only an approximation.

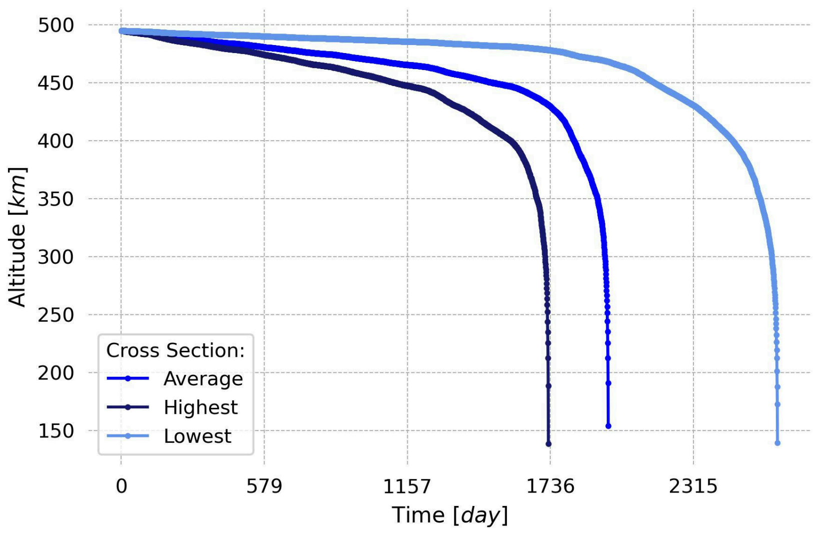


Figure 3.7: Satellite's ifetime with different cross sections

3.4. Visualizations

The application produces plots of the orbital elements evolution during the simulation. In Figure 8, we observe the evolution of the RAAN and the eccentricity during 1 year. The RAAN goes over 1 full cycle, which is a characteristic of sun-synchronous orbits. The eccentricity stays constant (up to numerical errors), which is also an expected feature.

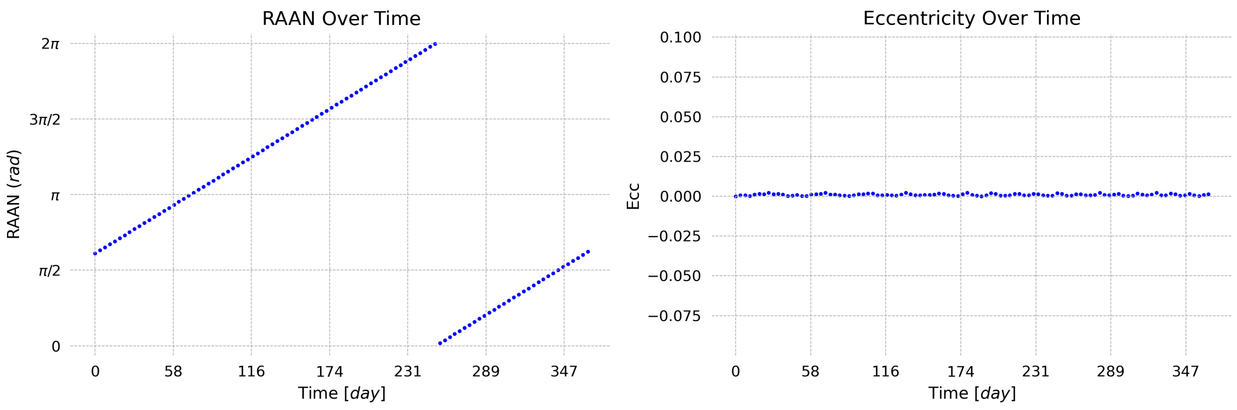


Figure 8: Evolution of RAAN and ECC during a simulation of 1 year

In addition, we produce 2D plots of the initial orbit and a 3D plots of the trajectory (Figure 9), as well as a ground-track plots. The ground track displayed in Figure 3.10 simulates the satellite's trajectory during 10 days.

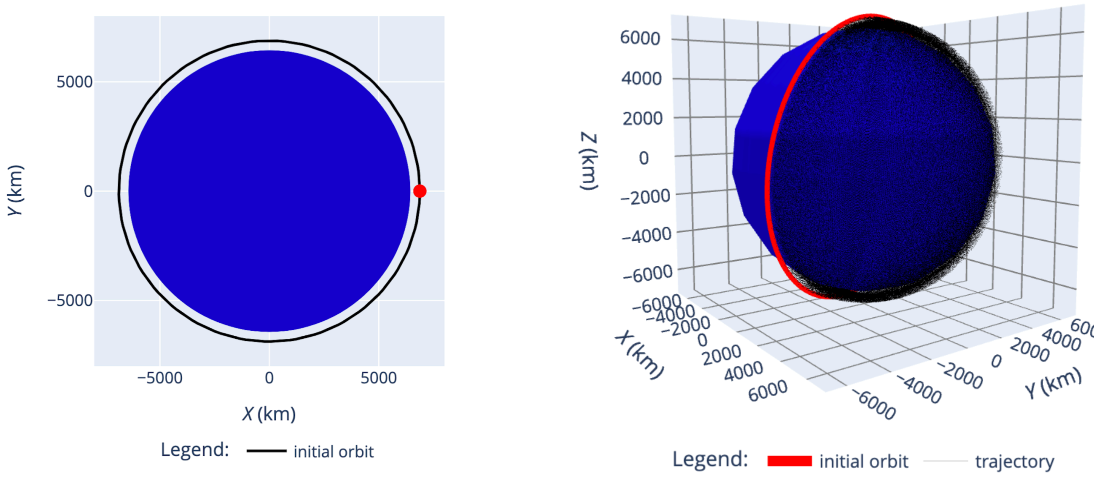


Figure 9: 2D and 3D plots of the orbit

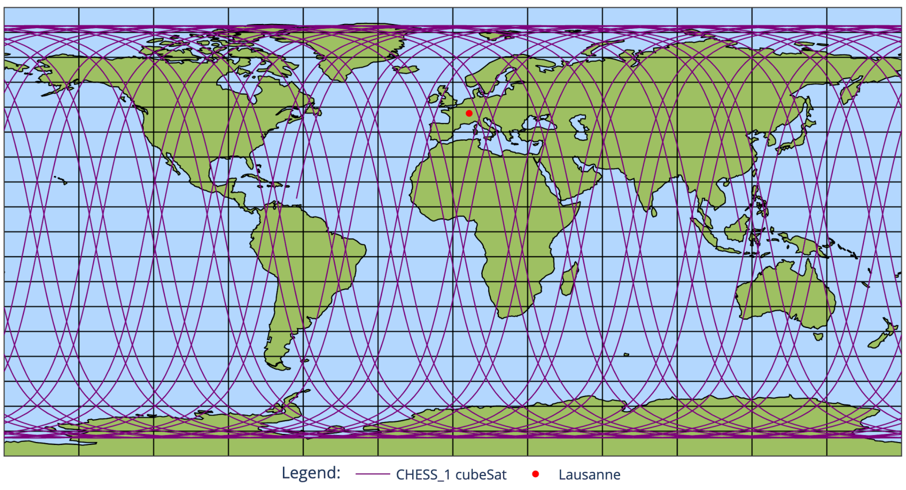


Figure 3.10: Ground-track plot for a simulation of 10 days

3.5. Associated Functions

3.5.1. Eclipse Status

For eclipse status, we use the function already implemented by *Poliastro*. This function uses a shadow function taken from the book *Methods of Orbit Determination* from Pedro Escobal [9]. It assumes circular bodies and doesn't account for flattening.

3.5.2. Visibility Windows

Calculating the visibility windows is necessary for the Telecom module. The implementation is straightforward: we calculate the angle between the vector pointing to the satellite from the ground station and the horizontal plane at the ground station and make sure it is bigger than the ground station's minimum elevation angle (which is fixed at 10 deg for Lausanne).

The framework supports multiple ground stations. However, when calculating visibility windows, if one ground station is found to be in range, it is assumed to be the only active station at that time. The order in which ground stations are checked depends on their arrangement in the user-provided ground station configuration file.

4. Subsystem Implementation

This section outlines the modeling of each subsystem. While significant effort was dedicated to creating the simulation framework, with a particular emphasis on orbit propagation, the current subsystem models remain simple. However, they are designed to be modular and flexible, ensuring that more advanced features can be integrated easily as the project progresses.

4.1. ADCS

The class `Adcs` assures the simulation of satellite orientations according to the CONOPs. Each operating mode corresponds to a specific attitude, detailed in Table 4.1.

Operating Mode	Attitude
IDLE	**Y/Z Thomson spin**: The satellite rotates along inertia axes. Enables to passively stabilize the satellite without consuming too much energy and is commonly used during eclipses
SAFE	**Sun spin**: Z+ axis towards the Sun, can rotate around Z axis
CHARGING	**Sun spin**: Z+ axis towards the Sun, can rotate around Z axis
MEASUREMENT	**Nadir pointing**: Z- axis towards nadir
UHF-COM	**Nadir spin**: Z- axis towards nadir, can rotate around Z axis
XBAND-COM	**Ground tracking**: Z- axis towards ground station

Table 4.1: Spacecraft attitudes

In the CHESS CubeSat, the ADCS system is comprised of a multitude of sensors that feed information to the ADCS computer which in turns runs the attitude determination algorithms and then sends commands to the actuators to move the CubeSat accordingly. The primary components include a deployable magnetometer, three reaction wheels, three magnetorquers, a sun sensor, and an earth infrared (IR) sensor.

In the simulation framework, this subsystem is implemented at a high level. It updates the stored attitude instantaneously based on the selected operating mode, without simulating the detailed behavior of individual sensors or actuators. This, in turn, affects power generation within the `Eps` subsystem. Enhancing the simulation to include sensor and actuator dynamics is an important future improvement to increase the realism of the model.

4.2. EPS

The CHESS CubeSat Electrical Power System is responsible for managing the power and distribute it to the other subsystems. It has secondary batteries (used during eclipses) and solar panels. It is implemented in the simulation framwework with the class `Eps`. The general idea is to iteratively update power charge by computing energy consumption and generation per timestep and subsequently updating battery charge.

4.2.1. Energy Consumption

Energy consumption is computed via mean (+margin) power consumption with respect to active mode multiplied by the current timestep duration. The `Spacecraft` instance gathers the consumption of all subsystems and passes it to the `Eps` instance to update battery charge.

4.2.2. Energy Generation

Energy input is entirely derived from solar radiation on the solar panels and is modeled as instantaneous power generation. This depends on the eclipse status (no power is generated when the Earth blocks sunlight), as well as the satellite's attitude and location, which influences the orientation of the panels relative to the Sun. If the satellite is not in eclipse, power generation is calculated using the following formula:

$$\text{power} = \text{panel\_effective\_surface} \times \text{solar\_flux} \times \text{panel\_efficiency}$$

By default, the solar panel efficiency is 0.25 (modifiable in the "spacecraft.json" configuration file), and the solar flux constant is $1\,367\,\mathrm{W\,m^{-2}}$. These values are approximations. In practice, the solar flux varies slightly as the Earth orbits the Sun, and efficiency could be modeled more accurately by incorporating a temperature model.

The panel effective surface depends on both the panel surface area and the attitude, which determines the incident angle of sunlight on the panels:

$$\text{panel\_effective\_surface} = \text{panel\_surface} \times \text{intensity},$$

where the intensity is defined as:

$$\text{intensity} := \begin{cases} 0.3 \times \cos(\theta) & \text{if in IDLE mode,} \\ \cos(\theta) & \text{otherwise.} \end{cases}$$

Here, θ is the angle between the normal to the solar panels and the vector pointing from the satellite to the Sun (indident angle of sunlight).

For "sun spin" attitude, $\theta = 0$ so the intensity is maximum (intensity $= 1$). For "nadir spin" and "nadir pointing," the panel normal aligns with the vector from the Earth's center to the satellite. In "ground tracking," it aligns with the vector from the ground station to the satellite. For "Thomson spin", θ is simulated as a random value within $-\frac{\pi}{2} \leq \theta < \frac{\pi}{2}$, with the cosine of this angle scaled by 0.3 to generate an intensity between 0 and 0.3. This scaling provides a conservative approximation since the attitude of this mode is not easily predictable.

4.2.3. Battery Thresholds

Battery thresholds are critical for the mode switch algorithm. They are used to asses if the batteries are charged enough to switch to MEASUREMENT, XBAND-COM or UHF-COM modes. The first step in determining these thresholds is to estimate the mean solar input. To do this, we run a 15-day simulation which yields a mean solar input of 7.23 W. The thresholds are then calculated using the following formulas:

$$\begin{aligned} \text{measurement\_threshold} &= (\text{mean\_measurement\_consumption} - \text{mean\_solar\_input}) \\ &\quad * \text{measurement\_max\_duration} + \text{min\_battery} \\ \text{uhf\_com\_threshold} &= (\text{mean\_uhf\_com\_consumption} - \text{mean\_solar\_input}) \\ &\quad * \text{max\_uhf\_com\_duration} + \text{min\_battery} \\ \text{xband\_com\_threshold} &= (\text{mean\_xband\_com\_consumption} - \text{mean\_solar\_input}) \\ &\quad * \text{max\_xband\_com\_duration} + \text{min\_battery} \end{aligned}$$

The values for each variable in these formulas are documented in the "parameters.xlsx" file available in the *docs/* folder of this project's GitHub repository.

4.2.4. SAFE Mode Trigger for Low Battery Level

As detailed in Section 2.3.5, SAFE mode is triggered when the battery energy level falls below a user-specified threshold (defined by default as 20% of the battery capacity). Safe mode remains active until the battery level rises above this threshold.

4.3. OBC

The CHESS On-Board Computer subsystem comprises two computers, each featuring 10 GB of eMMC Flash memory for mass storage. The design ensures that only one computer operates at a time, while the other remains as a backup to take over if the primary computer fails.

In the simulation, the `Obc` class handles data storage management through an instance of the `DataStorage` class, generates housekeeping data, and manages SAFE mode triggers. However, it does not simulate the execution of flight software directly.

4.3.1. Data Management

The `DataStorage` class monitors the amount of data stored, accounting for data generated by the `Payload` and transmitted to ground stations by `Telecom`. The system handles three types of data:

- **Housekeeping data**: Always generated by `Obc`
- **CubeSatTOF data**: Generated during measurement sessions by `Payload`
- **GNSS data**: Generated in all operating modes except SAFE mode by `Payload`

In practice, GNSS and CubeSatTOF data are grouped together since both are transmitted to the ground station when in XBAND-COM operating mode.

The class also determines how much data to downlink at the start of an XBAND-COM communication window. This avoids continuous GNSS data overwhelming XBAND-COM mode, hence allowing sufficient time for UHF communication.

Housekeeping data is added continuously using a constant data rate. While in reality HK data is typically generated at specific intervals of time, this continuous approach avoids complications arising from the user's choice of simulation timestep.

The `Obc` updates the storage via the `DataStorage` class based on information provided by the `Spacecraft` about the scientific and hoursekeeping data generated or transmitted.

4.3.2. Safe Flag Handling

`Obc` is responsible for generating a safe flag when an issue arises in any subsystem. It collects a `safe_flag` boolean variable from each subsystem which are subsequently used by `ModeSwitch` to determine whether or not to switch to SAFE mode.

4.4. Payload

The GNSS and the CubeSatTOF are the two payloads of the CHESS CubeSat. In the simulation, the `Payload` class is responsible for managing these two subsystems' measurements. It maintains attributes related to measurements, such as data generation rates for GNSS and CubeSatTOF, and parameters for timing measurement pre-conditioning, post-conditioning, and duration (the measurement windows have a specific time as defined in the power budget). It also contains the related functions used by `ModeSwitch` to initiate or terminate a measurement session.

The key responsibility of `Payload` is to calculate the amount of data generated at each timestep and pass it to the `Obc`. Data generation is handled as follows:

- **GNSS data**: Added continuously in all modes except SAFE mode
- **CubeSatTOF data**: Generated during MEASUREMENT mode, starting after pre-conditioning and ending before post-conditioning

4.5. Telecom

The CHESS CubeSat's telecommunications capabilities are provided by two subsystems. The UHF subsystem consists of a UHF transceiver and a UHF antenna, while the X-band subsystem includes an X-band transmitter and an X-band antenna. The `Telecom` class is responsible for managing the spacecraft's communication operations of these two subsystems.

4.5.1. Data Rates

Data transmission operates as follows:

- During XBAND-COM mode, scientific (CubeSatTOF and GNSS) data is downlinked to the ground station
- During UHF-COM mode, housekeeping data is transmitted to the ground station

This implementation enables to calculate a “data rate” budget, assuming perfect communication between the antenna and the ground station, rather than a “link” budget. Future versions will incorporate more detailed link budget calculations.

4.5.2. Safe Mode Trigger for Communication Gaps

As outlined in Section 2.3.5, SAFE mode is triggered if there is no communication for more than 1 day. This duration acts as an upper limit until a specific value is provided by the EST, and users can modify this value in the “spacecraft.json” configuration file. The SAFE mode termination occurs at the next communication window. This choice was based on the trigger type since not yet formally defined by the team.

4.5.3. Safe Mode Trigger From Ground Station

The `Telecom` class also includes a module for handling commands sent from the ground station to initiate SAFE mode.

Concretely, these commands can be specified by the user in the “mission_design.json” configuration file under the `"uplink_safe_mode"` key before starting the simulation. Each command includes the visibility window to send the SAFE mode flag and the conditions for resolution. Two options are implemented:

- **Duration-based resolution**: SAFE mode resolves after a specific duration. This simulates scenarios where the ground station anticipates an event requiring satellite preservation for a set period.
- **Communication window-based resolution**: SAFE mode resolves at a subsequent communication window. This is for cases where the ground station sends a SAFE mode trigger with indefinite duration and later sends a command to terminate it.

Here is a configuration example:

```
"uplink_safe_mode": {
  "1": {
    "resolve_type": "com_window",
    "resolve_value": 2
  },
  "3": {
    "resolve_type": "duration",
    "resolve_value": 3600
  }
}
```

This configuration simulates:
- A SAFE mode trigger during the first visibility window, resolved in the next visibility window
- Another SAFE mode trigger during the third visibility window, resolved after 1 hour (3600 seconds).

This acts as a simplified SAFE mode implementation, designed to provide basic functionality. It will need to be updated when uplink communications are implemented and SAFE mode triggers are better defined.

5. Simulation Results

Unless mentioned otherwise, simulations are run with the default parameters gathered in the "parameters.xlsx" file in the *docs/* folder. This section demonstrates how the framework can be used for mission design. In particular, the last part shows a case study that was performed in order to evaluate the safety of shutting ADCS down during SAFE mode.

5.1. Runtimes

Table 5.1 presents the simulation runtimes for various simulation lengths, timesteps, and whether subsystem updates are performed. When focusing on propagation attributes only, such as for lifetime analysis, the simulation can be run without updating subsystems at each step. This significantly reduces runtime and can be configured in the "simulation" configuration file. As shown in Table 5.1, longer simulations are typically used for propagation studies, while shorter simulations are more suitable for operating mode and subsystem evolution analysis. We observe that simulations with subsystem updates are approximately 4 times longer than simulations with propagation only.

Simulation Duration	**Timestep**	**Runtime WITHOUT subsystem update**	**Runtime WITH subsystem update**
1 year	$\Delta t = 600s$	$\mathbf{174.40s}$	\
15 days	$\Delta t = 10s$	289.6s	$1127.9s$
1 day	$\Delta t = 10s$	19.5s	$\mathbf{79.79s}$

Table 5.1: Simulation durations

5.2. Analysis of a Short Simulation

The simulation is ran for 1 day, and as discussed in Section 3.2.1, we use a time step of $\Delta t = 10s$. We simulate a SAFE mode trigger for 1 hour sent by the ground station at the 4^{th} communication window.

The first result is the main dashboard, shown in Figure 5.1. This visualization provides a clear picture of the operating modes the spacecraft cycles during the day. Generally, there is one measurement session per day. Afterward, the satellite alternates between IDLE mode when in eclipse or fully charged, and CHARGING mode otherwise (yellow indicates sunlight exposure). When a visibility window (in pink) opens, the satellite switches to UHF-COM mode and then XBAND-COM mode to downlink scientific data. Once the science data is transmitted, it reverts to UHF-COM mode to downlink any remaining housekeeping data before returning to IDLE mode. At the 4^{th} communication window, SAFE mode is activated for 1 hour as planned.

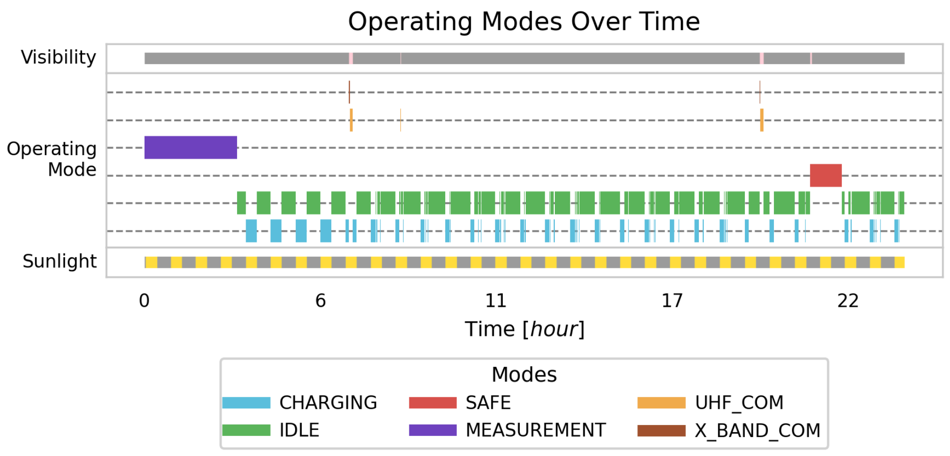


Figure 5.1: Dashboard for a simulation of 1 day

The evolution of the battery energy level is shown in Figure 5.2. We can clearly observe the energy drop during the measurement session. Afterward, the battery level fluctuates with periodic increases and decreases corresponding to the transitions between eclipse and sunlight phases.

The data storage evolution over time is plotted in Figure 5.3, showing the accumulation of housekeeping data (left), and scientific data from CubeSatTOF and GNSS (right). We observe that housekeeping data is generated at a constant rate and resets to zero each time UHF-COM mode is activated, which corresponds to visibility windows. For scientific data, there are two distinct rates: CubeSatTOF generates data at $0.111\,\mathrm{mB\,s^{-1}}$ while GNSS data is produced at $0.006\,50\,\mathrm{mB\,s^{-1}}$. Initially, the data generation rate is high due to the satellite being in MEASUREMENT mode, where both CubeSatTOF and GNSS are active. After the measurement phase, only GNSS continues to collect data, leading to a slower rate of data accumulation. Finally, during SAFE mode, neither the GNSS nor the CubeSatTOF generate scientific data: the data storage stays constant.

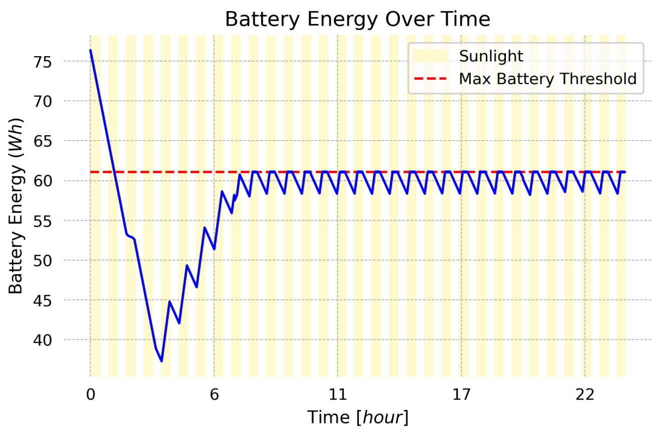


Figure 5.2: Battery energy level evolution over a simulation of 1 day

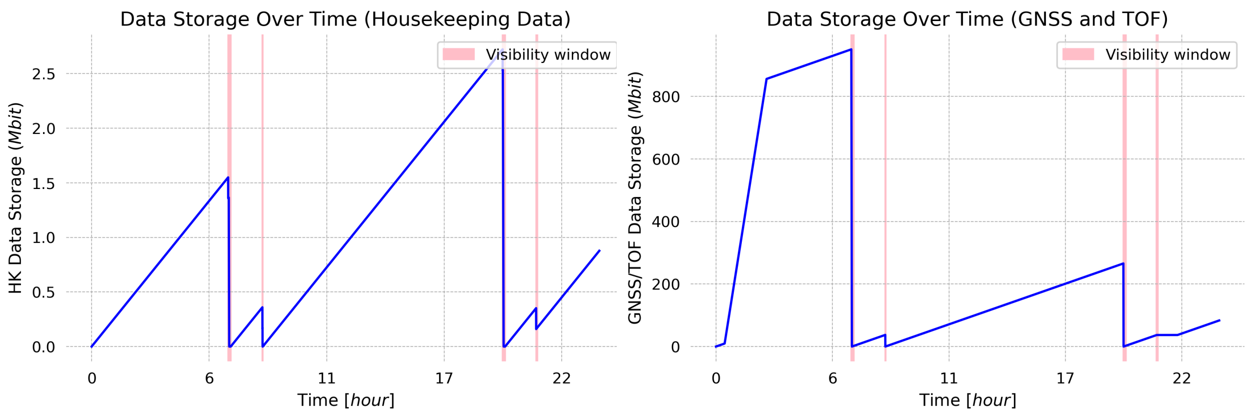


Figure 5.3: Data storage evolution over a simulation of 1 day

Finally, the pie chart in Figure 5.5 provides a view of the percentage of time the spacecraft spends in each operating mode. From the chart, we can observe that the highest percentage is for IDLE mode (61.6%).

A potential explanation is that the spacecraft operates with a limitation on the number of measurement windows (by default 1 per day). This constraint helps to prevent the battery from draining too quickly. In addition, the other modes which consume the most battery, XBAND-COM and UHF-COM, are not active for long durations (Figure 5.1). Hence, the battery remains close to its maximum allowed value and frequent recharging isn't necessary. This explains why CHARGING mode is not more often used.

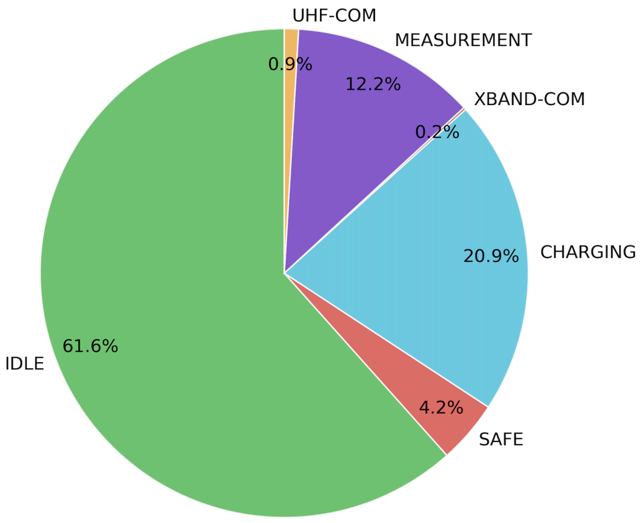


Figure 5.5: Percentages in each operating mode for a simulation of 1 day

5.3. Statistics Over a 15 Days Simulation

5.3.1. Telecom Analysis

Figure 5.6 presents an analysis of visibility windows during a 15-day simulation. The left panel shows that the average pass (also called visibility window) duration is approximately 340.2 seconds, with 75% of passes exceeding 300 seconds. In contrast, the right panel provides statistics on gap durations, revealing a broader distribution. Gap durations typically fall into two main ranges: between 0 and 2 hours or between 8 and 14 hours, resulting in an average gap duration of approximately 6.4 hours.

This analysis is valuable for making informed predictions and decisions, particularly in areas like data rate budgeting, ground station optimization, and ensuring that the communication opportunities meet mission requirements. We also present related statistics in Table 5.2.

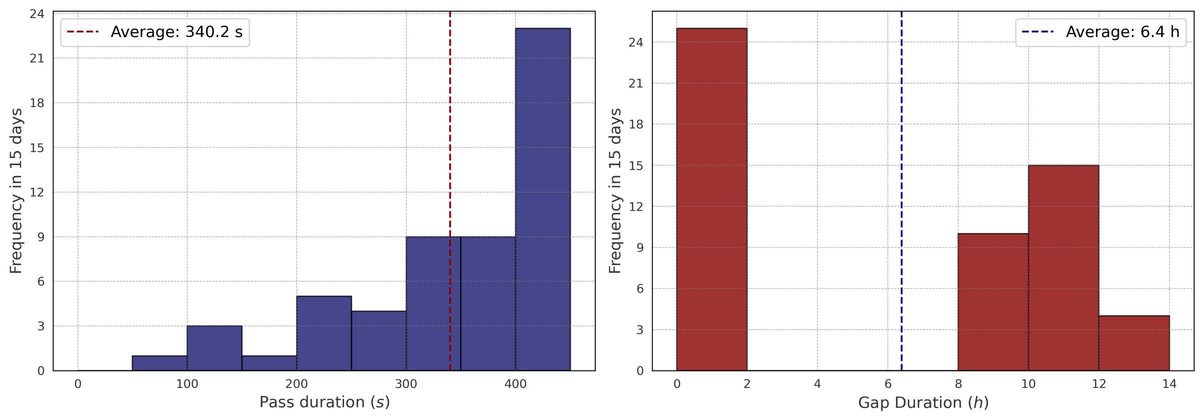


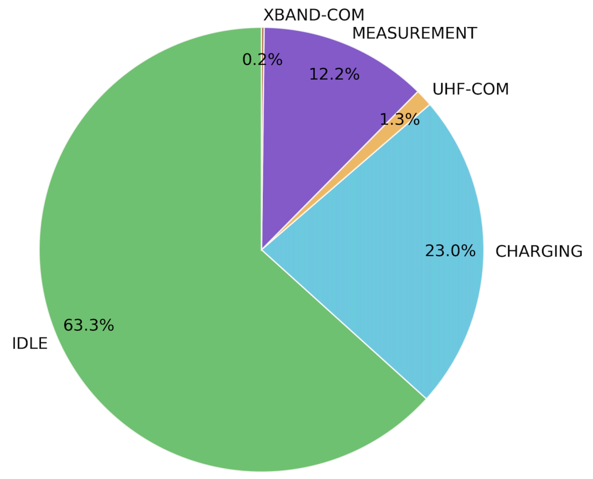
Figure 5.6: Frequency of pass and gap durations

Average visibility time per day	20min 47s
Average number of visibility windows per day	3.67

Table 5.2: Telecom statistics

5.3.2. Operating Modes Analysis

Figure 5.7 present the percentages spent in each operating mode and their corresponding average daily durations. We observe that that results are similar to the 1-day simulation except for SAFE mode, which is not simulated here. This consistency confirms that the 1-day simulation provides a representative approximation of the spacecraft's operational behavior over longer durations.



Operating mode	Average daily time
IDLE	15h 11min 28s
CHARGING	5h 31min 44s
SAFE	0s
UHF-COM	18min 9s
XBAND-COM	2min 37s
MEASUREMENT	2h 55m 50s

Figure 5.7: Operating mode statistics

5.3.3. Eps Analysis

We additionally perform an analysis of the battery energy, calculating the minimum and maximum battery energy levels per day. We observe in Figure 5.9 that the lower threshold of 15.264 W h (20% of battery capacity) and the upper threshold of 61.056 W h (80% of battery capacity) are not crossed, except initially since the battery energy level is initialized with the battery capacity.

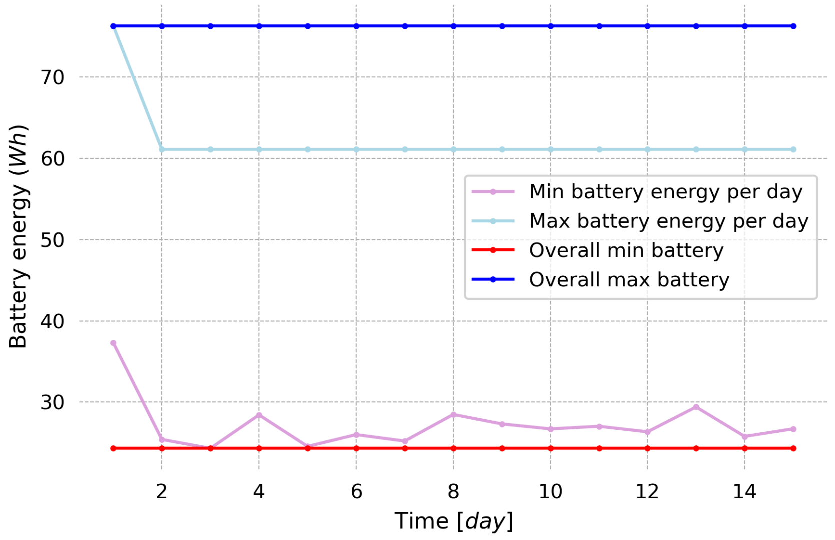


Figure 5.9: Battery energy level statistics

5.4. Case Study: ADCS Subsystem Necessity During SAFE Mode

We study the influence of disabling the ADCS subsystem during SAFE mode. Currently, the average consumption during SAFE mode is notably high, which is a concern. Indeed, during the last EST review (December 2024), reviewers strongly recommended shutting down the ADCS in this mode. If disabled, the satellite would naturally align its principal axis (Z) with the gravity gradient over time. This results in two possible orientations for the solar panels:

- Towards the center of Earth (nadir-pointing).
- Opposite to the center of Earth (opposite nadir-pointing).

One potential issue with this configuration is that the satellite's ability to charge could be compromised, as the solar panels would no longer be oriented towards the Sun. The key question we address is:

How long can the satellite sustain itself without the ADCS, in both configurations?

Specifically, we explore how this configuration affects the battery's energy level over time. Since we do not simulate the satellite's attitude change directly, we assume it updates in one of these two configurations instantaneously.

Simulation set-up

The simulation begins with the satellite in IDLE mode, transitioning to SAFE mode upon receiving a command from the ground station during the first communication window. To optimize battery state-of-charge at SAFE mode entry, the number of measurement windows is set to zero, ensuring continuous charging until the transition. This scenario assumes a best-case starting battery level of 80%. The main simulation dashboard is shown in Figure 5.10.

The study evaluates three configurations:

- ADCS active: sun-pointing attitude (optimal power generation)
- ADCS inactive: opposite nadir-pointing attitude
- ADCS inactive: nadir-pointing attitude (no power generation)

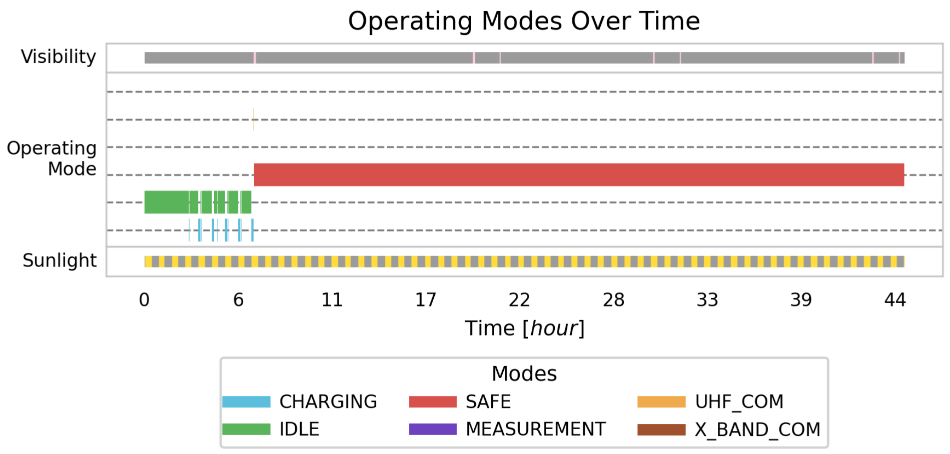


Figure 5.10: Main dashboard for the SAFE mode case study

The modularity of the simulation framework makes the configuration of this case study straightforward:

- Add the `uplink_safe_mode` command in the "mission_design.json" configuration file to simulate a ground station-triggered SAFE mode. Choose the indeterminate option (Section 4.5.3).
- Set `nb_measurements_per_day` to 0 in the "spacecraft.json" configuration file.
- Set the ADCS consumption rate to 0 for SAFE mode in the "spacecraft.json" configuration file.
- Update `attitude_mode_dict` in `constants.py` to set nadir-pointing attitude for SAFE mode. For the opposite nadir-pointing orientation, modify the solar panel class directly, as this specific attitude is not predefined.

Results are presented in Figure 5.11.

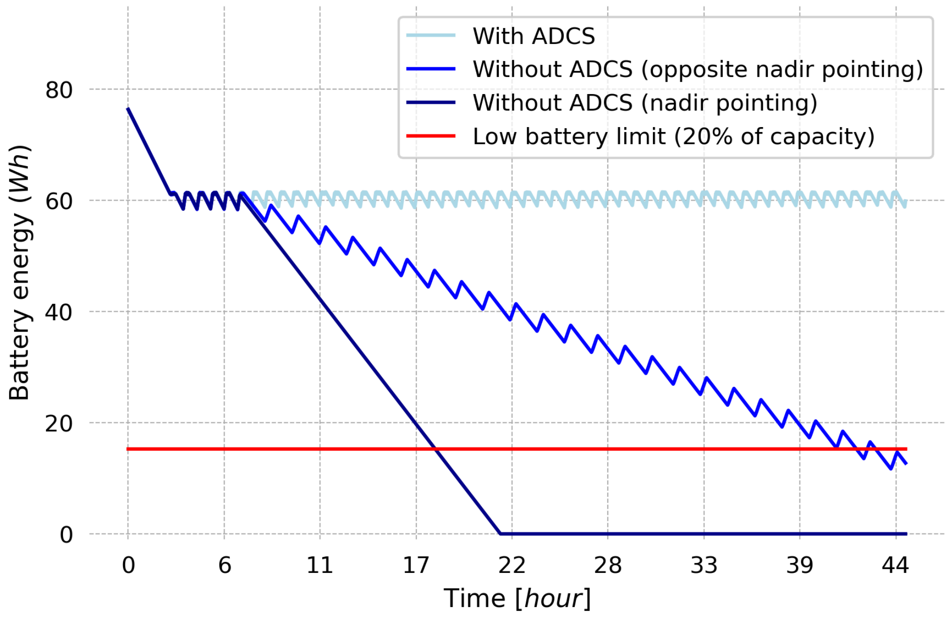


Figure 5.11: Evolution of battery energy for different scenarios

When ADCS remains active, the satellite maintains full battery charge due to optimal power generation. Conversely, disabling ADCS reduces energy consumption but leads to rapid battery energy decrease. In particular:

- Nadir-pointing attitude results in no power generation, causing the battery to reach the critical threshold in 11h 18m 59s.
- Opposite nadir-pointing attitude provides partial power generation, extending battery life to 35h 42m 0s.

This analysis informs mission planning by quantifying survival times under SAFE mode without ADCS subsystem active. Future investigations could explore SAFE mode initiation with lower initial battery levels, quantify the time taken by the satellite to reach the two configurations, and estimate the probabilities of each orientation to refine risk assessment.

6. Next Steps

Over the course of the semester, significant effort was dedicated to identifying features that would make this tool most useful. Below is a (non-exhaustive) list of recommended next steps to guide the subsequent development of the project:

- **Develop a Graphical User Interface (GUI)**: Implement a user-friendly interface, potentially using *Tkinter*, one of the most widely adopted GUI libraries in Python.
- **Enable user inputs to support digital twin functionality**. For example:
  - Allow users to overwrite satellite positions at specific times, using GNSS data to correct propagator errors.
  - Provide the capability to update solar panel efficiency dynamically in the event of damage or degradation.
- **Improve the propagator**
  - Incorporate gravitational influences from the Moon, Jupiter, and the Sun.
  - Simulate lunar eclipses.
  - Validate results using additional tools such as NASA test cases (NESC), STK software, SGP4 algorithm, or SwissCube data, if available.
  - Add support for dynamic cross-section variations (dynamic drag propagator).
- **Resolve timestep limitations**: Some subsystem updates, such as those for visibility windows, require very small timesteps, which can make simulations computationally expensive. To address this, separate the timestep for subsystem updates (requiring high precision) from propagation (already optimized by the Poliastro library).
- **Simulate the thermal environment**: Extend the simulation to account for material properties of the satellite's outer structure and solar panels, to understand the thermal behavior and get a better approximation of power generation. For additional details on implementing a temperature model for a digital twin, refer to [10].
- **Add varying solar flux**: The current model uses a constant value of $1\,367\,\mathrm{W\,m^{-2}}$, but solar flux varies across solar cycles and throughout the year due to Earth's elliptical orbit [11]. Future updates could integrate a dynamic solar flux model to improve accuracy.
- **Model solar panel efficiency degradation**: Currently, the tool assumes a constant efficiency of 0.25, which is a standard value. Adding a degradation model based on manufacturer data or predictive analysis would enable to have a more realistic simulation.
- **Enhance telecom modeling**: Develop a more comprehensive telecom model, including support for uplink messages, definition of communication standards, and implementation of a link budget (considering transmission losses). Additionally, introduce conditions to terminate UHF communication earlier than the default if needed, even if the satellite remains visible.
- **Introduce dynamic thresholds**: For instance, adjust measurement thresholds based on available data space. Similarly, add mechanisms to monitor battery levels continuously, with operations terminating automatically if critical thresholds are reached.
- **Complete SAFE mode simulation**: Implement additional SAFE mode triggers. For that, the spacecraft team needs to define the confitions for transitioning into SAFE mode as well as for terminating it more clearly.
- **Simulate LEOP and end-of-life scenarios**: Extend the tool to address these phases.
- **Add non-continuous housekeeping data generation and beacons**: Currently, housekeeping data is modeled as continuously generated due to the timestep challenges outlined earlier. However, in reality, such data is produced at defined intervals. Similarly, beacon transmissions, containing critical data, occur periodically and should be represented.

7. Conclusion

This project provides a flexible and accessible framework for simulating the CHESS CubeSat mission and developing its digital twin. Using Python and available libraries like *Poliastro* and *Astropy*, a modular code was produced to support detailed analyses across different subsystems for mission planning while remaining user-friendly.

The simulations revealed important insights regarding:

- **Safe mode power consumption**: Disabling ADCS during SAFE mode saves energy but introduces risk depending on how the satellite aligns with the gravity gradient. We quantified how long the battery can last in both nadir and opposite nadir orientations.
- **Energy management**: Careful control of operations, like limiting measurement sessions, is effective at maintaining a sustainable battery charge.
- **Communication**: Analyzing visibility windows and gaps offers useful data for planning downlink sessions and optimizing communication time.

In short, this framework is a big step forward for building a digital twin of the CHESS CubeSat. It already helps with mission planning and risk assessment, and future updates will make it even better at predicting performance and supporting real-time decisions.

Bibliography

[1] European Space Agency, “Model Based for System Engineering, Technology Harmonisation Dossier,” 2024.

[2] D. A. Vallado and W. D. McClain, *Fundamentals of astrodynamics and applications*, 3rd ed. in Space technology library vol. 21. Hawthorne, CA: Microcosm Press, 2007.

[3] Y. Xie, Y. Lei, J. Guo, and B. Meng, *Spacecraft Dynamics and Control.* in Space Science and Technologies. Singapore: Springer Singapore Pte. Limited, 2021.

[4] P. L. Sheridan, S. N. Paul, G. Avendaño-Franco, and P. M. Mehta, “Updates and improvements to the satellite drag coefficient Response Surface Modeling toolkit,” *Advances in Space Research*, vol. 69, no. 10, pp. 3828–3846, 2022.

[5] V. U. Nwankwo *et al.*, “Atmospheric drag effects on modelled low Earth orbit (LEO) satellites during the July 2000 Bastille Day event in contrast to an interval of geomagnetically quiet conditions,” in *Annales Geophysicae*, 2021, pp. 397–412.

[6] W. de Vries, “Cubesat Drag Calculations,” 2010.

[7] A. Muller, “State-of-the-art study of Python libraries for orbit propagation: Comparison of poliastro & Tudatpy with CelestLab & GMAT,” 2022, [Online]. Available: https://github.com/Nosudrum/orbit-propagation-python-study

[8] V. Braun *et al.*, “Probabilistic Orbit Lifetime Assessment with OSCAR,” in *Proceedings of the 6th International Conference on Astrodynamics Tools and Techniques*, 2016.

[9] P. Escobal, *Methods of orbit determination.* 1985.

[10] R. H. Beyond gravity, “Physics documentation for the Beyond Gravity ditital twin.” 2024.

[11] M. Iqbal, *An introduction to solar radiation.* Elsevier, 2012.

Appendix

A Mode Switch Decision Tree

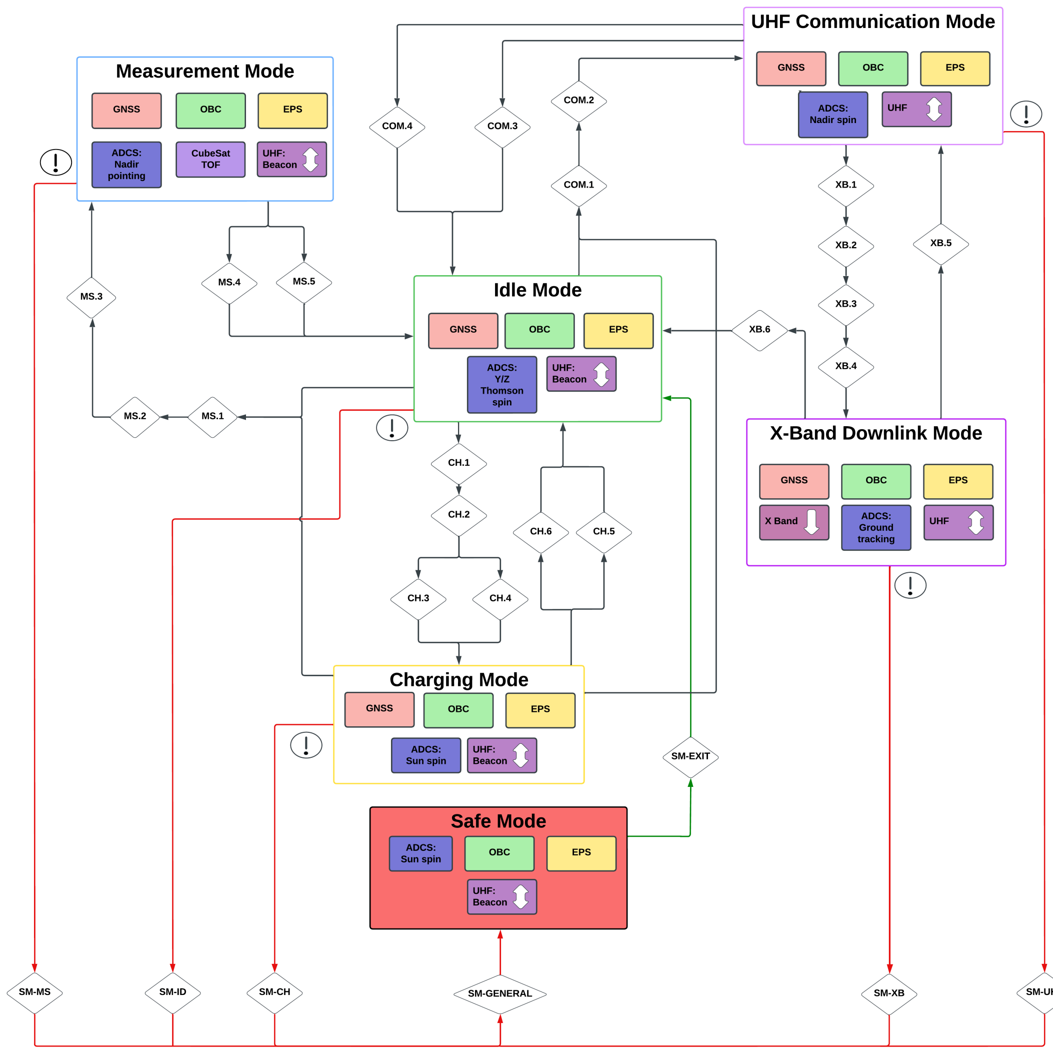


Figure A.1: Mode switch decision tree

The legend for the diagram can be found in Table A.1.

Reference	**Description**
CH.1	The CubeSat is currently not in an eclipse
CH.2	Battery charge below “Full Battery threshold”
CH.3	No other scheduled activities or tasks to finish
CH.4	Battery charge below “Low Battery threshold”
CH.5	Battery charge above “Full Battery threshold”
CH.6	The CubeSat is entering an eclipse
COM.1	Battery charge above “Communication Charge threshold”
COM.2	Communication window incoming
COM.3	Communication window finished
COM.4	Communication session finished
XB.1	Battery charge above “X-band Downlink Charge threshold”
XB.2	Handshake with ground station, request for scientific data downlink
XB.3	Satellite is still in a communication window
XB.4	There is data to downlink
XB.5	Downlink of scientific data completed
XB.6	Communication window finished
MS.1	Battery charge above “Measurement Charge threshold”
MS.2	Scheduled measurement session incoming
MS.3	Science data storage not full
MS.4	Measurement completed
MS.5	Science data storage full
SM-CH	SAFE flag raised. during CHARGING mode
SM-UHF	SAFE flag raised during UHF-COM mode
SM-XB	SAFE flag raised during XBAND-COM mode
SM-MS	SAFE flag raised during MEASUREMENT mode
SM-ID	SAFE flag raised during IDLE mode
SM-GENERAL	The detection of a SAFE flag triggers general SAFE mode
SM-EXIT	No SAFE mode condition is active anymore

Table A.1: Legend for the mode switch decision tree