UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

BACHELOR FINAL THESIS

Design of an attitude control system and flight schedule for CubeSats with NASA 42 Simulator

Document:

Report

Author:

Ramón Revilla Bouso

Director/Co-director:

Javier Gago Barrio / Juan José Alins Delgado

Degree:

Bachelor in Aerospace Technology Engineering

Examination session:

Spring 2023

Abstract

Over the last decade, CubeSats, a standardised satellite design, have opened up doors of space exploration to a wide range of universities, startups and research groups by offering low-cost, versatile operations. This thesis is within the framework of the DISEN research group and its *Amazon Mission*, which intends to establish a communication link between a remote and inaccessible place such as the Amazon, with the DISEN ground station.

The thesis aims to develop an attitude control system and flight schedule for the *Amazon Mission* CubeSat using the NASA's open-source 42 Simulator. In addition, Matlab is used to plot and analyse the simulation outputs.

The attitude control system is designed with two possible states: a ground station pointing state, to be used during a flyby over a ground station, and an Earth pointing state, to be used during the rest of the orbit. Based on this, the flight schedule calls each state at the required time, depending on the CubeSat relative position with the Earth. Furthermore, a preliminary analysis on the battery management and data flow of the CubeSat throughout its orbit is implemented.

To analyse the simulation results, data such as the pointing accuracy, the angular velocity of the reaction wheels and the battery consumption is particularly focused and detailed due to being critical factors for the performance of the CubeSat. Regarding the final design, it provides the CubeSat with a pointing accuracy below 4 degrees at all times and with a 20% battery charge surplus during each orbit. It also ensures that the operation of the reaction wheels is within limits.

With the results obtained, it can be stated that the CubeSat designed has a reliable operation and fulfills the *Amazon Mission* goals. Nevertheless, this thesis has built the mission foundations within the 42 Simulator, and it is future work to refine many of the processes and models designed to achieve even more realistic simulations.

Keywords:
CubeSat, Attitude Determination and Control System, ADCS, 42 Simulator, Earth pointing, Ground Station pointing, Battery Management System, CubeSat communication

Resumen

En la última década, los CubeSats, un diseño estandarizado de satélites, han abierto las puertas de la exploración espacial a un gran número de universidades, startups y grupos de investigación al ofrecer operaciones versátiles y de bajo coste. Esta tesis se enmarca dentro del grupo de investigación DISEN y su *Misión Amazonas*, la cual pretende establecer un enlace de comunicación entre un lugar remoto e inaccesible como es el Amazonas, con la estación terrestre del grupo DISEN.

La tesis tiene como objetivo desarrollar un sistema de control de actitud y un programa de vuelo para el CubeSat de la *Misión Amazonas* utilizando el Simulador 42 de código abierto de la NASA. Además, se utiliza Matlab para graficar y analizar los resultados de la simulación.

El sistema de control de actitud se diseña con dos estados posibles: un estado de apuntamiento a la estación terrestre, que se utilizará durante el sobrevuelo de una de ellas, y un estado de apuntamiento a la Tierra, que se utilizará durante el resto de la órbita. Basándose en esto, el programa de vuelo llama a cada estado en el momento adecuado dependiendo de la posición relativa del CubeSat con la Tierra. Además, se implementa un análisis preliminar sobre la gestión de la batería y el flujo de datos del CubeSat a lo largo de su órbita.

Para analizar los resultados de la simulación, se presta especial atención a la precisión de apuntamiento, la velocidad angular de las ruedas de reacción y el consumo de batería, por ser factores críticos para el rendimiento del CubeSat. En cuanto al diseño final, este proporciona al CubeSat una precisión de apuntamiento inferior a 4 grados en todo momento y con un excedente de carga de batería del 20% en cada órbita. También asegura que el funcionamiento de las ruedas de reacción esté dentro de sus límites.

Con los resultados obtenidos, se puede afirmar que el CubeSat diseñado tiene un funcionamiento estable y cumple con los objetivos de la *Misión Amazonas*. No obstante, en esta tesis se han construido las bases de dicha misión dentro del Simulador 42, y es trabajo futuro perfeccionar muchos de los procesos y modelos diseñados para lograr simulaciones aún más realistas.

Palabras clave:
CubeSat, Sistema de Control y Determinación de Actitud, ADCS, Simulador 42, Apuntamiento a Tierra, Apuntamiento a Estación Terrestre, Sistema de Gestión de Baterías, Comunicación del CubeSat

Declaration of Honour

I, RAMÓN REVILLA BOUSO, declare that this thesis titled, DESIGN OF AN ATTITUDE CONTROL SYSTEM AND FLIGHT SCHEDULE FOR CUBESATS WITH NASA 42 SIMULATOR, and the work presented in it are my own. I confirm that:

the work in this Bachelor Thesis is completely my own work,

no part of this Bachelor Thesis is taken from other people's work without giving them credit,

all references have been clearly cited,

I am authorised to make use of the information related to the DISEN research group that I am providing in this document.

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by the *Universitat Politècnica de Catalunya - BarcelonaTECH.*

Signature:

Date: 21st June 2023

Acknowledgements

I would like to express my most sincere gratitude to my thesis director and co-director, Javier Gago and Juan José Alins, for their guidance and expertise throughout the entire research process. Their feedback and support has been fundamental in the development and quality of this thesis.

Additionally, I would like to deeply thank my family and friends for their support and encouragement during these four academic years.

Contents

List of Figures

List of Tables

List of Abbreviations

Abbreviation	Meaning
ADCS	Attitude Determination and Control System
AMZ	Amazon Ground Station
BMS	Battery Management System
cFS	core Flight System
DCM	Direct Cosine Matrix
DISEN	Distributed Sensor Networks
ECEF	Earth-Centered Earth-Fixed
ECI	Earth-Centered Inertial
GPS	Global Positioning System
GUI	Graphical User Interface
HITL	Hardware in the Loop
NOAA	National Oceanic and Atmospheric Administration
OSK	OpenSatKit
TDRS	Tracking and Data Relay Satellite
TGC	Terrassa Ground Control
USAF	U.S. Air Force

Chapter 1

Introduction

1.1 Aim

The purpose of this thesis is to develop, using the 42 Simulator, an attitude control system as well as a flight schedule for the *Amazon Mission* CubeSat, which needs to be able to point at Earth and ground stations at any time.

1.2 Scope

On one hand, this thesis will include:

- In-depth study of 42 Simulator.
- Design of the mission flight schedule.
- Description of the mission specifications.
- Description of the CubeSat specifications.
- Design and development of the pointing techniques in the 42 environment.
- Design and development of the attitude control system in the 42 environment.
- Design and development of the flight schedule in the 42 environment.
- Design and development of a preliminary battery management system in the 42 environment.
- Design and development of the data flow between the CubeSat and the ground stations in the 42 environment.
- Testing of the 42 flight software designed.
- Analysis and visualization of the results through 42 GUI and Matlab.

On the other hand, this thesis will not include:

- In-depth study of COSMOS platform.
- In-depth study of cFS platform.
- Mission programming in the cFS environment.
- Any other aspect not mentioned in the included items list.

1.3 Requirements

The following are the basic specifications and constraints that the final solution must have:

- The CubeSat must follow a Low Earth Orbit.
- The CubeSat must follow an orbit that allows it to periodically flyby TGC and AMZ ground stations.
- The CubeSat must point accurately to the ground stations at the proper time to communicate correctly.
- The CubeSat must point at Earth's center when its not pointing to a ground station.
- The CubeSat must be designed to be able to perform successfully all of its actions.
- The CubeSat must be designed to be fully autonomous.
- NASA 42 Simulator must be used.

1.4 Justification

CubeSats are a class of satellites that have emerged as a promising solution to perform space missions with low cost and short orbital periods. To be classified as such, satellites must meet specific criteria in terms of shape, size and weight.

In 2021, the CubeSat market had an estimated value of 250M$ and is expected to grow almost +300% in the next 8 years [1]. This huge popularity, mainly in the commercial, academic and government sectors, has been gained due to their reduced costs, standardized design, and ability to carry out multiple space missions such as communication, scientific experiments and Earth observation.

This thesis is developed within the framework of the *Amazon Mission* carried out by the Distributed Sensor Networks (DISEN) group. This mission intends to collect animal data from the AMZ Ground Station, such as images, videos and GPS tracking, and transmit it to the TGC. This data will allow future researchers to study in detail the Amazon wildlife behaviour and keep track of a population which, otherwise, would not be possible to analyse due to its remoteness and inaccessibility.

Many theses have been produced prior the current, each one developing a different facet of the *Amazon Mission* and the CubeSat itself. Two clear examples of them can be, in first place the design and manufacture of the reaction wheels and magnetorquers which constitute the attitude control. And in second place, the manufacture of a structure that reproduces space conditions in terms of magnetic field and free rotation, which will allow a future HITL simulation.

This thesis is focused on designing and developing an attitude control system for the CubeSat in order to accurately point at ground stations each time it flyby one. The main goal is achieved using one of the three OpenSatKit platforms, the 42 Simulator. In early stages of the mission simulation, this platform can handle all the work. However, it needs to be taken into account that the end goal of the DISEN Group for this mission, is to full implement it inside the OpenSatKit environment, using all three platforms available, and be able to perform a HITL simulation. Therefore, this thesis serves to ease the way in terms of attitude control and flight schedule of the CubeSat within the 42 Simulator, bringing the mission one step closer to its completion.

Chapter 2

State of the art

This chapter serves as an introduction to the main three points of the thesis: the CubeSats, the *Amazon Mission* and the 42 Simulator. To do so, the CubeSat concept and its standards are stated, then, two missions similar to the *Amazon Mission* are analysed, and finally, three high-level software used to develop space missions are explained.

Thereafter, more technical concepts can be introduced. Four coordinate systems are defined, along with two types of rotations to transform coordinates from one frame to another. Lastly, two commonly used controllers in CubeSat control systems are explained. With all that, valuable insights are provided for the upcoming chapters of this thesis.

2.1 CubeSat classification and evolution

A CubeSat is a scaleable satellite that complies with established shape, size and weight standards. Thanks to that, companies can mass-produce components allowing a reduced cost and time of development for universities, research groups and companies.

A standard CubeSat unit, referred to as 1U CubeSat, must be a $10 \times 10 \times 10$ cm cubic module with a maximum mass of 1.33 kg. Multiple units can be assembled resulting in a larger CubeSat which can be referred to as a 2U, 3U or 6U among others, depending on the number of units involved.

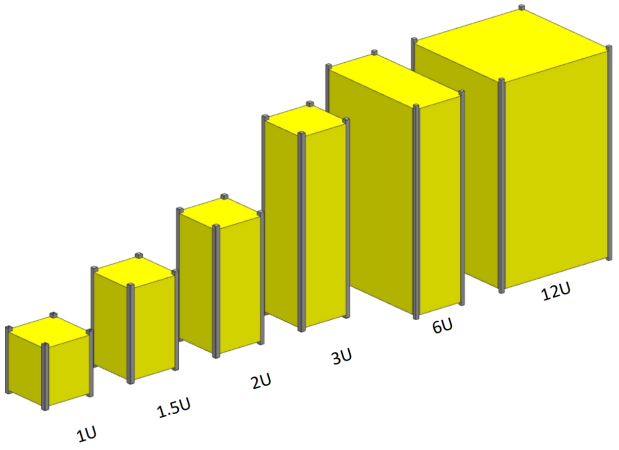


Figure 2.1: Most common CubeSat stacking options. Source: [2]

Looking at the total amount of nanosatellites[1] and CubeSats launched over the years, the value has not stopped increasing since the beginning. Following an exponential growth since 2013, the CubeSats launches have increased by 20 times in the last 10 years, foreseeing a bright future for this business.

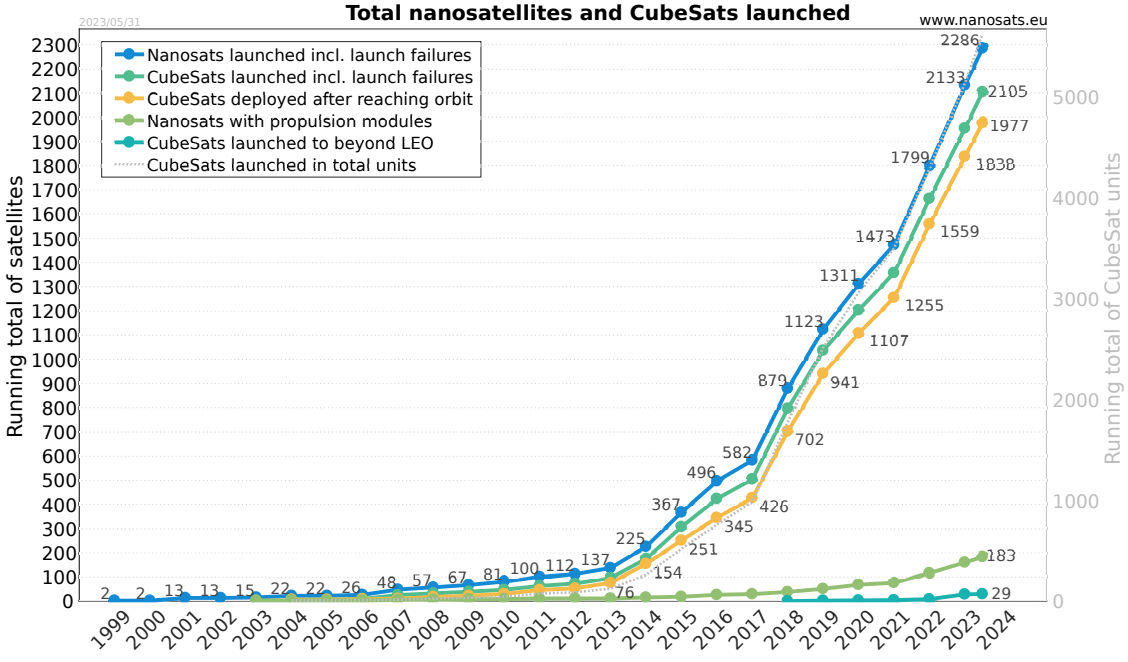


Figure 2.2: Total nanosatellites and CubeSats launched. Source: [3]

2.2 Earth observation missions

The *Amazon Mission* can be compared to Earth observation missions due to performing in a similar way. Both missions need to point at Earth's surface while orbiting as well as to maintain a Low Earth Orbit. Therefore, the *STU-2A* and *ALEASAT* missions are presented.

[1] Nanosatellites are a class of small satellites with a mass between 1 to 10 kg.

2.2.1 STU-2A

On September 25th, 2015, researchers from the University of Shangai and Beijing launched the STU-2A CubeSat. It was a 3U CubeSat which carried an observation camera for the Antarctic glacier and sea ice observation. To achieve that, the CubeSat followed a polar LEO orbit. As for the attitude determination sensors, it carried a magnetometer, five coarse sun sensors, a fine sun sensor and a star tracker, while for the actuators, it carried three magnetorquers and three reaction wheels [4].

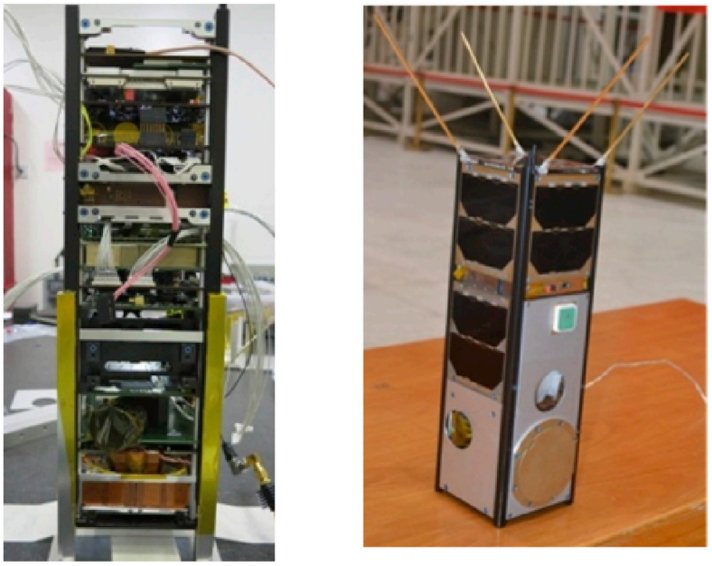


Figure 2.3: STU-2A Internal Layout (Left) and Satellite outlook (Right). Source: [4]

It is important to notice the considerable amount of sensors that needs to carry a CubeSat to ensure an efficient operation over its orbit. As conclusions, the STU-2A CubeSat achieved a ground resolution below 100 m for images covering 165×240 km.

2.2.2 ALEASAT

Expected in 2023, the Satellite Design Team of the Simon Fraser University in Canada, aims to launch a 1U CubeSat [5]. The ALEASAT purpose is to "provide on-demand satellite imagery directly to AROs[2] for disaster relief" [6]. As for the attitude determination sensors, the CubeSat will carry a gyroscope and a sun sensor. It should be noted that only two sensors will be used, which may be imposed by a tight budget or by the decision not to add more complexity to the project. However, this means that more risks will have to be taken, increasing the probability of failure.

[2]Amateur Radio Operators.

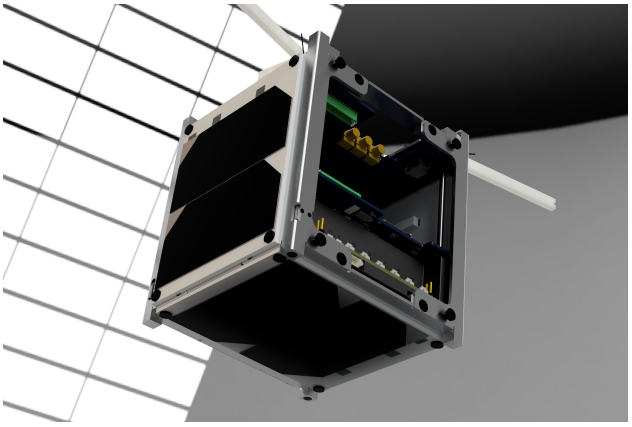


Figure 2.4: ALEASAT CubeSat 3D Model. Source: [7]

2.3 Space missions software

When designing space missions, companies require a powerful software to analyse the performance and feasibility of the mission. In this Section, three software for space missions design are explained.

2.3.1 Systems Tool Kit

The STK software, developed by *Analytical Graphics, Inc.* (an Ansys Company) allows the user to model complex systems in a 3D simulation. This software tool can create multidomain scenarios and analyse complex relationships between objects. It is a very powerful software commonly used in the aerospace sector by organizations such as NASA, Lockheed Martin and Airbus, among others. However, in this thesis STK is used at specific moments for its easiness to visualize the *Amazon Mission* orbit, both in a 3D model of the Earth and its groundtrack. It is important to highlight that STK is a private software, and thus a larger mission budget is needed in case of use.

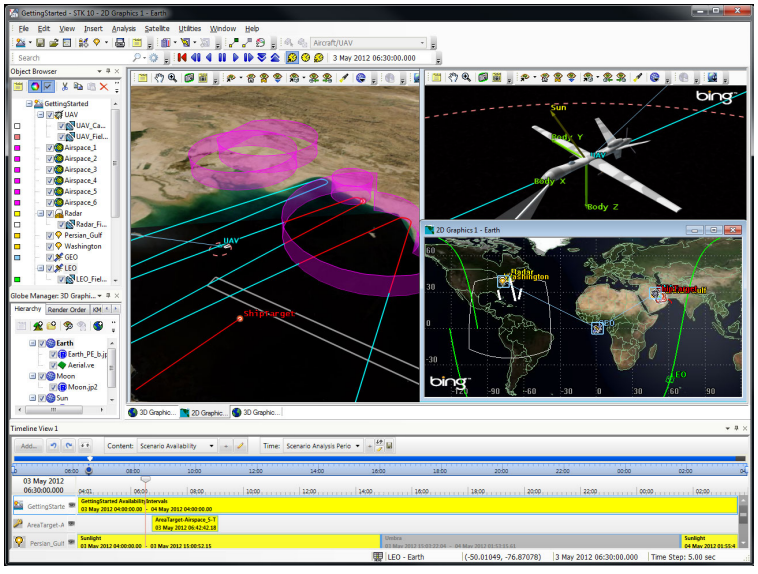


Figure 2.5: STK GUI screenshot. Source: [8]

2.3.2 FreeFlyer

The FreeFlyer software, developed by *a.i. solutions, Inc.* can support the full lifecycle of a space mission, taking it through the design, analysis and operations phase. It is one of the most powerful tools for solving astrodynamics problems with an extensive heritage on over 250 missions, partnering with NASA, NOAA and USAF, among others. The software provides a high level of customisation in terms of 2D and 3D visualizations of the mission, helping the user to focus on its specific critical aspects of the mission. It can support missions ranging from GND (Ground) to BEO (Beyond Earth Orbit) thanks to its extensive capabilities, which are: spacecraft propagation, coverage and contact for ground station communication, attitude modelling, maneuvering, targeting and optimization of thrust events, interplanetary support, orbit determination, and terrain analysis for ground operations. As before, it is important to highlight that FreeFlyer is a private software, therefore, a larger mission budget is needed in case of use.

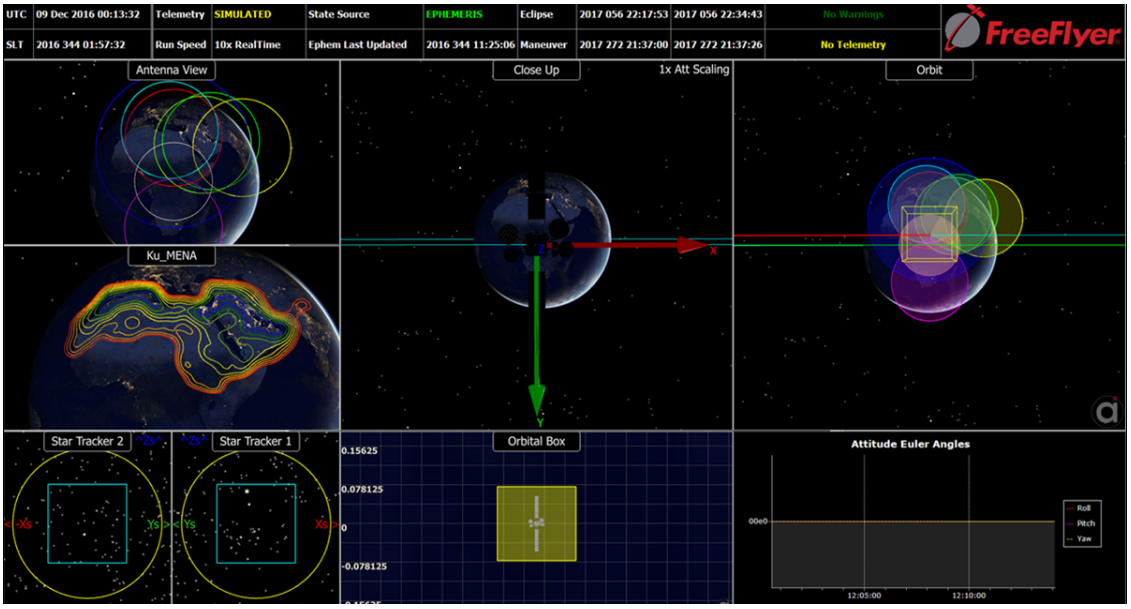


Figure 2.6: Custom mission visualization using FreeFlyer. Source: [9]

2.3.3 OpenSatKit

The OpenSatKit software consists of the combination of three open-source platforms: COSMOS Command and Control, core Flight System and 42 Simulator [10]. These three platforms constitute a software free to use which can support design and development phases of space missions. Below all three platforms are briefly explained.

COSMOS

The COSMOS platform, developed by *Ball Aerospace Corporation*, provides support to space missions development phases by being capable of sending commands and receiving data from one or multiple embedded systems [11]. Despite having lots of tools available, not all of them are used when combined with OpenSatKit (see Figure 2.7). The tools selected for the OSK environment and its purpose are summarised below:

- Launcher: Provides a graphical interface for launching each tool.
- Command and Telemetry server: For real-time commanding and telemetry processing.
- Telemetry and Packet Viewer: To customise and organise the display of multiple data.
- Command Sender: To send flight software commands using a GUI.
- Telemetry Grapher: Real-time graphing of any telemetry group of data.
- Table Manager: To edit and display binary files.
- Script and Test Runner: To develop and execute test scripts and procedures.

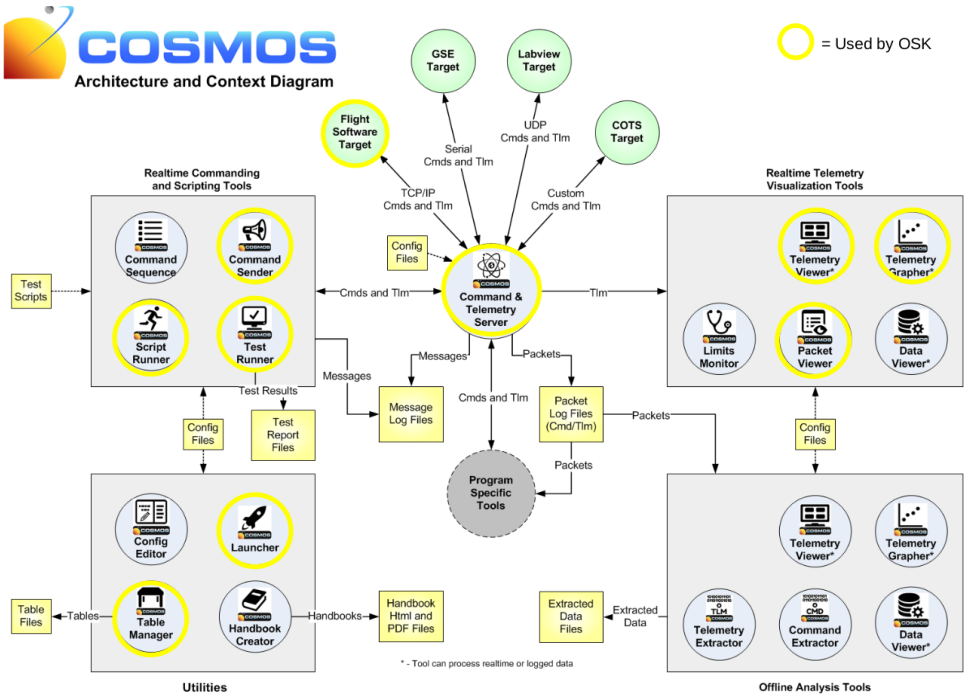


Figure 2.7: COSMOS tools used by OSK. Source: [12]

core Flight system

The core Flight System, developed by *NASA*, uses a three-tiered software architecture to provide a portable and adaptable flight software framework. The utility of having a portable framework is that it can be augmented with user-developed applications to create cFS distributions. Therefore, the software is easier to be tested and maintained across different missions and spacecraft.

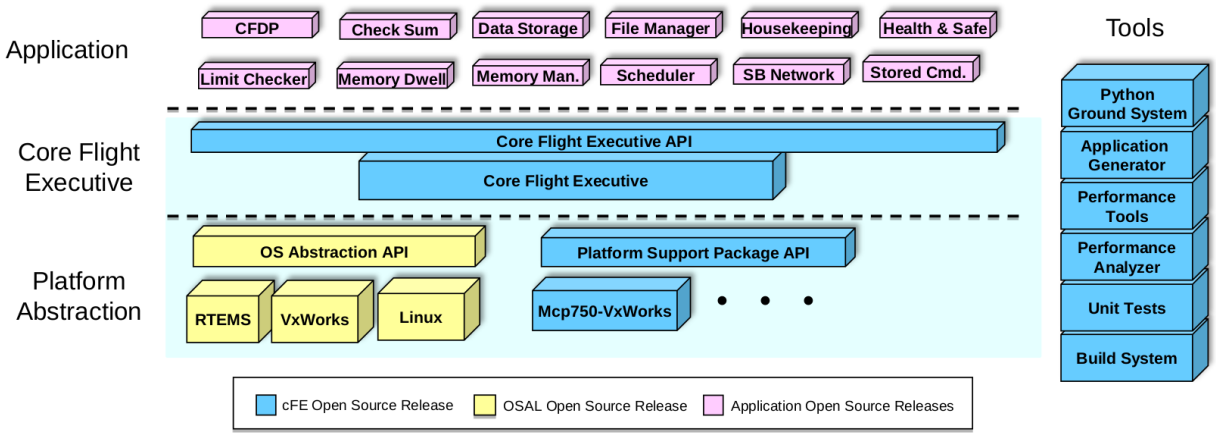


Figure 2.8: core Flight System Architecture Layers. Source: [13]

In the Figure above, the two lower layers constitute the aforementioned cFS framework. While the top layer is the user application.

42 Simulator

The 42 Simulator, developed by *NASA*, is used for simulation of spacecraft attitude and control and orbital dynamics. Its usability goes from mission concept studies and prototyping to complex flight software development. It provides high-fidelity dynamics and environmental models and easy portability and visualization. The simulator supports multiple and multi-body spacecraft anywhere in the solar system as well as formation flying. It also can communicate through sockets and externalise processes. Chapter 3 focuses on the 42 Simulator structure and operation.

2.4 Coordinate systems

Coordinate systems are the way to locate bodies in space and depending on the task at hand, a suitable frame can make the work much easier. In this Section, four common coordinate systems used in 42 Simulator are defined.

2.4.1 J2000 ECI

The $J2000$ ECI frame is a specific model of inertial frame. It is centered at the Earth's centre of mass and, as all ECI frames, has its axes fixed with respect to the distant stars. As depicted in Figure 2.9, the X axis points towards the vernal equinox[3] and the Z axis is aligned with the Earth's rotational axis, pointing towards the North Pole. Therefore, Y axis is defined following the right-hand rule. This frame is commonly used for stargazing and satellite tracking.

[3]The vernal equinox is defined as the intersection of the ecliptic plane and the celestial equator.

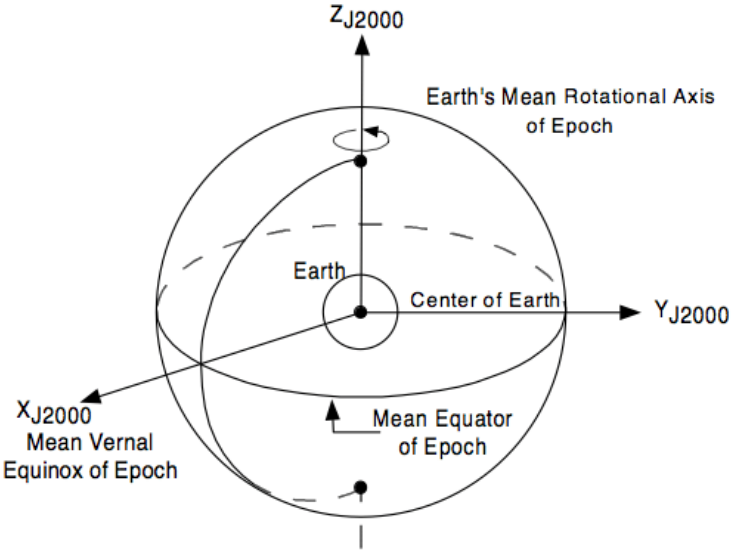


Figure 2.9: J2000 Frame representation. Source: [14]

In addition, Earth-Centered Inertial frames are those in which orbital elements are defined. Therefore, they are briefly explained below and shown in Figure 2.10.

- Inclination (i): Of the orbital plane with respect to the ecliptic plane.
- Right Ascension of Ascending Node (Ω): The angle between the vernal equinox and the point where the orbital plane crosses the ecliptic plane from south to north.
- Argument of Periapsis (ω): The angle between the ascending node and the periapsis of the orbit.
- True Anomaly (ν): The angular position of the orbiting body along its orbit at a specific point in time.

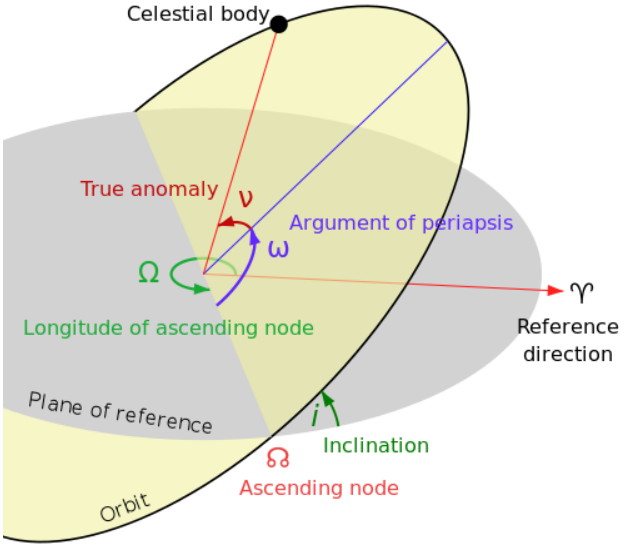


Figure 2.10: Orbital elements representation. Source: [15]

2.4.2 ECEF

The ECEF frame is centered at the Earth's centre of mass and has its axes rotating with the Earth's rotation. As depicted in Figure 2.11, the X axis points towards the intersection of the Greenwich meridian and the equator, while the Z axis is aligned with the Earth's rotational axis, pointing towards the North Pole. Therefore, Y axis is defined following the right-hand rule, pointing towards the east and lying in the equatorial plane. This frame is commonly used to express positions on or near the Earth's surface and it can be easily converted to latitude and longitude units.

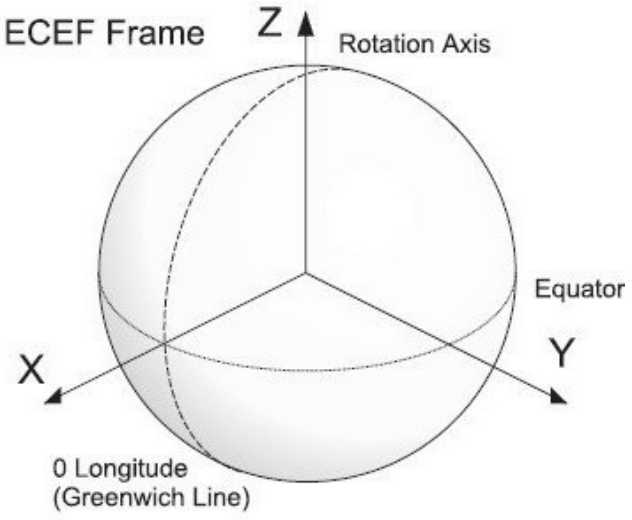


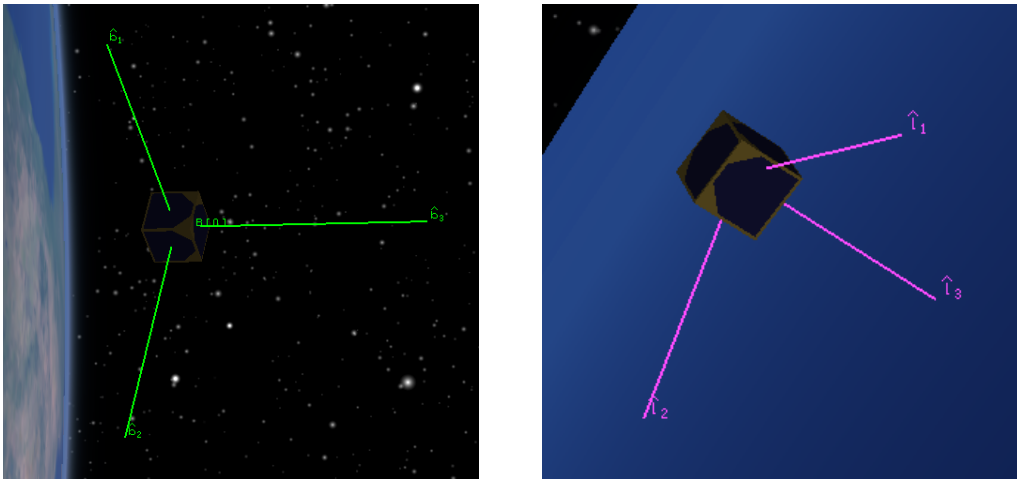
Figure 2.11: ECEF Frame representation. Source: [16]

2.4.3 Body and LVLH Frame

On one hand, a Body frame is attached to a spacecraft and its axes are aligned with the physical structure of the body. This frame moves along with the body as it rotates and translates. It is commonly used for attitude determination and control.

On the other hand, the Local Vertical, Local Horizontal frame is a local frame aligned with the orientation of a spacecraft orbiting around a celestial body. The Local Vertical axis points towards the celestial body nadir[4], while the Local Horizontal axis points in the direction of the spacecraft's velocity vector. Therefore, the remaining axis is defined following the right-hand rule, pointing in the direction of the orbit normal. This frame is commonly used to analyse and control the relative attitude and motion of a spacecraft with respect to its orbit. Screenshots of a CubeSat model inside the 42 Simulator with both frames are seen below.

[4]The nadir refers to the direction pointing directly downward from the satellite towards the Earth's surface. It represents the opposite direction of the satellite's line of sight.



(a) Body Frame representation. (b) LVLH Frame representation.

Figure 2.12: Body and LVLH frames.

2.5 Rotations

Rotations allow to transform coordinates between reference frames, which is a useful method to simplify many problems. There are several types of rotations, but only the two mainly used in this thesis are discussed in this Section.

2.5.1 Euler angles

Euler angles consist of three rotations around axis in a starting frame, resulting in a newer rotated frame. The rotations must be always around the frame axis, however, the order and repeatability can vary, resulting in a total of 24 variants.

The strengths of this type of rotations, is being intuitive and having only one parameter per degree of freedom. But it also has its weaknesses due to high quantity of variants and singularities such as the *gimbal lock*[5]. Below, a transformation example from an arbitrary N frame to a B frame is shown.

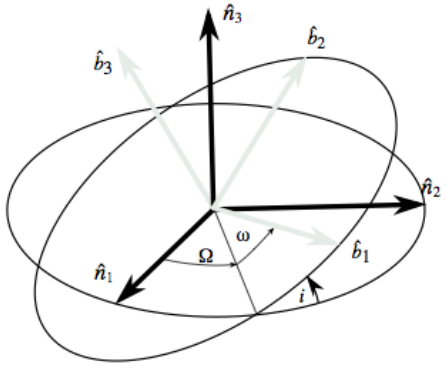


Figure 2.13: Euler angles representation. Source: [17]

[5]The gimbal lock singularity appears when the rotational axes of a three-axis system align, losing one degree of freedom and resulting in an ambiguous representation of the orientation.

This transformation using Euler angles follows a rotation sequence of 313. Firstly, the rotation Ω around the initial $\hat{b}_3$ axis which is coincident with $\hat{n}_3$ is done, then the rotation i around the rotated $\hat{b}_1$ axis occurs, and finally the rotation ω around the rotated $\hat{b}_3$ ends the transformation.

2.5.2 Quaternions

Quaternions are a mathematical extension of complex numbers that provide an efficient way to represent and manipulate rotations in three-dimensional space. They are not as intuitive as Euler angles but do avoid singularities. A quaternion is defined as a four-dimensional and it is typically represented as:

$$q = q_1 \cdot i + q_2 \cdot j + q_3 \cdot k + q_4 \quad \text{where} \quad [q_1, q_2, q_3, q_4] \in \mathbb{R} \quad and \quad i^2 = j^2 = k^2 = -1 \tag{2.1}$$

The vector consists of a scalar (real) part and a three-element vector (imaginary) part. Furthermore, it can be understood as a rotation around a unit vector by means of an axis-angle representation.

$$\begin{aligned} q_1 &= e_1 \cdot sin(\theta/2) \\ q_2 &= e_2 \cdot sin(\theta/2) \\ q_3 &= e_3 \cdot sin(\theta/2) \\ q_4 &= cos(\theta/2) \end{aligned} \tag{2.2}$$

Where the unit vector $\{e_1, e_2, e_3\}$ is the Euler axis and θ is the rotated angle. An axis-angle representation is shown below.

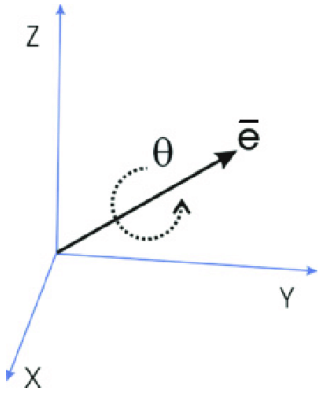


Figure 2.14: Axis-angle representation. Source: [18]

2.6 Controllers

Controllers are the core of every closed-loop control system, they are responsible for processing the feedback data and providing the required commands to make the system achieve a setpoint. The controllers to be discussed in this Section are the Proportional-Derivative and the Proportional-Integral-Derivative. Beginning

with the control function for the PD,

$$u(t) = K_p e(t) + K_d \frac{de(t)}{dt} \tag{2.3}$$

And for the PID,

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} \tag{2.4}$$

where K_p, K_i and K_d are the controller gains and $e(t)$ the error value computed as the difference between the setpoint and the process variable. In the table below, the most relevant features of both controllers are compared.

Feature	**PD**	**PID**
Steady State Error	It may appear due to lack of integral control	Low
Stability	High, may be lower than PD	High
Response Speed	High	High
Oscillations and Damping	Low	Low
Tuning Complexity	Medium complexity, only two gains	Higher complexity, three gains

Table 2.1: PD and PID controllers comparison.

To sum up, PID controllers perform better than PD in almost every feature analysed. However, its performance depends strongly on the system to be controlled and, in certain cases, a PD controller may be enough for an acceptable behaviour, thus reducing complexity. Due to that, PID controllers cannot be chosen blindly and alternatives such as the PD need to be evaluated.

Chapter 3

42 Simulator. Structure and Operation

The 42 is a powerful open-source simulator of spacecraft attitude and orbit dynamics developed by NASA. Its usage is highly oriented towards qualified engineers with knowledge of C++ language as well as spacecraft dynamics. Some of the features that distinguish this powerful simulator are firstly, the communication through sockets with other apps, which allows to externalise processes and paves the way for a HITL simulation. And secondly, the level of customisation of a mission to be developed, from multiple space dynamics models to complex, multiple spacecraft. In this chapter, all the folders and files that are indispensable to understand, as well as how the simulator works internally is described.

3.1 File system

The default version of the 42 simulator is composed by a total of 21 folders and 2 external files. They are mainly divided into three groups: the 42 root folders, the simulation folders and the miscellaneous folders.

Regarding the first group, its file hierarchy and connectivities can be depicted as follows:

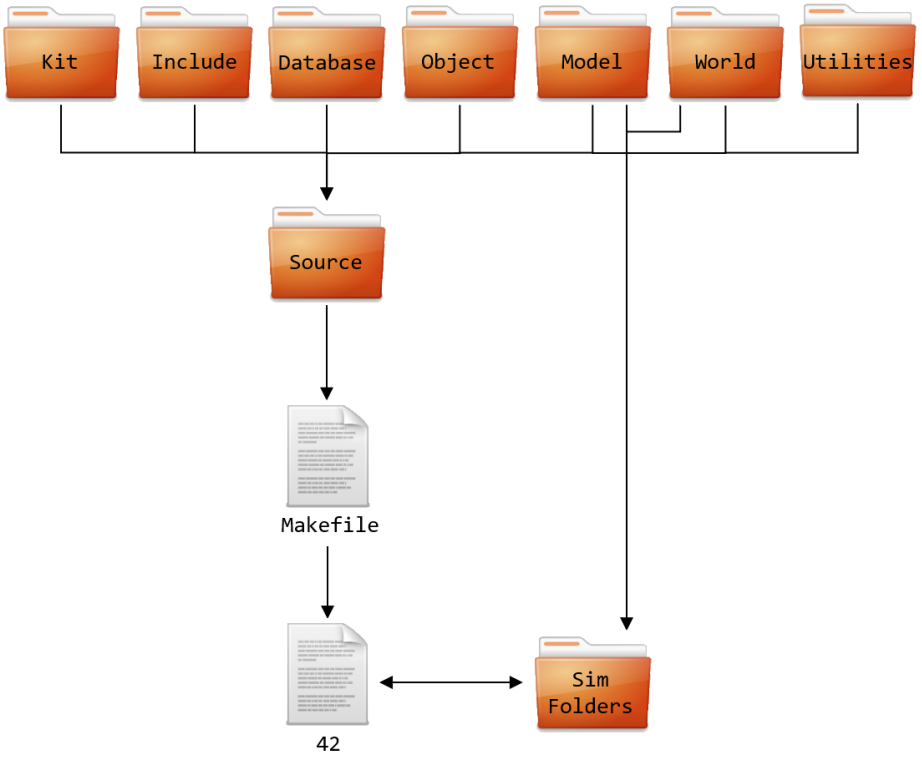


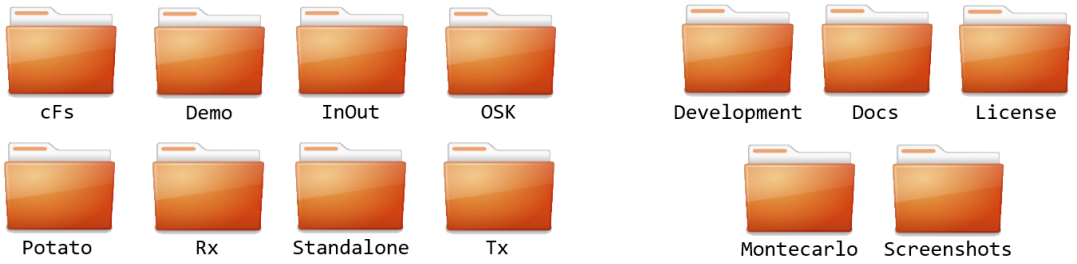
Figure 3.1: 42 root folders system.

From top-level to low-level:

- **42:** This file is the executable that allows to run the simulator when commanded from the terminal. To do so, any simulation folder needs to be called altogether when executing this file. It can be generated and erased using makefile specific commands.
- **Makefile:** It contains everything that is required to build the 42 executable. It calls every single file that is at a lower level than it. In addition, some 42 properties can be changed from inside the makefile such as, running the 42 in standalone or enabling the use of a GUI.
- **Source:** This folder contains the top-level code files of the simulator, the files that are executed at simulation rate are inside this folder. Section 3.1.1 takes a deeper look on this folder.
- **Kit:** It contains 16 low-level code files where all functions to be used at top-level code are stored. In this code there are no global variables and everything goes in and out through arguments and returns. Functions stored go from the scalar product of a vector or a two matrix product, to orbit frames or time frames conversions, among others. The best practice is to learn top-level code files first and afterwards, if some functions have not been understood or any complex function is needed, a `grep` command can be used inside Kit folder to search for it.
- **Include:** It contains every structure definition used in any 42 file, along with all the variables definition inside each structure. These files are useful to define new variables inside certain structures to store values during the simulation and avoid a reset in each iteration.

- **Database:** It contains some Python scripts to transform `json` type data into 42 type data.
- **Object:** It contains all the object files. These are created for each source files and called from the Makefile to build the 42 executable.
- **Model:** It contains the required files (geometry and textures) to build all the spacecraft simulation models. If a new spacecraft model needs to be added, its corresponding files have to be inside this folder.
- **World:** It contains all the planet's textures and gravity models to be used in the simulations.
- **Utilities:** It contains a variety of codes for different purposes. Those files are only useful in certain situations.

As for the two other groups, they are not as relevant as the root folders since the simulation folders are customized differently in each case and the miscellaneous folders do not take part in the simulation itself. However, each default folder is shown and described below.



(a) Simulation folders system. (b) Miscellaneous folders system.

Figure 3.2: Simulation and miscellaneous default folders.

Every simulation folder has a different purpose depending on the parameters defined.

- **cFs:** This simulation uses a customized flight software that enables communication with external software (such as the core Flight system) through sockets while running the 42 simulator in standalone mode.
- **Demo:** This simulation contains a variety of spacecrafts and orbits to serve as multiples examples.
- **InOut:** This simulation is the one called as default when no specific simulation folder is called.
- **OSK:** This simulation communicates with the Attitude Control App (AcApp) through a socket and has different spacecraft to choose.
- **Potato:** This simulation is useful to test new things without concern for the modifications done.
- **Rx:** This simulation reads incoming data through a socket interface.
- **Standalone:** This simulation enables to control the spacecraft attitude from an external app (`AcApp.c`) and communicate with the 42 running in standalone mode through sockets.
- **Tx:** This simulation writes outgoing data through a socket interface.

The miscellaneous folders serve mainly as additional information to some 42 functionalities.

- **Development:** It contains a variety of files of new 42 features that are yet under development or in testing phases.
- **Docs:** It contains all the official documentation of the simulator.
- **License:** It contains copyright and credits files as well as the open source software agreement.
- **Montecarlo:** It contains instructions and a matlab script to run a Montecarlo simulation with any simulation folder that is copied inside this folder.
- **Screenshots:** It contains multiple screenshots of different spacecraft visualised through the 42 GUI, along with instructions on how to record a simulation.

From all the folders defined, the most important ones that need to be understood and most likely modified when trying to implement new missions are: the Source, Include and Kit folders. Therefore, the next section takes a deeper look in the most complex folder out of the three mentioned, the Source folder.

3.1.1 Source files

The source files are the basis of the 42 simulator, therefore to implement new features and missions this are the files to be changed. Below, the most useful and important source files for design and development mission phases are described.

- **42main.c:** This file contains a minimal "main" function to facilitate compiling 42 as a library to be called from other applications. It calls the 42's top-level executive function `exec`.
- **42exec.c:** This file contains the 42's top-level executive function `exec`, which is responsible for calling directly or indirectly the rest of source files at simulation rate, except for the `42main.c`.
- **42init.c:** This file reads every input file and stores all the information in its associated variable. It also initializes time variables and loads planets properties. This file's main function is called only once at the beginning of the simulation from the `42exec.c`.
- **42cmd.c:** This file interprets and processes any simulation command that is called from the command input file. There is a total of 8 simulation commands that can be executed at the selected `SimTime`. The main function of this file is called at simulation rate.
- **42sensors.c:** This file processes all the information gathered by the sensors and stores it into multiple variables and structures. It is useful to know which information the user has available and where it is stored when using any sensor in the spacecraft simulated. The main function of this file is called at simulation rate.
- **42fsw.c:** This file's main function is executed at simulation rate and it contains, mainly, all the different types of flight software available for the spacecrafts. Error determination, feedback and failure detection and correction fall within the scope of this file.

The purpose of a flight software is to control and manage the operations of a spacecraft during its mission. If a new flight software wants to be implemented, changes need to be made in four places.

1. In the `SC_XX.txt`, the new flight software identifier must be introduced, `XX_FSW`.
2. In `Include>42defines.h`, the identifier must be added to the FSW Tags list with a unique integer (e.g. `#define XX_FSW 11`).
3. In `42fsw.c:FlightSoftware` function, a new case with the new function must be added, `XXFSW`. To do that, the function needs to be already coded.
4. In `42init.c:DecodeString` function a new line must be added to change the string `"XX_FSW"` into the constant `XX_FSW`.

- **42actuators.c:** This file processes the commanded signals to the actuators and limits them to avoid surpassing maximum values. For instance, for the thrusters the code checks its force, for the reaction wheels its torque and for the magnetorquers its momentum. The main function of this file is called at simulation rate.
- **42report.c:** This file is responsible for generating and writing all the output files of the simulation. Some lines must be added if the user wants to extract more data from the simulation (see Section 3.1.3). The main function of this file is called at simulation rate.
- **AcApp.c:** This file is responsible for generating the Attitude Control application when 42 is going to run in standalone mode.

3.1.2 Input files

To run the 42 properly, the simulation folder which is to be called must contain the input files with the desired information in them. In this Section, only the input files that are considered inside the scope of this thesis and are modified for the *Amazon Mission*, are detailed. However, the input files that are not explained in this Section, can be seen in Appendix B. Those files contain information about the simulation commands (`Inp_Cmd.txt`), Field of View (`Inp_FOV.txt`), Graphical User Interface (`Inp_Graphics.txt`), Inter-process communication (`Inp_IPC.txt`) and TDRS system (`Inp_TDRS.txt`).

Simulation Overview (Inp_Sim.txt)

This is the main input file, it contains top-level information of the simulation and calls other input files.

```
1 <<<<<<<<<<<<<<<<<  42: The Mostly Harmless Simulator  >>>>>>>>>>>>>>>>>
2 ************************* Simulation Control **************************
3 FAST                            !  Time Mode (FAST, REAL, or EXTERNAL)
4 5670.0      0.1                 !  Sim Duration, Step Size [sec]
5 0.1                             !  File Output Interval [sec]
6 FALSE                           !  Graphics Front End?
```

```
7  Inp_Cmd.txt                    !  Command Script File Name
```

Simulation Control

The time mode specifies the simulation speed, which can be set as `FAST` to run at maximum computing speed, `REAL` to run as in a real case and `EXTERNAL` to externalise the value to other applications such as the cFS.

The simulation duration, step size and file output interval are also adjusted here. In line 6, the GUI can be activated by changing the boolean `FALSE` by `TRUE` and in line 7 the command input file name is specified. For the development of this thesis, the command input file is not used because every command is given by the flight software.

```
8  ************************  Reference Orbits  ************************
9  1                              !  Number of Reference Orbits
10 TRUE   Orb_1.txt               !  Input file name for Orb 0
```

Reference Orbits

The number of reference orbits quantifies the orbits to be written in its following lines. These lines specify the availability of the orbit and the input file to be called.

```
11 ****************************  Spacecraft  ****************************
12 1                              !  Number of Spacecraft
13 TRUE  0 SC_cubesat.txt         !  Existence, RefOrb, Input file for SC  0
```

Spacecraft

The number of spacecraft quantifies the spacecraft to be written in its following lines. These lines specify the availability of the spacecraft, the reference orbit number to be followed and the input file to be called.

```
14 ***************************** Environment  *****************************
15 09 19 2022                     !  Date (UTC) (Month, Day, Year)
16 08 00 00.00                    !  Time (UTC) (Hr,Min,Sec)
17 18.0                           !  Leap Seconds (sec)
18 NOMINAL                        !  F10.7, Ap (USER, NOMINAL or TWOSIGMA)
19 230.0                          !  If USER_DEFINED, enter desired F10.7 value
20 100.0                          !  If USER_DEFINED, enter desired AP value
21 IGRF                           !  Magfield (NONE,DIPOLE,IGRF)
22 8   8                          !  IGRF Degree and Order (<=10)
23 2   0                          !  Earth Gravity Model N and M (<=18)
24 2   0                          !  Mars Gravity Model N and M (<=18)
25 2   0                          !  Luna Gravity Model N and M (<=18)
26 FALSE   FALSE                  !  Aerodynamic Forces & Torques (Shadows)
27 FALSE                          !  Gravity Gradient Torques
28 FALSE   FALSE                  !  Solar Pressure Forces & Torques (Shadows)
```

```
29 FALSE                          !  Gravity Perturbation Forces
30 FALSE                          !  Passive Joint Torques
31 FALSE                          !  Thruster Plume Forces & Torques
32 FALSE                          !  RWA Imbalance Forces and Torques
33 FALSE                          !  Contact Forces and Torques
34 FALSE                          !  CFD Slosh Forces and Torques
35 FALSE                          !  Output Environmental Torques to Files
```

Environment

In the first two lines, the exact start date of the simulation is set and, in the third line, the number of leap seconds[6] corresponding to the start date is specified. From lines 18 to 20 the Solar Radio Flux (`F10.7`) and the Geomagnetic Activity Planetary Index (`Ap`) values are set. From there on, the magnetic field and earth gravity models are specified (lines 21 to 25) and multiple perturbations can be activated (lines 26 to 35).

```
36 ************************ Ground Stations ***************************
37 2                                          ! Number of Ground Stations
38 TRUE  EARTH  -65.337  -4.997  "AMZ"         ! Exists, World, Lng, Lat, Label
39 TRUE  EARTH  2.02  41.56  "TGC"             ! Exists, World, Lng, Lat, Label
```

Ground Stations

The number of ground stations quantifies the locations to be written in its following lines. These lines specify the availability of the ground station, the world name, the coordinates where it is located and its label.

Orbit Model (Orb_1.txt)

This is input file contains all the information related to the orbit that runs in the simulation.

```
1  <<<<<<<<<<<<<<<<  42: Orbit Description File   >>>>>>>>>>>>>>>>
2  Test orbit                     !  Description
3  CENTRAL                        !  Orbit Type (ZERO, FLIGHT, CENTRAL, THREE_BODY)
4  ::::::::::::::  Use these lines if Body-Centered Orbit  ::::::::::::::::
5  EARTH                          !  Orbit Center
6  FALSE                          !  Secular Orbit Drift Due to J2
7  KEP                            !  Use Keplerian elements (KEP) or (RV) or FILE
8  AE                             !  Use Peri/Apoapsis (PA) or min alt/ecc (AE)
9  6871   6871                    !  Periapsis & Apoapsis Altitude, km
10 500.0  0.0                     !  Min Altitude (km), Eccentricity
11 45                             !  Inclination (deg)
12 55.0                           !  Right Ascension of Ascending Node (deg)
13 0                              !  Argument of Periapsis (deg)
```

[6] Leap second are used to keep UTC time synchronised to the Earth's rotation [19]. Seconds are either added or removed to compensate for irregularities and slowdown in the Earth's rotation rate.

```
14 320                          !  True Anomaly (deg)
15 0.0  0.0  0.0                !  RV Initial Position (km)
16 0.0  0.0  0.0                !  RV Initial Velocity (km/sec)
17 TRV  "ORB_ID"                !  TLE or TRV format, Label to find in file
18 "TRV.txt"                    !  File name
```

Orbit Description

The orbit type is selected from four possible options: `ZERO` corresponds to the selected world's surface (e.g. to simulate rovers), `FLIGHT` corresponds to the bodies between the world's surface and the Kármán line, `CENTRAL` is based on the two-body problem and `THREE_BODY` in the three-body problem.

For this thesis, a `CENTRAL` orbit is considered, therefore only the lines for this orbit type are explained. It begins with the selection of the world to orbit, then the J2 perturbation[7] can be activated. After that, the orbital elements are specified. If the Keplerian elements are chosen, the ellipse can be defined through its apoapsis and periapsis or through its minimum altittude and excentricity. In addition, the orbital elements are also defined: this includes the orbit inclination, RAAN, argument of periapsis and true anomaly (all explained in Section 2.4.1). If the `RV` option is chosen to define the orbit, lines 15 and 16 are used and, if the `FILE` option is chosen, the file format and name is obtained from lines 17 and 18.

CubeSat Model (SC_Cubesat.txt)

This is input file contains all the information related to the spacecraft model that runs in the simulation. It provides the body parameters and all the actuators and sensors specifications.

```
1 <<<<<<<<<<<<<<<<  42: Spacecraft Description File   >>>>>>>>>>>>>>>>
2 1-U Cubesat                   !  Description
3 "Cube 1"                      !  Label
4 GenScSpriteAlpha.ppm          !  Sprite File Name
5 CUBESAT_FSW                   !  Flight Software Identifier
6 0.1                           !  FSW Sample Time, sec
```

Spacecraft Description

The description purpose is to help the user identify better the spacecraft. The label is the tag to be displayed in the GUI and the sprite file contains the textures to be shown. After that, the flight software and its sample time is chosen. This is an important parameter because it determines the mission schedule to be followed by the spacecraft.

```
7 *************************** Initial Attitude ****************************
8 NAN                           ! Ang Vel wrt [NL], Att [QA] wrt [NLF]
9 0.0    0.0    0.0             ! Ang Vel (deg/sec)
10 0.0    0.0    0.0    0.0      ! Quaternion
```

[7]The J2 perturbation is a gravitational effect that influences the motion of orbiting satellites. It is caused by the oblateness of the celestial body that is orbited.

```
11 0.0    0.0    0.0    123      ! Angles (deg) & Euler Sequence
```

Initial Attitude

An initial attitude and angular velocity can be given to the spacecraft with respect to the frames chosen in line 8.

```
12 ********************************************************************
13 ************************ Body Parameters ******************************
14 ********************************************************************
15 1                              ! Number of Bodies
16 ================================ Body 0 ================================
17 1                              ! Mass
18 2.217E-3   2.217E-3   2.217E-3      ! Moments of Inertia (kg-mˆ2)
19 0.0  0.0  0.0                  ! Products of Inertia (xy,xz,yz)
20 0.0  0.0  0.0                  ! Location of mass center, m
21 0.0  0.0  0.0                  ! Constant Embedded Momentum (Nms)
22 Cubesat_1U.obj                 ! Geometry Input File Name
23 NONE                           ! Flex File Name
```

Body Parameters

The number of bodies and its parameters are specified in these lines. Mass and inertia values of each body are defined as well as its geometry.

```
24 ************************** Wheel Parameters ***************************
25 3                              ! Number of wheels
26 ============================  Wheel 0  ================================
27 0.0                            ! Initial Momentum, N-m-sec
28 1.0   0.0   0.0                ! Wheel Axis Components, [X, Y, Z]
29 7E-3   4.96E-4                 ! Max Torque (N-m), Momentum (N-m-sec)
30 1.58E-6                        ! Wheel Rotor Inertia, kg-mˆ2
31 0.00                           ! Static Imbalance, g-cm
32 0.00                           ! Dynamic Imbalance, g-cmˆ2
33 0                              ! Flex Node Index
```

Reaction Wheel Parameters

The number of reaction wheels is firstly defined. Then, an initial momentum can be set and its operating axis is specified. After that, the maximum torque and momentum, and the rotor inertia are required by the 42 model. The two imbalances are used for perturbation models.

For the sensors input data, they all follow the same pattern: First, the number of sensors carried is specified. Then, for each sensor, the sample time and operating axis are set. Finally, a series of specifications which depends on the sensor type are required. Below, the input data requested by each type of sensor carried by the *Amazon Mission* CubeSat is shown.

```
34 ****************************** Gyro ***********************************
35 3                              ! Number of Gyro Axes
36 ============================== Axis 0 ==================================
37 0.1                            ! Sample Time,sec
38 1.0  0.0  0.0                  ! Axis expressed in Body Frame
39 5000.0                         ! Max Rate, deg/sec
40 2000.0                         ! Scale Factor Error, ppm
41 1.0                            ! Quantization, arcsec
42 0.17                           ! Angle Random Walk (deg/rt-hr)
43 0.4  1.0                       ! Bias Stability (deg/hr) over timespan (hr)
44 0.0                            ! Angle Noise, arcsec RMS
45 0.0                            ! Initial Bias (deg/hr)
46 0                              ! Flex Node Index
```

Gyroscope

```
47 ********************** Coarse Sun Sensor ******************************
48 1                              ! Number of Coarse Sun Sensors
49 ============================== CSS 0 ===================================
50 0.1                            ! Sample Time,sec
51 0   1.0  1.0  0.0              ! Axis expressed in Body Frame
52 90.0                           ! Half-cone Angle, deg
53 1.0                            ! Scale Factor
54 0.0                            ! Quantization
55 0                              ! Flex Node Index
```

Coarse Sun Sensor

```
56 ************************* Star Tracker ********************************
57 1                              ! Number of Star Trackers
58 ============================== ST 0 ====================================
59 0.1                            ! Sample Time,sec
60 0.0  0.0  0.0  123             ! Mounting Angles (deg), Seq in Body
61 40.0   40.0                    ! X, Y FOV Size, deg
62 0.0  0.0  0.0                  ! Sun, Earth, Moon Exclusion Angles, deg
63 0.0  0.0  0.0                  ! Noise Equivalent Angle, arcsec RMS
64 0                              ! Flex Node Index
```

Star Tracker

```
65 ****************************** GPS ************************************
66 1                              ! Number of GPS Receivers
67 ============================= GPSR 0 ====================================
```

```
68 0.1                          ! Sample Time,sec
69 0.0                          ! Position Noise, m RMS
70 0.00                         ! Velocity Noise, m/sec RMS
71 0.00                         ! Time Noise, sec RMS
72 0                            ! Flex Node Index
```

GPS Receiver

The GPS Receiver is the only sensor that do not require an axis of operation because it can receive signals in any direction.

3.1.3 Output files

After the 42 ends its last iteration, it generates all the output files inside the simulation folder. This files continuously store data during the simulation through calling the `42report.c` source file at a rate specified in the `Inp_Sim.txt` input file. Below, all the output files that are generated by default are explained.

- **Acc.42:** It contains the spacecraft true and mean acceleration measured by the accelerometers.
- **Dyn00.42:** It contains a summary of the spacecraft dynamics data.
- **DynTime.42:** It contains the simulation time respect to the $J2000$ epoch in seconds.
- **Hvn.42:** It contains the spacecraft angular momentum in N frame.
- **Hwhl.42:** It contains the angular momentum of every reaction wheel.
- **KE.42:** It contains the kinetic energy of the spacecraft.
- **MTB.42:** It contains a summary of the magnetorquers.
- **PosN.42:** It contains xyz coordinates of the spacecraft position in N frame.
- **PosR.42:** It contains xyz coordinates of the spacecraft position in R frame.
- **PosW.42:** It contains xyz coordinates of the spacecraft position in W frame.
- **qbn.42:** It contains the spacecraft quaternion (B frame) in N frame.
- **RPY.42:** It contains the spacecraft Euler Angles in the $LHLV$ frame.
- **svb.42:** It contains the Sun Vector expressed in B frame.
- **svn.42:** It contains the Sun Vector expressed in N frame.
- **time.42:** It contains the net simulation time in seconds.
- **Tree00.42:** It contains a summary of the spacecraft body and joints connections.
- **u00.42:** It contains the spacecraft angular velocity in N frame and the linear velocity relative to its center of mass also in N frame.
- **VelN.42:** It contains the spacecraft velocity expressed as a vector in N frame.

- **VelR.42:** It contains the spacecraft velocity expressed as a vector in R frame.
- **VelW.42:** It contains the spacecraft velocity expressed as a vector in W frame.
- **wbn.42:** It contains the spacecraft angular velocity (B frame) in N frame.
- **x00.42:** It contains the spacecraft quaternion in N frame and the position relative to its center of mass also in N frame.

In case the user desires to obtain specific data of the simulation, some lines of code must be added to the `42report.c` source file. With that, a new file is generated with the commanded data stored during the simulation. The lines required are highlighted below where `newdata` is the new output file name.

```
void Report(void)
{
    static FILE **newdatafile; /*This line*/
    if (First) {
         First = FALSE;
         newdatafile = (FILE **) calloc(Nsc,sizeof(FILE *)); /*This line*/
         for(Isc=0;Isc<Nsc;Isc++) {
            if (SC[Isc].Exists) {
                sprintf(s,"newdata%02ld.42",Isc); /*This line*/
                newdatafile[Isc] = FileOpen(InOutPath,s,"w"); /*This line*/
            }
        }
    }
    if (OutFlag) {
        for(Isc=0;Isc<Nsc;Isc++) {
            if (SC[Isc].Exists) {
                fprintf(newdatafile[Isc],"%le\n", SC[Isc].newdata); /*This line*/
            }
        }
    }
}
```

3.2 Simulation procedure

Although every simulation can be different, they all follow the same steps. The purpose of this section is to explain how to set up and run a new simulation as well as define what the simulator does in every iteration.

3.2.1 Setup

In order to run a new simulation the steps below must be followed:

1. The input files must be checked to confirm all the simulation parameters are correct, such as `"Sim Duration"` or `"Graphics Front End? TRUE/FALSE"`.
2. The flight software must have no compilation errors and must fulfill its purpose. In addition, it is important to check that the spacecraft input file has the appropiate flight software identifier.

3. The 42 executable file must be created through the `"make"` command in the 42 folder terminal.

4. Now, the 42 Simulator can run any simulation folder. To do so, the `"./42 folder_name"` command have to be executed in the same terminal as in the previous step. The folder name corresponds to the simulation folder that the user wants to run.

With all that, the simulation begins just after Step 4 and once it finishes, it generates all the output files inside the simulation folder. In case the 42 executable files and the output files want to be deleted, the `"make clean"` command needs to be executed.

3.2.2 Simulation flowchart

As said before, once the command to run the simulation is executed, the 42 takes action and the software begins to iterate. In each simulation iteration, there are 6 main steps that are executed in succession, each of them being associated with a source file.

1. Initialize: This step is only done once per simulation. It reads all the user input files and stores the information into variables. Some of this information helps to initialize the ephemeris models.

2. Ephemeris Models: This step either initializes or updates all the information about positioning. It sets the new position of planets and spacecrafts and propagates orbits. Then, this information is used by both environment and sensor models.

3. Environment Models: This step computes the forces and torques that the environment exerts over the spacecraft. Then, information such as the magnetic or gravity field is used by the sensor models while the forces and torques are used in the dynamics models.

4. Sensor Models: This models take as input the simulated "truth" coming from the ephemeris and environment models as it would be in a real scenario. After that, depending on the properties set, the sensors output their measurements.

5. Flight Software Models: This models take all the sensors measurements and use them to process the control laws, which compute the actuator commands. Additionally, the flight software can be configured to do more actions such as update spacecraft variables or print data through the terminal.

6. Actuator Models: This models take the flight software commands as input, do the processing and output the resulting forces and torques actuating on the spacecraft.

7. Dynamics Models: This models are in charge of calculating how the spacecraft respond to the actuators output forces and torques. The dynamic equations of motion are integrated over a timestep and then, the time is advanced to the next timestep. Therefore, this is the last step of the simulation loop and the flow returns to Step 2: "Ephemeris Models" with updated dynamic states.

Finally, when the `SimTime` reaches its limit value, there is no return from the dynamics models to the ephemeris models and the simulation comes to an end. Below, the whole simulation loop is depicted with its associated source files names.

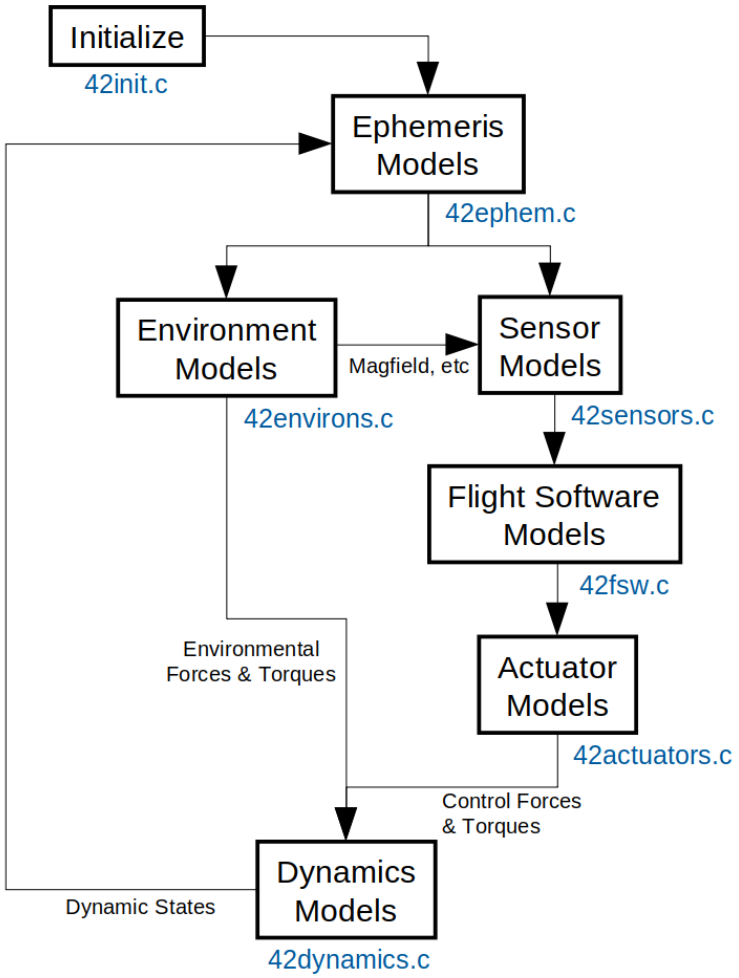


Figure 3.3: Simulation flowchart. Source: [20]

.

Even though existing 18 source files, as it can be seen in Figure 4.5, only 7 are at the top level of execution, except for `42main.c` and `42exec.c` that are on top of them all. These 7 files are so important due to being the ones to do the calculations and obtain the results.

Chapter 4

Amazon Mission. Concept

The *Amazon Mission*, created and developed by the DISEN Group, intends to receive and transmit data from the amazon wildlife to the Terrassa Ground Control. The purpose of the mission is accomplished by a 1U CubeSat orbiting the Earth and flying by each ground station as many times as it is allowed by its orbit. Therefore, this chapter sums up all the information about the mission and goes into detail in the aspects inside the scope of this thesis.

4.1 Flight schedule

To provide an overview of the *Amazon Mission* and introduce its architecture inside the 42 Simulator, a simulation loop is detailed. It begins determining the attitude and position of the CubeSat through the sensor processing. After that, the software can locate the CubeSat in space and check whether or not it is inside the control zone of a ground station. If it is inside, the required setpoint to point at the ground station is calculated and, if the pointing accuracy is less than the threshold imposed, the total visible time of the CubeSat during the flyby is increased by the simulation time step. However, if the CubeSat is outside any control zone, the setpoint is calculated and sent to the control system.

After that, the PD control processes the setpoint and outputs a torque command which is sent to the reaction wheels. To end the simulation loop, some more data is computed. The battery charge is updated with the energy consumption and absorption of the CubeSat components. In case the CubeSat was executing a flyby and the software detects that it has left the control zone, it computes the total amount of data exchanged and the energy consumed if the antenna was transmitting. Otherwise, after updating the battery charge, the pointing accuracy is computed and stored for the next iteration, which begins with the sensor processing. Finally, if the last iteration is completed, the simulation ends and the output files are generated.

To sum up, the whole simulation loop is depicted below.

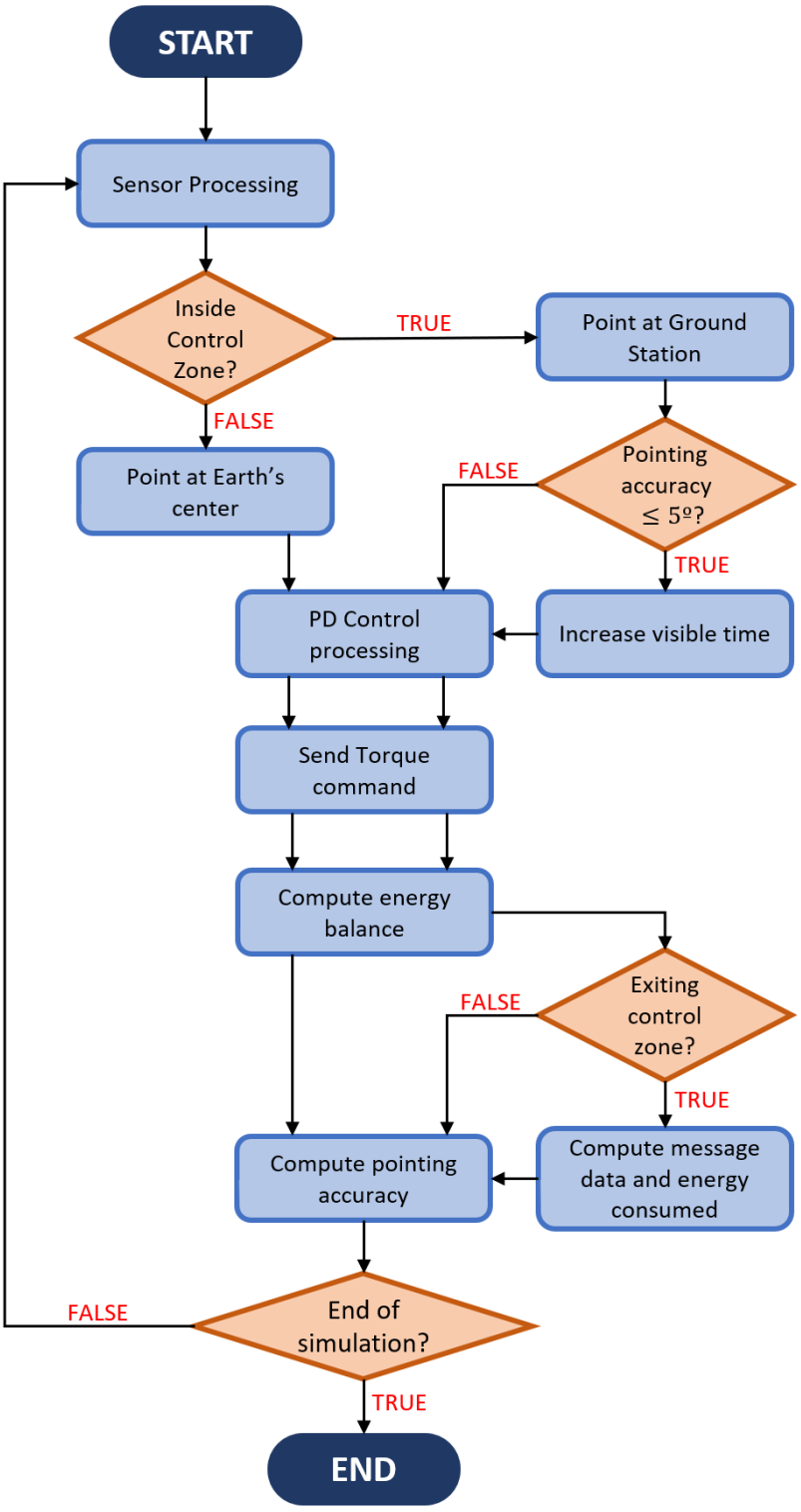


Figure 4.1: Amazon Mission flowchart inside 42 Simulator.

Once it has been defined what the CubeSat does at any moment, it needs to be defined how it accomplishes every action. This can be divided into two parts: one that involves everything around the CubeSat and it is mission relevant, and one that involves everything about the CubeSat itself.

4.2 Mission specifications

The mission details which are independent of the CubeSat are the orbit that follows, the ground stations that flyby and the 42 input parameters that are set.

4.2.1 Orbit

For the orbital elements, the data provided from the DISEN Group is used. This values come from a thesis developed by Mar Bonet Manzanares [21]. To initialise the orbit, the initial date has been set the 19^{th} September, 2022 at 08 : 00 : 00 am.

Orbit element	Value
Semimajor axis (above Earth's surface)	500 km
Eccentricity	0 deg
Inclination	45 deg
Right Ascension of Ascending Node	55 deg
Argument of Periapsis	0 deg
True Anomaly	320 deg

Table 4.1: Amazon Mission orbital elements.

In addition, the orbital period and periodicity are computed. For the orbital period of the CubeSat (T_{CS}):

$$T_{CS} = 2\pi\sqrt{\frac{(R_E + h)^3}{G \cdot M_E}} \tag{4.1}$$

where the Earth radius $R_E = 6371$ km, the orbit height $h = 500$ km, the gravitational constant $G = 6.67 \times 10^{-11}\ \frac{m^3}{kg \cdot s^2}$ and the Earth mass $M_E = 5.972 \times 10^{24}$ kg. Then,

$$T_{CS} = 5670\ s \equiv 1h\ 34min\ 30seg \tag{4.2}$$

To analyse the orbit periodicity, the Ansys software *Systems Tool Kit* (see Section 2.3.1) is used. To estimate a number of days, the error in distance (respect to the first value obtained) when the CubeSat flyby TGC is computed (see more detail in Appendix A). With that, three intervals are considered and summarised in the table below:

5 days interval		**6 days interval**	
Date (UTC)	**Error [km]**	**Date (UTC)**	**Error [km]**
19/09/22 at 8:30h	0	19/09/22 at 8:30h	0
24/09/22 at 8:21h	45	25/09/22 at 7:59h	45
29/09/22 at 8:12h	91	01/10/22 at 7:29h	77
04/10/22 at 8:03h	129	07/10/22 at 6:58h	124

Table 4.2: Amazon Mission orbit periodicity analysis (5 and 6 days interval).

11 days interval

Date (UTC)	Error [km]
19/09/22 at 8:30h	0
30/09/22 at 7:51h	16
11/10/22 at 7:11h	40
22/10/22 at 6:32h	52

Table 4.3: Amazon Mission orbit periodicity analysis (11 days interval).

The results show that setting an orbit periodicity of 11 days reduces the error in distance by half compared to a 5 and 6 days interval. Therefore, even though the error value grows as time passes, a value of 11 days is accepted taking into account that it may be refined if much longer time intervals want to be considered.

4.2.2 Ground Stations

The ground stations are responsible for mainly transmitting and receiving data from the CubeSat, and in case any software update wants to be done ground station are also the way. Regarding the two ground stations involved in this mission, the table below specifies its coordinates and its main purpose.

Ground Station	Longitud [deg]	Latitude [deg]	Purpose
Amazon (AMZ)	-65.337	-4.997	Transmitting data
Terrassa (TGC)	2.023	41.563	Receiving data

Table 4.4: Amazon Mission ground stations.

To clearly understand the coordinates, they are also represented in a map.

(a) Amazon Ground Station. (b) Terrassa Ground Station.

Figure 4.2: Amazon Mission Ground Stations in map.

For the Amazon Ground Station an estimated location has been taken while for the Terrassa Ground Station a precise location for the *ESEIAAT - TR5* building has been set.

4.2.3 Mission Inputs

To translate the previous information into 42 language, the input files of the simulation must be modified. Below, each file change related to the mission and not the CubeSat is explained. Complete files can be seen in Appendix B.

- **Inp_Cmd.txt:** No changes on command input file has been done.
- **Inp_FOV.txt:** No changes on Field of View input file has been done.
- **Inp_Graphics.txt:** No changes on Graphics input file has been done.
- **Inp_Region.txt:** No changes on Region input file has been done.
- **Inp_Sim.txt:** `Sim Duration` value has been continuously changing since it depends on the case to be studied. `Orbit file ID` has been changed to match the file name. `Date, Time` values has been set at the defined in Section 4.2.1. `Ground Stations` has been updated to the two previously defined in Table 4.4.
- **Inp_TDRS.txt:** No changes on TDRS input file has been done.
- **Orb_1.txt:** `Orbit Type` has been set as Earth-Centered and orbital elements has been updated with the ones specified in Table 4.1.

4.3 CubeSat specifications

Regarding the CubeSat properties they can be easily divided into two parts, the hardware components that theoretically carries and the software that translates that information to the 42 simulator.

4.3.1 Hardware

Beginning with the CubeSat as a whole, the maximum weight and dimensions for a 1U CubeSat of $m = 1.33$ kg and $a = 0.1$ m edge length are chosen. Then, considering a perfect and axisymmetric mass distribution, the matrix of inertia is obtained.

$$I = \frac{1}{6} m\, a^2 = 2.217 \times 10^{-3}\ kg\ m^2 \tag{4.3}$$

Therefore,

$$I = \begin{pmatrix} 2.217 & 0 & 0 \\ 0 & 2.217 & 0 \\ 0 & 0 & 2.217 \end{pmatrix} \times 10^{-3}\ kg\ m^2 \tag{4.4}$$

For the hardware components, the CubeSat has 3 reaction wheels to modify its attitude, a 3-axis gyroscope to measure angular velocities, a star tracker to measure its orientation, a GPS receiver to know its position,

an antenna to transmit the information received to the TGC, a battery to power the CubeSat and solar arrays to recharge the battery with solar energy. It has also been added a coarse sun sensor to be able to estimate the power absorbed by the solar arrays. It is important to point out that the components mentioned do not represent the entirety of hardware that the CubeSat really carries, but they are the ones that are required to know its specifications by the 42 and the flight software.

Reaction Wheels

Reaction wheels are used for three-axis attitude control in spacecrafts. They can provide high levels of accuracy, being a deciding factor when choosing attitude control actuators for high pointing precision spacecrafts. Its operation is based on the principle of conservation of angular momentum, meaning that they can change the spacecraft's orientation by transferring angular momentum between the wheels and the satellite. For the *Amazon Mission* the reaction wheel model provided by the DISEN is used, which, in turn, was developed by Ariana Miño in his Master Thesis [22].

Maximum Torque	7 mN·m
Wheel Rotor Inertia	$1.58 \times 10^{-6}\ kg \cdot m^2$
Maximum Angular Velocity	314.16 rad/s
Maximum Momentum	0.496 mN·m·s
Motor efficiency	50%

Table 4.5: Reaction wheels specifications.

In addition, for power consumption purposes, an efficiency of 50% is considered, meaning that the reaction wheels consume twice the ideal value.

Gyroscope

Gyroscopes operation is based on the principle of rigidity. They consist of a spinning rotor that maintains a stable orientation in space and allows the spacecraft to measure angular velocity variations and detect deviations from its expected orientation. For the *Amazon Mission* the gyroscope model used is a *STIM202 3-axis Gyro Module* [23].

Figure 4.3: STIM202 3-axis Gyro Module. Source: [23]

Mass	55 g
Dimensions	$45 \times 20 \times 39$ mm
Idle Power Consumption	1.5 W
Output rate	62 Hz
Maximum Rate	480 deg/s
Wheel Rotor Inertia	$1.58 \times 10^{-6}\ kg \cdot m^2$
Scale Factor Error	2000 ppm
Angular Random Walk	0.17 deg/$\sqrt{h}$
Bias Stability	0.4 deg/h

Table 4.6: Gyroscope specifications.

Star Tracker

Star trackers are optical instruments that can determine the spacecrafts orientation through stars observation using photocells or a camera. It provides the spacecraft with a high accuracy attitude measure. For the *Amazon Mission* the star tracker model used is from *KairoSpace* [24].

Figure 4.4: KairoSpace Star Tracker. Source: [24]

Mass	197 g
Dimensions	$56 \times 60 \times 93$ mm
Idle Power Consumption	0.7 W
Output rate	10 Hz
Field of View	40 deg
Sun exclusion angle	30 deg
Moon exclusion angle	20 deg

Table 4.7: Star tracker specifications.

GPS Receiver

The GPS receiver allows to position the CubeSat in an *Earth Centered Inertial* frame through x, y, z coordinates. For the *Amazon Mission*, the GPS receiver model used is a *WarpSpace* GPS Module [25]. This model is perfect for CubeSat due to being available at high altitude and velocity in space.

Figure 4.5: WarpSpace GPS Module. Source: [25]

Mass	3 g
Dimensions	$24 \times 20 \times 5.3$ mm
Idle Power Consumption	0.15 W
Output rate	$1 - 20$ Hz
Maximum Altitude	8000 km
Maximum Velocity	11.2 km/s

Table 4.8: GPS receiver specifications.

Antenna

The antenna is responsible for transmitting data to the ground station. For the *Amazon Mission* a parabolic antenna with a power of P = 1 W is used [26]. Also, a communication bidirectional bit rate of 512 kbps is considered [26]. No more information about the communications is needed since is only used to estimate power consumption and amount of data being received and transmitted.

Battery

As a power storage system, a battery pack consisting on 3x L_iF_ePO4 batteries in series is used. The whole battery pack has a voltage of $V = 9.6$V and a capacity of $C = 1500$ mAh.

Solar Arrays

For the solar arrays, it is considered that they are present in four out of the six faces of the CubeSat, with an efficiency of $\eta = 90\%$. It is also considered that the solar arrays cover a 90% of the total face area.

Coarse Sun Sensor

Sun sensors deliver information about the position of the sun relative to the spacecraft. This information is not required for the CubeSat operation but is useful to estimate the power absorbed by the solar arrays. Therefore, a fictitious 3-axis sun sensor is set at the simulator to be able to compute this value.

4.3.2 Software

To implement all the hardware on the 42 simulator, the input files must be modified. Below, each file change related to the CubeSat is explained. Complete files can be seen in Appendix B.

- **Inp_IPC.txt:** No changes on inter-process communication input file has been done. This thesis comprises early phases of this mission development in the 42 simulator, hence all the work is done inside the simulator and it is future work to separate the processes.
- **Inp_Sim.txt:** `Spacecraft file ID` has been changed to match the file name.
- **SC_cubesat.txt:** `Flight software ID` has been updated to `CUBESAT_FSW`. Inertia and geometry input file have also been updated to the required. Finally, reaction wheels, gyroscopes, sun sensor, star tracker and GPS parameters have been updated to the previously defined.

4.4 Pointing procedures

The *Amazon Mission* CubeSat purpose is based on its capability for 1-axis pointing to specific points with high levels of accuracy. Hence, the computation to obtain the angles that deliver the desired attitude at every moment needs to be done. As seen in Section 4.1, the CubeSat needs to be capable of pointing at the Earth's center at any time, as well as to point to ground stations when it flyby them. Below, both pointing procedures are described, beginning with the Earth pointing due to not being as complex as the ground station pointing.

4.4.1 Earth pointing

Through the GPS, the CubeSat is provided at every simulation step with its position with respect to the *J2000 ECI* Reference frame (N) $A_N = \{x_N^A, y_N^A, z_N^A\}$, (see more detail about this frame in Section 2.4.1). To be able to point at the Earth's center, the CubeSat needs a setpoint to reach. According to the way the CubeSat control system is configured, the setpoint has to be the angle rotated around each axis respect to the frame N to point in the desired direction. Since the purpose is to point with one axis, only two rotations around the two remaining axis are needed.

The first thing needed to point at the Earth's center is a vector that unites both Earth's center and CubeSat. This is easily obtained from the GPS coordinates since the origin of the N frame is where the CubeSat aims. With that, making the vector unitary and considering a simulation specific case to exemplify, it is obtained that.

$$T_N = A_N = \{0.388, 0.913, 0.122\} \tag{4.5}$$

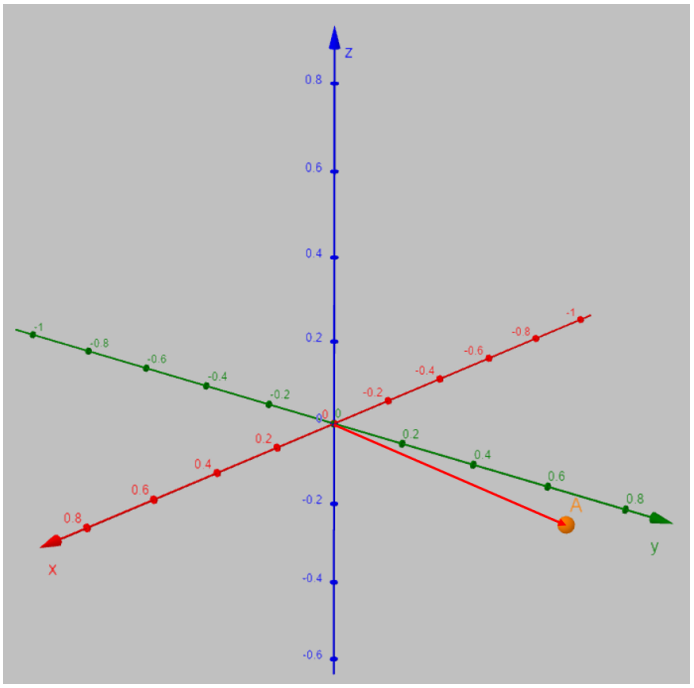


Figure 4.6: Isometric view of the target vector (red) before axis rotations.

where A_N represents the unitary coordinates vector of the CubeSat at a certain moment. The desired pointing axis is the Z, therefore two rotations around X and Y are required. Beginning with the rotation around X, it can be represented as:

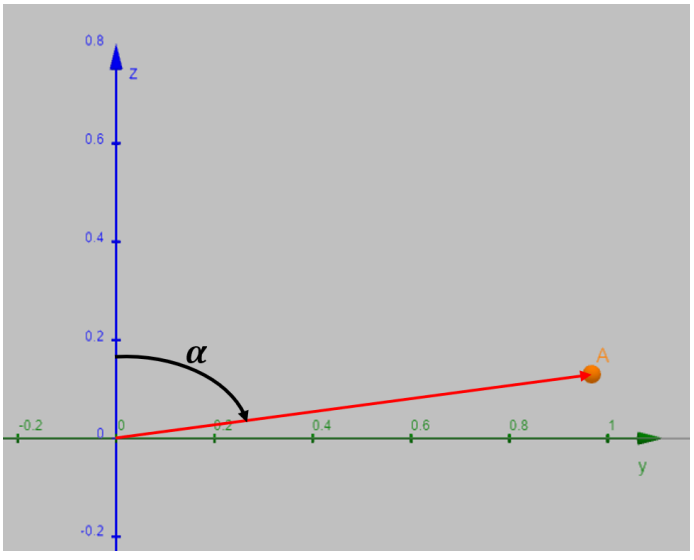


Figure 4.7: A point projected in Y-Z plane.

Where the red vector is the target vector for the Z axis and α the required negative rotation around X axis, which is computed as:

$$\alpha = -arctan\left(\frac{y_N^A}{z_N^A}\right) = -arctan\left(\frac{0.913}{0.122}\right) = -82.39\ deg \tag{4.6}$$

Thus, the new rotated axis Y' and Z' look like,

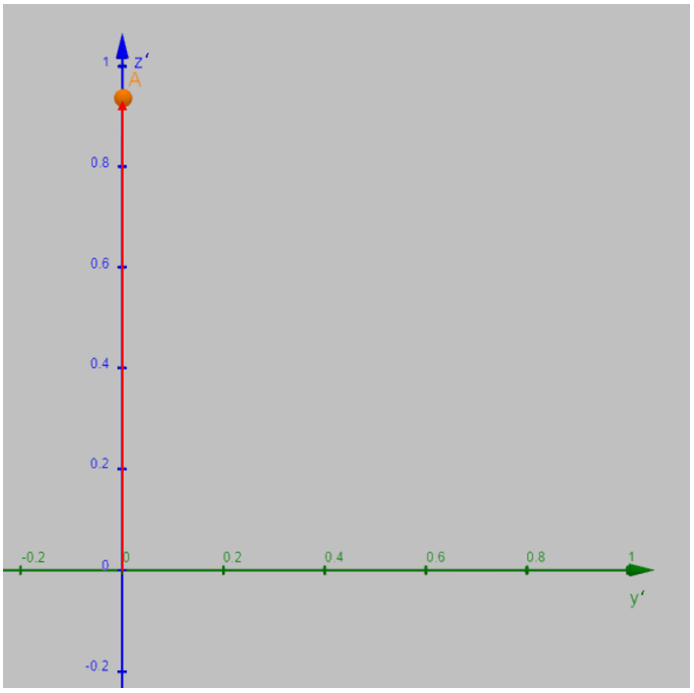


Figure 4.8: A point projected in rotated Y'-Z' plane.

After that, there is only a rotation left around Y' axis. It is importance to point out that, even though Y, Z axis have rotated the vector is still unitary and therefore, its length is still the unit. Thanks to that, no matter the value on Z axis the second rotation angle can be obtained.

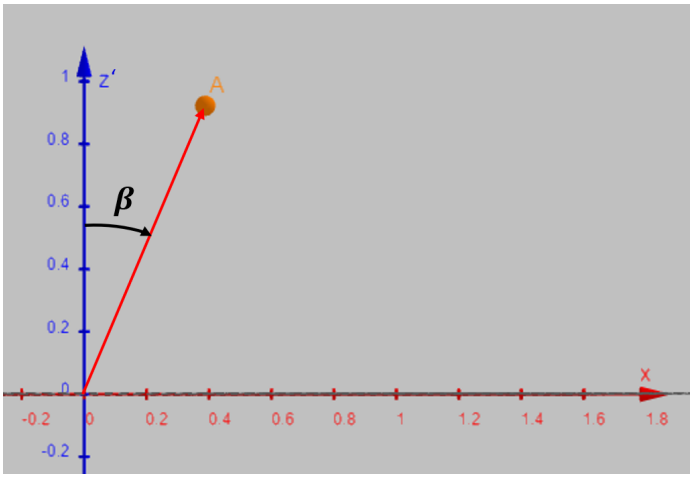


Figure 4.9: A point projected in X-Z' plane.

Where the red vector is the target vector for the Z axis and β the required positive rotation around Y' axis, which is computed as:

$$\beta = arcsin\left(\frac{x_N^A}{l}\right) = arcsin\left(\frac{0.388}{1}\right) = 22.83\ deg \tag{4.7}$$

Finally, after a rotation around X axis and then, one around Y' axis, the initial Z axis points to the desired direction.

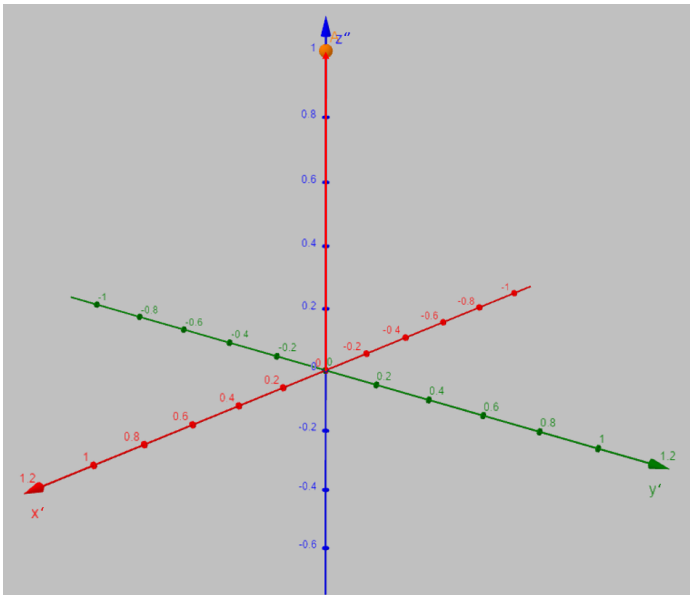


Figure 4.10: Isometric view of the target vector (red) after axis rotations.

To sum up, the required rotations of each axis respect to the frame N to point at Earth's center are compared with the results printed by the 42 simulator.

	Angle rotation around axis		
	X	**Y**	**Z**
Numerical results	−82.39 deg	22.83 deg	0
42 results	−82.41 deg	22.86 deg	0

Table 4.9: Rotation values comparison.

Regarding the code that does this job, it follows the same process as the one defined.

```
1 PV->R[3]={0,0,1}
2 for(i=0;i<3;i++) Cmd->targ_vec[i] = AC->GPS[0].PosN[i];
3 UNITV(Cmd->targ_vec);
4 PointGimbalToTarget(123,CGiBi,CBoGo,Cmd->targ_vec,PV->R,Cmd->Ang);
5 Cmd->RotSeq = 123;
6 A2C(Cmd->RotSeq,Cmd->Ang[0],Cmd->Ang[1],Cmd->Ang[2],C);
7 C2Q(C,Cmd->qrn);
8 for(i=0;i<3;i++) Cmd->wrn[i] = 0.0;
```

First, the code defines the axis chosen in the body frame to point at Earth's center. After that, the targeting vector is assigned to the `Cmd→targ_vec` variable and it is made unitary.

Then in line 4, a 42 default function called `PointGimbalToTarget` does all the job. This function's purpose is *"to find the euler angles to point a given boresight vector fixed in a joint's outer body parallel to a target vector fixed in the joint's inner body"*. However, if `CGiBi` and `CBoGo` are identity matrix, this function outputs the axis rotations to point `PV→R` vector to `Cmd→targ_vec` vector, limiting its value between $[-\pi, +\pi]$.

From line 5 to 8, the output angles are transformed into quaternions and the CubeSat's angular velocity setpoint is commanded null (see Appendix C for full code).

In addition, a visual check can be done visualizing the simulation in the 42 GUI.

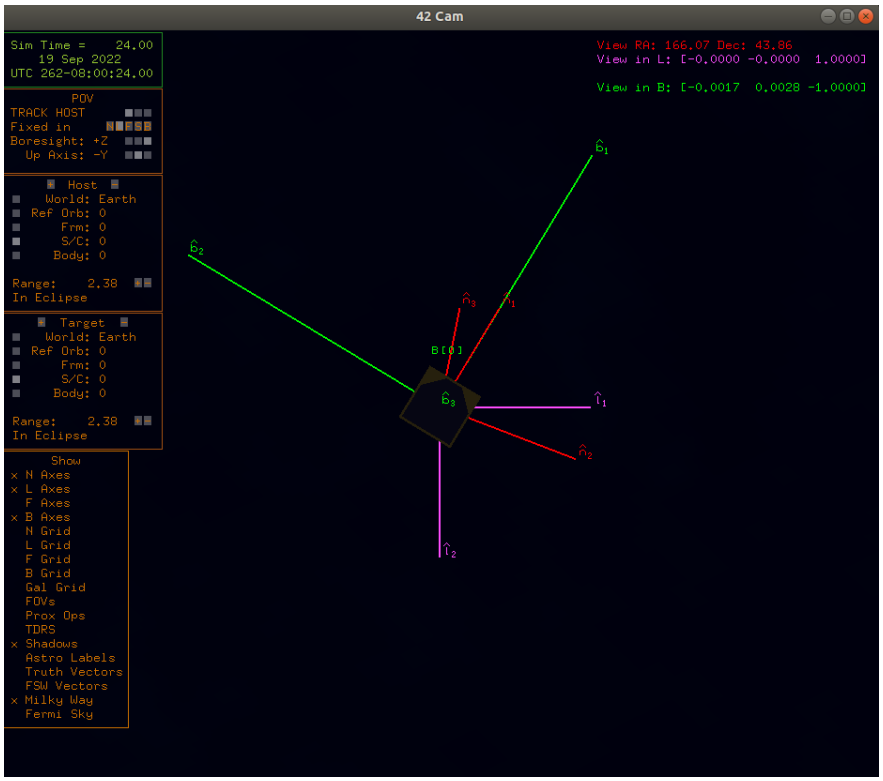


Figure 4.11: 42 GUI showing the CubeSat attitude.

In top-right corner it can be seen how L and B frame almost match, meaning that both Z axis are lined up. This comparison also verifies the pointing accuracy because, as seen in Section 2.4.3, L frame Z axis always points to nadir.

4.4.2 Ground station pointing

Regarding the pointing procedure to aim at ground stations, it is quite similar to the previously explained. Instead of setting the vector's origin at the Earth's center, it must be set at the ground station position. Then, once the vector that goes from the ground station to the CubeSat is obtained, the same operations are done as to obtain the axis rotations for Earth pointing.

There is only one difficulty to obtain the pointing vector and it is because the 42 simulator only provides ground station positioning in the *Earth Centered Earth Fixed* frame (W) $B_W = \{x_W^B, y_W^B, z_W^B\}$. Therefore, to do the corresponding transformation from the N frame to the W frame, the *Prime Meridian Angle* provided by the 42 is used.

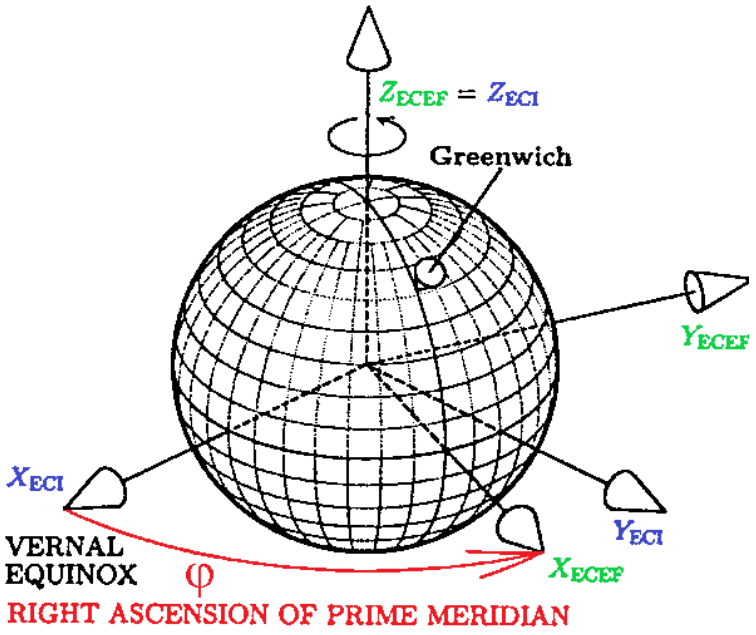


Figure 4.12: Prime Meridian Angle (φ) representation. Source: [27].

This variable angle represents the angle difference between both reference frame at a specific time. Since the Z axis match, only a rotation on $\{x_W^B, y_W^B\}$ coordinates is needed (a simulation specific case is considered to exemplify the process).

The CubeSat position in N frame is defined as,

$$A_N = \{0.486, 0.868, 0.101\} \tag{4.8}$$

While the ground station position in W frame is defined as,

$$B_W = \{0.416, -0.905, -0.087\} \tag{4.9}$$

The Prime Meridian Angle at this specific time is,

$$\varphi = 121.59\ deg \tag{4.10}$$

Looking at the XY plane, the ground station position (B) and both axis are represented.

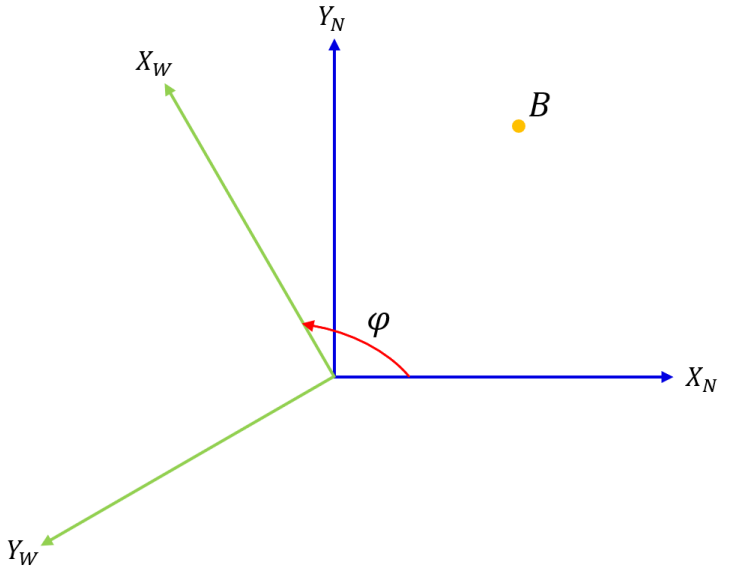


Figure 4.13: Ground station position in both ECI (N) and ECEF (W) frames.

Therefore, to transform coordinates from W to N, the equations below must be used.

$$\begin{aligned} x_N^B =& x_W^B \cdot cos(\varphi) - y_W^B \cdot sin(\varphi) = 0.553 \\ y_N^B =& x_W^B \cdot sin(\varphi) + y_W^B \cdot cos(\varphi) = 0.828 \\ z_N^B =& z_W^B = -0.087 \end{aligned} \tag{4.11}$$

Once the ground station coordinates are expressed in N frame, the new target vector is computed. To do this, vectors cannot be unitary and original coordinates values must be used during all the process previous process. In this example, vectors are unitary to ease the understanding.

$$T_N = A_N - B_N = \{-0.131, 0.479, 0.868\} \tag{4.12}$$

With the unitary target vector obtained, the same procedure to compute the axis rotations as in Earth pointing is followed.

$$\begin{aligned} \alpha =& -arctan\left(\frac{y_N^T}{z_N^T}\right) = -arctan\left(\frac{0.479}{0.868}\right) = -28.89\ deg \\ \beta =& arcsin\left(\frac{x_N^T}{l}\right) = arcsin\left(\frac{-0.131}{1}\right) = -7.53\ deg \end{aligned} \tag{4.13}$$

Comparing the results with the ones outputted by the 42 in this specific case.

	Angle rotation around axis		
	X	**Y**	**Z**
Numerical results	−28.89 deg	−7.53 deg	0
42 results	−28.87 deg	−7.53 deg	0

Table 4.10: Rotation values comparison.

The code responsible for doing the previous computations is seen below.

```
1  W = &World[EARTH];
2  CosPriMerAng = cos(W->PriMerAng);
3  SinPriMerAng = sin(W->PriMerAng);
4  if (AC->QFLAG[0] == 1) {
5      for(i=0;i<3;i++) PV->W[i] = GroundStation[0].PosW[i];
6  }
7  else if (AC->QFLAG[1] == 1) {
8      for(i=0;i<3;i++) PV->W[i] = GroundStation[1].PosW[i];
9  }
10 pn[0] =  PV->W[0]*CosPriMerAng - PV->W[1]*SinPriMerAng;
11 pn[1] =  PV->W[0]*SinPriMerAng + PV->W[1]*CosPriMerAng;
12 pn[2] =  PV->W[2];
13 for(i=0;i<3;i++) Cmd->targ_vec[i] = AC->GPS[0].PosN[i] - pn[i];
14 UNITV(Cmd->targ_vec);
15 PointGimbalToTarget(123,CGiBi,CBoGo,Cmd->targ_vec,PV->R,Cmd->Ang);
16 Cmd->RotSeq = 123;
17 A2C(Cmd->RotSeq,Cmd->Ang[0],Cmd->Ang[1],Cmd->Ang[2],C);
18 C2Q(C,Cmd->qrn);
19 for(i=0;i<3;i++) Cmd->wrn[i] = 0.0;
```

The code begins assigning the Earth's Prime Meridian Angle to its variables. Then, from line 4 to 13 the ground station coordinates in W frame are obtained, transformed to the N frame and the new target vector `Cmd→targ_vec` is computed. After that, from line 14 to 19, the code follows the same procedure as in Earth pointing to obtain the required commands (see Appendix C for full code).

4.4.3 Pointing angle error

To be able to check at every simulation step if the CubeSat is actually pointing where it is commanded, the error in angle must be calculated. It is defined as the angle between the target vector (`Cmd→targ_vec`) and the actual Z body vector. Since the target vector is defined in the pointing procedures, only the read Z body vector needs to be computed.

As available data, the quaternion of B frame in N frame is provided by the sensors, therefore, the rotated angles to transform a vector in N frame to B frame can be obtained. Once the angles are calculated, a vector coincident with the Z axis in N frame is rotated and the coordinates of the actual Z body vector are obtained. The process is detailed below.

To obtain the Euler angles from the quaternion, two 42 functions are used, one to transform the quaternion into a DCM and the other to transform from a DCM to an Euler angles sequence of y, x, z. After that, rotations around Y axis and then, around X axis must be done. The sequence must be followed to be

consistent with the transformation done with the DCM. The rotation matrix are defined below.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & -sin(\alpha) \\ 0 & sin(\alpha) & cos(\alpha) \end{bmatrix} \qquad R_y = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ 0 & 1 & 0 \\ -sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{4.14}$$

Doing the matrix product to add the rotations, $R_{xy} = R_x \times R_y$

$$R_{xy} = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ sin(\alpha)sin(\beta) & cos(\alpha) & -sin(\alpha)cos(\beta) \\ -cos(\alpha)sin(\beta) & sin(\alpha) & cos(\alpha)cos(\beta) \end{bmatrix} \tag{4.15}$$

Multiplying by the vector to be rotated, $read\_vec = R_{xy} \times \{0; 0; 1\}$

$$read\_vec = \begin{bmatrix} cos(\beta) & 0 & sin(\beta) \\ sin(\alpha)sin(\beta) & cos(\alpha) & -sin(\alpha)cos(\beta) \\ -cos(\alpha)sin(\beta) & sin(\alpha) & cos(\alpha)cos(\beta) \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} sin(\beta) \\ -sin(\alpha)cos(\beta) \\ cos(\alpha)cos(\beta) \end{bmatrix} \tag{4.16}$$

Therefore, the angle (θ) between the target vector and the actual Z body vector is computed.

$$\theta = arccos\left(\frac{targ\_vec \cdot read\_vec}{||targ\_vec|| \cdot ||read\_vec||}\right) \tag{4.17}$$

The code computing the angle error at simulation rate is found below.

```
1 /* Compute pointing angle error */
2 Q2C(AC->qbn,CBN);
3 C2A(213,CBN,read_vec[0],read_vec[1],read_vec[2]);
4 read_vec[0] = sin(read[1]);
5 read_vec[1] = -sin(read[0])*cos(read[1]);
6 read_vec[2] = cos(read[0])*cos(read[1]);
7 AC->point_err = acos(VoV(Cmd->targ_vec,read_vec)/(MAGV(Cmd->targ_vec)*MAGV(read_vec)));
```

Lines 2 and 3 transform the quaternion into Euler angles. After that, lines 4 to 6 compute the actual Z body vector following Eq. (4.16) and finally, line 7 obtains the pointing error following Eq. (4.17) (see Appendix C for full code).

Chapter 5

Amazon Mission. Development

In this chapter, it is detailed how the main purposes of this thesis are achieved for the *Amazon Mission*. It includes the ADCS design, the definition of the active control zones and the preliminary design of the battery management system and the data exchanged with both ground stations. With that, all the initial goals are developed and only an analysis of the results is left.

5.1 Attitude Determination and Control System

This system is responsible for controlling the orientation and stability of the CubeSat at any moment. Through combining sensors, actuators and control algorithms, the desired attitude is achieved and therefore, the *Amazon Mission* objectives can be fulfilled. This section details the hardware that provides the attitude data as well as the control system that processes the data and sends the appropriate commands.

5.1.1 Attitude determination

In order to control the system correctly, accurate data about the attitude of the CubeSat must be provided. This includes information on its orientation and its angular velocity.

The sensors responsible for this action are the star tracker and the gyroscope, both defined in Section 4.3.1. On one hand, the purpose of the star tracker is to determine with high accuracy where the CubeSat is pointing and output that information as a quaternion. And, on the other hand, the 3-axis gyroscope provides information on how fast the CubeSat is changing its attitude. However, since the purpose is to point accurately, the setpoint for the angular velocity is always zero. Finally, once all the data is output by the sensors, it is sent to the control system to be processed.

5.1.2 Control system

To get the CubeSat pointing at specific points during all of its orbit, a proper control system is needed. The reaction wheels are the hardware responsible for doing the control, however, the right command must be sent

to achieve the desired position in short time and efficiently. To do so, a *Proportional Derivative* (PD) control is chosen. There are two reasons behind this decision [28]:

1. Controllers such as PID seem to work better in the theoretical level rather than actually being implemented in real CubeSats. PD controllers for CubeSats have more flight heritage as well as more case studies for implementation.

2. Even though PD being one of the most simple controllers, it does not have any optimized method for gain selection. Therefore, choosing any controller different from the PD would only further complicate the process without giving barely no gains in efficiency.

For the PD control system, a *Quaternion Feedback Controller* [29] is used, this is also the controller already implemented as default in the 42 simulator. The controller computes the control torque T_C based on the quaternion error and the CubeSat's angular velocity, following the equation below.

$$T_C = K_p \cdot q_e + K_d \cdot \omega \tag{5.1}$$

Where T_C is the commanded torque sent to the reaction wheels. Then, q_e is the error quaternion [30] between the current and the commanded quaternion. It is defined as

$$q_e = q_t \times q \tag{5.2}$$

Being q_t the body quaternion within the inertial frame (ECI) and q the desired quaternion. q_e is obtained through a quaternion multiplication. The angular velocity error ω is then computed as the difference between the current and the setpoint.

Regarding the feedback constant gains, "*they must be selected for asymptotic stability and transient performance of the system*" [29]. K_p represents the proportional gain which main purpose is to reduce rise time and the error in steady state. For the K_d, it is the derivative gain and it helps to reduce overshoot, increase stability and improve the system's transient response. To select both values, the option proposed in Reference [29] is chosen.

$$\begin{aligned} K_p &= I_s \cdot 2\omega_n^2 \\ K_d &= I_s \cdot 2\zeta\omega_n \end{aligned} \tag{5.3}$$

Where I_s is the CubeSat's moment of inertia in each axis, ζ is the damping ratio and ω_n is the natural frequency. Therefore, the two values to be adjusted for the control are the damping ratio and the natural frequency.

With that, the Attitude Control System designed is represented.

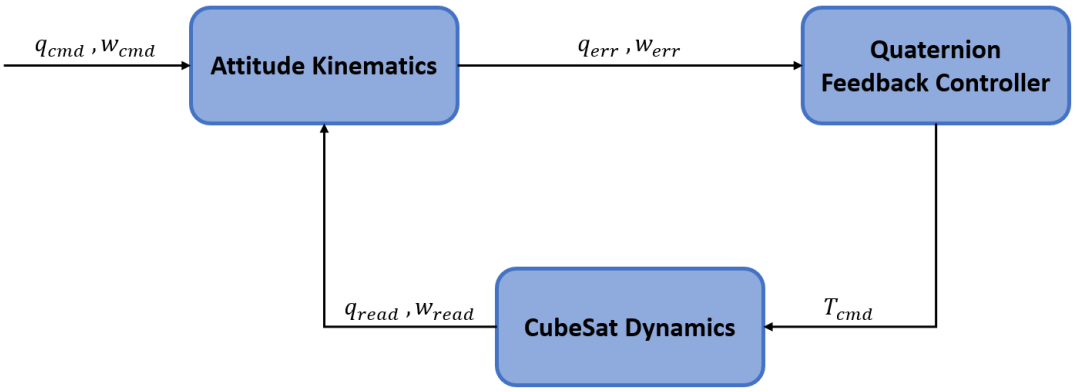


Figure 5.1: Attitude Control System Block Diagram.

The code responsible for executing all this tasks is summarised below.

```
/* Find PD Gains */
Kp = 2.0*I*w*w;
Kd = 2.0*I*z*w;
/* Form attitude error signals */
QxQT(AC->qbn,Cmd->qrn,qbr);
RECTIFYQ(qbr);
for(i=0;i<3;i++){
    C->therr[i] = qbr[i];
    C->werr[i] = AC->wbn[i] - Cmd->wrn[i];
}
/*Closed-loop attitude control */
for(i=0;i<3;i++) {
    C->Tcmd[i] = C->Kp[i]*C->therr[i]+C->Kd[i]*C->werr[i];
}
for(Iw=0;Iw<AC->Nwhl;Iw++) {
    W->Tcmd = Limit(VoV(AC->Tcmd,W->DistVec),-W->Tmax,W->Tmax);
}
```

The control system begins computing the controller gains following Equation 5.3. Then, the quaternion error `qbr` is obtained and rectified. The quaternion rectification consists on changing the whole quaternion sign when the fourth value is negative, it is done to maintain it always positive for numerical stability purposes.

After that, the error signals are calculated through the Quaternion Feedback Controller and the control torque can be obtained. To end the loop, each command is assigned to its corresponding wheel and limited to the specifications maximum values (see Appendix C for full code).

Once the control is totally defined, an step to point at Earth from its initial attitude is applied to the system, this maneuver can be considered as a worst case since it is way more aggressive than the ones that the CubeSat actually performs. The natural frequency and damping ratio values are set to its default by the 42, then, both the setpoint and the system response in each reaction wheel axis are plotted together.

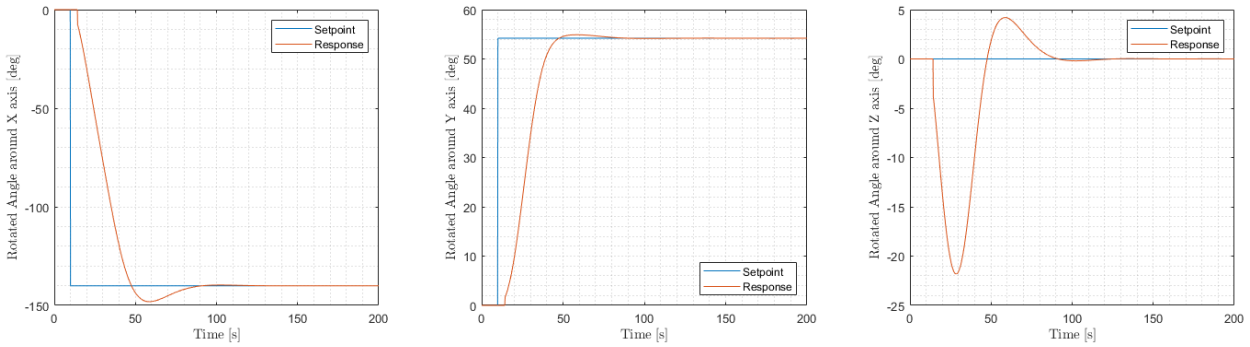


Figure 5.2: CubeSat System response to an Earth pointing step with $\omega_n = 0.1$ rad/s and $\zeta = 0.707$.

Analysing the Figure above, it can be seen that the system reaches the setpoint about a 100 seconds after sending the command. This seems quite a slow system behaviour, however, doing a rigorous study on the effect of variation of the natural frequency and damping ratio values, could improve the system response. In addition, it is verified that the system behaves as one of second order, hence the control system designed is suitable for the case.

From now on, to analyse the system behaviour, only the response on X axis is visualized since it gives a clear view on how the system acts. To improve the system behaviour, both the natural frequency and damping ratio need to be adjusted, therefore an analysis is done. For the natural frequency, different values are tested and the responses are plotted altogether. To do this, the damping ratio is fixed to its default value $\zeta = 0.707$.

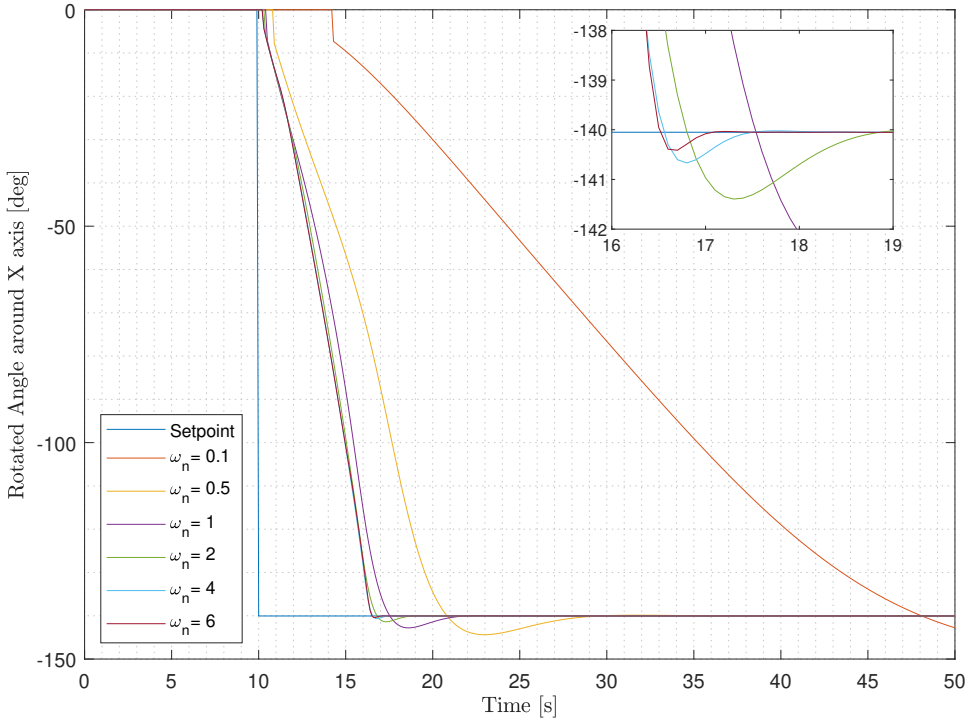


Figure 5.3: CubeSat System response to variable ω_n [rad/s] with fixed $\zeta = 0.707$.

As it is seen, increasing the natural frequency mainly reduces the settling time and slightly reduces the overshoot. For instance, a big jump in settling time is observed from $\omega_n = 0.1$ rad/s to $\omega_n = 0.5$ rad/s where it goes from 90 to 20 seconds (a 70% reduction). After that, increasing the value to $\omega_n = 1$ rad/s reduces the settling time by 8 seconds and from there on, surpassing the unity reduces the rise time by $3 - 5$ seconds.

With this data, it is concluded that the natural frequency need to be set at a value $\omega_n \geq 0.5$ rad/s, otherwise the system would be too slow. To decide an exact value with only this data is a tough task. Observing Figure 5.3 seems that increasing the natural frequency only improves the system behaviour, however, to achieve those high speeds in response it may require an excessive angular velocity at the reaction wheels, surpassing its maximum. Due to that, the angular velocity of the reaction wheels must be analysed during flyby maneuvers to ensure it does not reach its maximum and the CubeSat can point correctly. This study is done in Section 6.1 where different flyby are analysed.

For the adjustment of the damping ratio, the natural frequency is fixed at $\omega_n = 1$ rad/s and different values are tested.

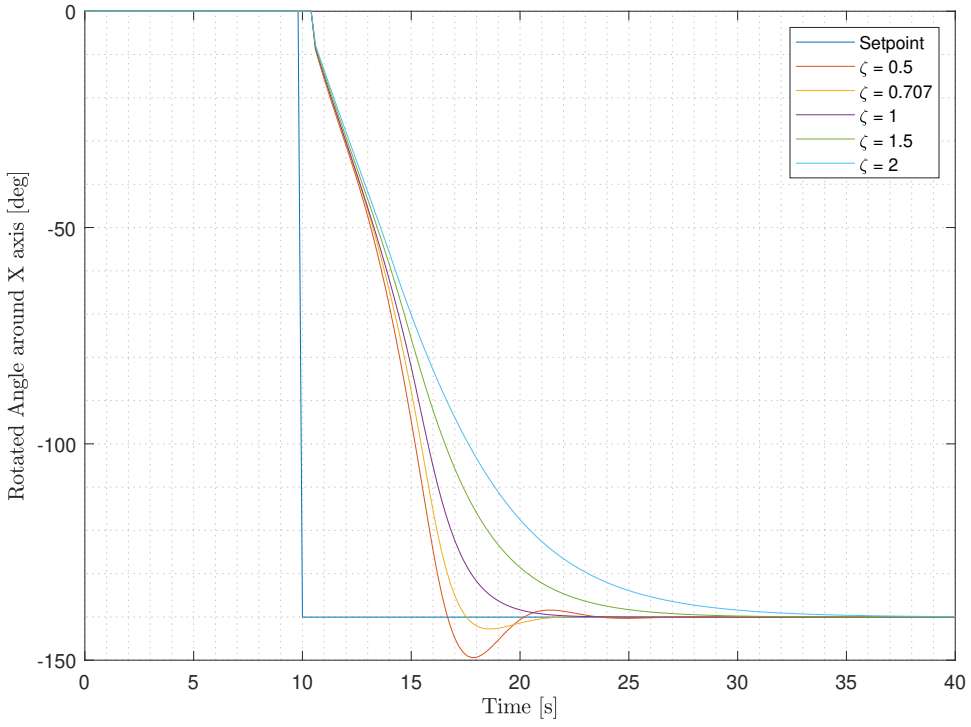


Figure 5.4: CubeSat System response to variable ζ with fixed $\omega_n = 1$ rad/s.

It is observed that the system reacts as in the theory for the different damping ratio values. For a value of $\zeta = 0.707$, the response has an slight overshoot while for $\zeta = 1$ the system is critically dumped and no overshoot appears. Since both values provide the lowest settling time, a damping ratio of $\zeta = 1$ is chosen for the control system. Doing so, the system can achieve the setpoint in short time and efficiently avoiding an overshoot.

To sum up, the conclusions extracted from this analysis are in the table below.

Natural Frequency [rad/s]	Damping Ratio
$\omega_n \geq 0.5$	$\zeta = 1$

Table 5.1: Control System gain parameters values after the analysis.

5.2 Active control zones

As the CubeSat approaches the ground stations, it needs to actively point at them to achieve the higher antenna gains available. In Section 4.4.2, it is defined how the CubeSat points at them but it is not defined when to point at them. To do so, two control zones for each ground stations are set: the Approximation Control Zone (ACZ) and the Visible Control Zone (VCZ). Thus, once the flight software detects that the CubeSat is inside ACZ it begins to point at the ground station, giving enough time to enter the VCZ with a perfectly pointed CubeSat and ensure the most efficient communication.

For the ground stations antennas, a minimum elevation angle of 10 deg is set. Therefore, the VCZ is defined.

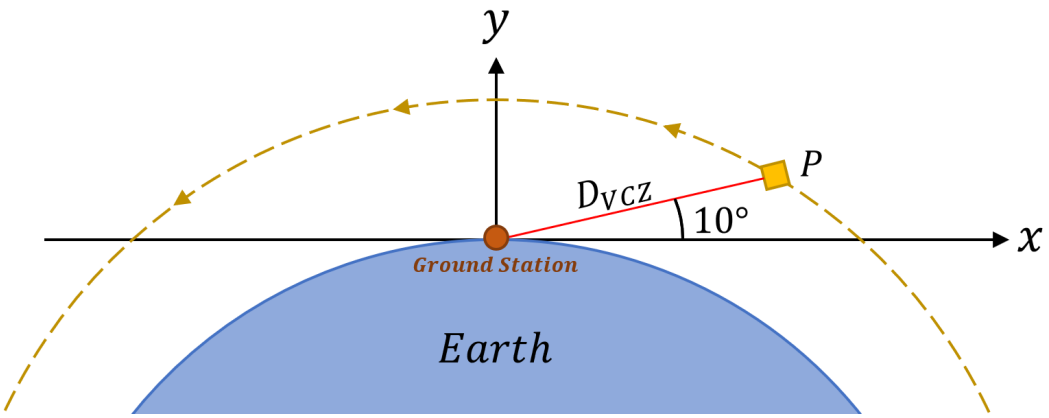


Figure 5.5: Visible Control Zone representation.

The red line and the CubeSat circular orbit equations are stated and solved.

$$\begin{aligned} y &= tan(10) \cdot x \\ x^2 + (y + R_T)^2 &= R_{CS}^2 \end{aligned} \tag{5.4}$$

Solving the equation system for a CubeSat orbit radius of $R_{CS} = R_T + 500$ km and an Earth radius of $R_T = 6371$ km,

$$\begin{aligned} x_P &\approx 1669\ km \\ y_P &\approx 294\ km \\ D_{VCZ} &= \sqrt{x_P^2 + y_P^2} \approx 1695\ km \end{aligned} \tag{5.5}$$

With that, the VCZ is defined at a distance of $D_{VCZ} = 1695$ km from the ground station.

For the ACZ, since the settling time is expected to be around 20 seconds as maximum for the control designed, this margin is given to the CubeSat. To obtain the margin value in distance units, the orbiting speed must be calculated.

$$V_{orbit} = \sqrt{\frac{G \cdot M_T}{R_{CS} \times 10^3}} = 7615\ m/s \equiv 7.615\ km/s \tag{5.6}$$

Where, G is the gravitational constant and M_T is the Earth's mass. Then,

$$d_{margin} = V_{orbit} \cdot t_{margin} = 7.615 \cdot 20 = 152.3\ km \tag{5.7}$$

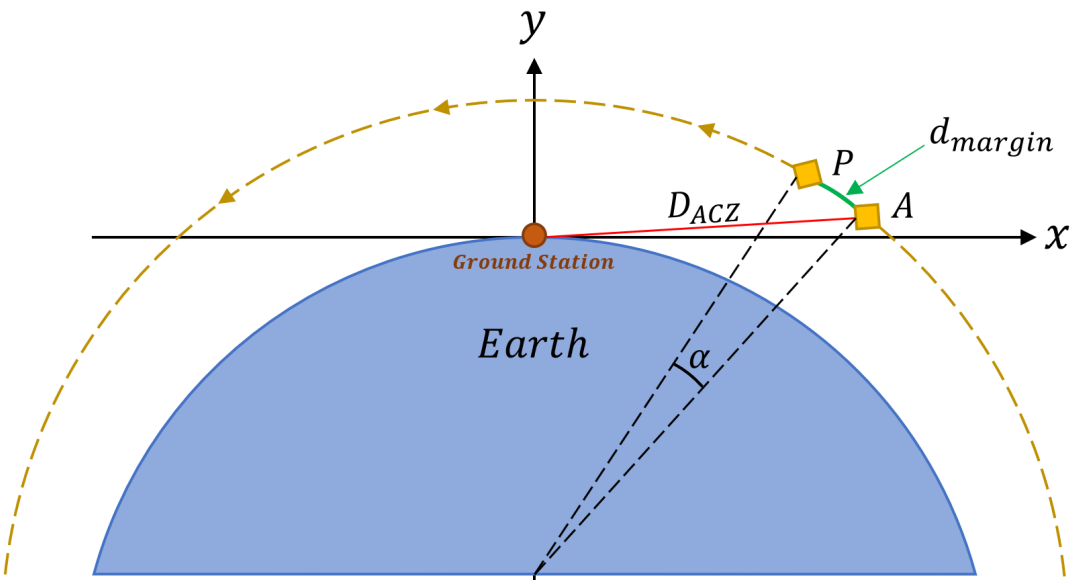


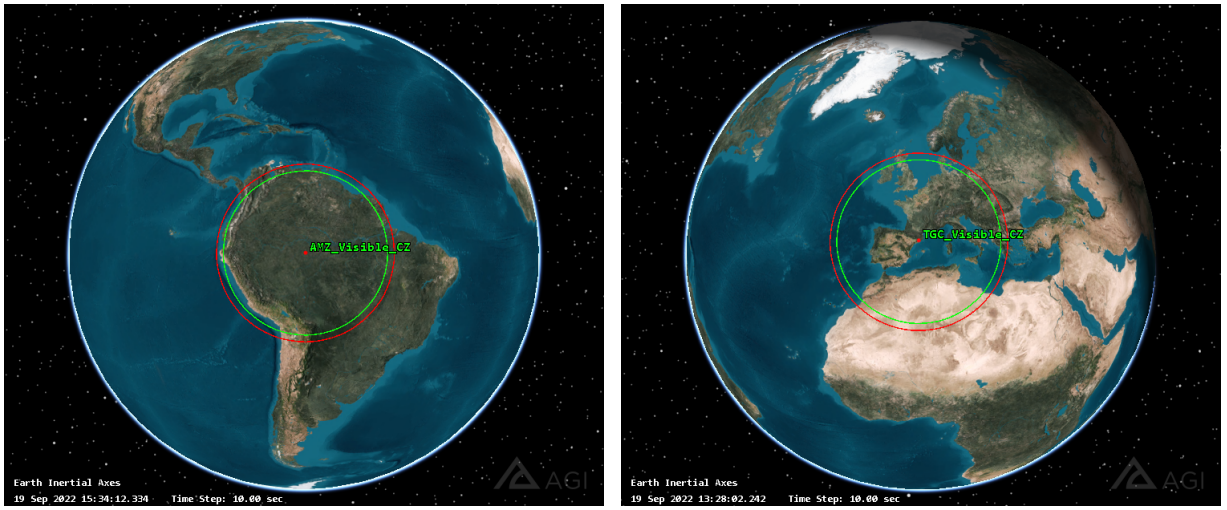
Figure 5.6: Approximation Control Zone representation.

Knowing the coordinates of point P, point A coordinates can be computed.

$$\alpha = \frac{152.3 \cdot 360}{2\pi \cdot R_{CS}} = 1.27\ deg \tag{5.8}$$

$$\begin{aligned} x_A &= x_P \cdot cos(\alpha) + (y_P + R_T) \cdot sin(\alpha) \approx 1816\ km \\ y_A &= -x_P \cdot sin(\alpha) + (y_P + R_T) \cdot cos(\alpha) - R_T \approx 255\ km \\ D_{ACZ} &= \sqrt{x_A^2 + y_A^2} \approx 1834\ km \end{aligned} \tag{5.9}$$

With that, the ACZ is defined at a distance of $D_{ACZ} = 1834$ km from the ground station. To give a clearer view, the control zones from space can be seen below.



(a) Amazon Control Zones from space. (b) Terrassa Control Zones from space.

Figure 5.7: VCZ in green and ACZ in red viewed from space.

Once both control zones are defined, a criteria to consider that the CubeSat is communicating correctly with the ground station must be established. The first obvious condition is that the CubeSat must be inside the VCZ. As a second condition to ensure efficient communication, a ≤ 5 degree error in the pointing angle is set. This value comes from the radiation pattern of the parabolic antenna, design and developed by *Xavier Pozo Diaz* [26] for the DISEN Group.

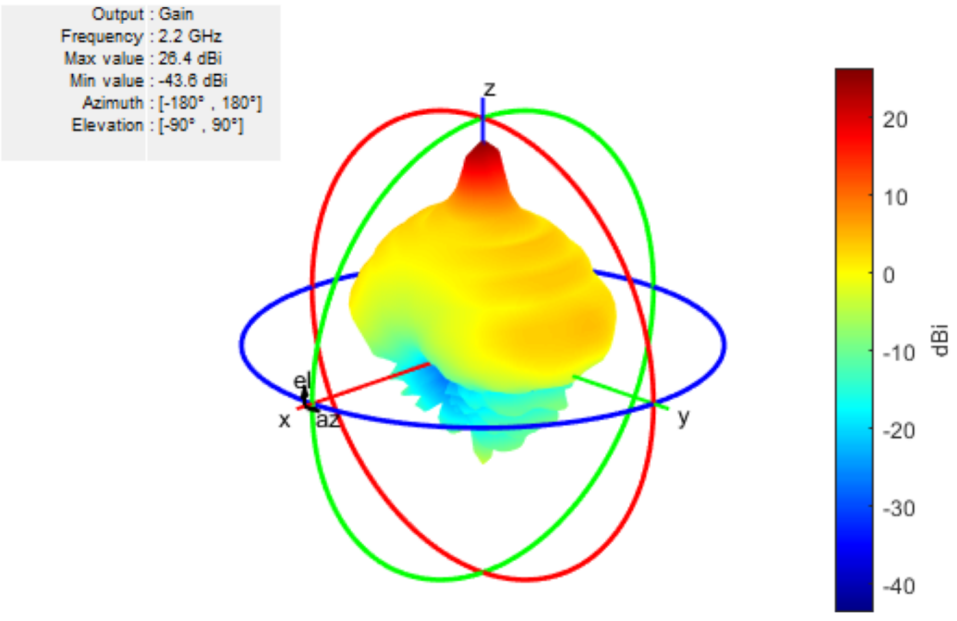


Figure 5.8: 3D Radiation pattern of the parabolic antenna. Source: [26].

Projecting in the XZ plane,

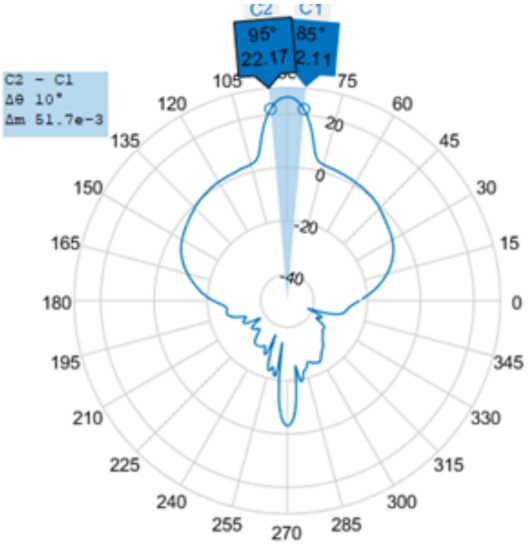


Figure 5.9: Projected beamwidth of the parabolic antenna. Source: [26].

It can be seen that setting an error in the pointing angle ≤ 5 deg provides the highest gain range, which is between $G = [22.17, 26.42]$ dB, hence an efficient communication is ensured.

5.3 Battery Management System

The BMS is one of the critical systems inside a CubeSat, responsible for managing the power and ensuring an efficient usage in order to maximize the batteries lifespan. As specified in Section 4.3.1, the CubeSat is powered by a battery pack with a voltage of 9.6 V and a charging capacity of 1500 mAh resulting in the energy capacity below.

$$C_W = C_A \cdot V = 1.5 \cdot 9.6 = 14.4\ Wh \equiv 51840\ Ws \tag{5.10}$$

These batteries are recharged with four solar arrays with a 90% efficiency and covering 90% of each four faces. The faces which are not covered by solar arrays are the one facing the Earth during the mission, since it is already covered by the communication parabolic antenna and its opposite for sensor positioning and simmetry purposes. However, the solar arrays placement and its properties should be discussed in future research.

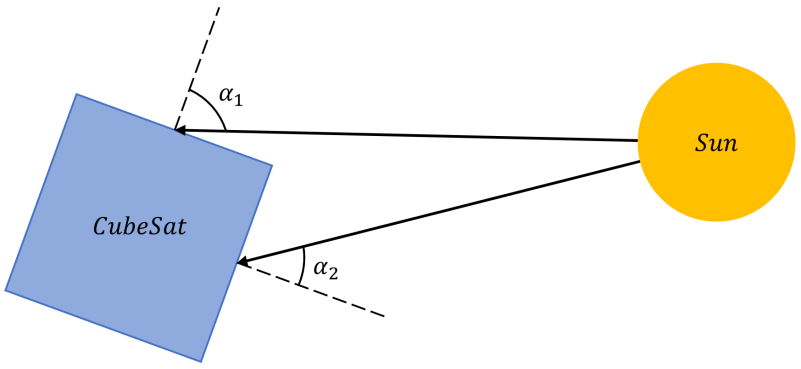


Figure 5.10: Two Sun vector angles representation respect to the Body Frame.

Then, the sun energy absorbed is

$$E_{in} = Irradiance \cdot \eta_{SA} \cdot \eta_{face} \cdot A_{face} \cdot [cos(\alpha_1) + cos(\alpha_2) + cos(\alpha_3)] \cdot \Delta t \tag{5.11}$$

Where the irradiance value is the one set by the World Meteorological Organization at $1367 W/m^2$ [31], η_{SA} is the solar arrays efficiency, η_{face} is the percentage of face area covered by solar arrays and A_{face} is the face area of a 10 cm side cube. For the cosine factor, it represents the portion of sunlight that illuminates each Sun visible face perpendicularly and is computed as the sun vector projected over the body face normal.

Regarding the consumption, the case is simplified to only considering the reaction wheels with a motor efficiency of $\eta = 50\%$, hence the energy consumption is computed.

$$E_{out} = \frac{|T \cdot \omega| \cdot \Delta t}{\eta} = 2 \cdot |T \cdot \omega| \cdot \Delta t \tag{5.12}$$

To optimize the BMS and reduce its degradation, the batteries only power the CubeSat whenever the energy required is greater than the absorbed from the sun. In this case, the energy absorbed is totally consumed and the batteries only provide the energy left. However, if the energy absorbed is greater than the required, the required amount is consumed and the energy left is used to recharge the batteries. The system is depicted below.

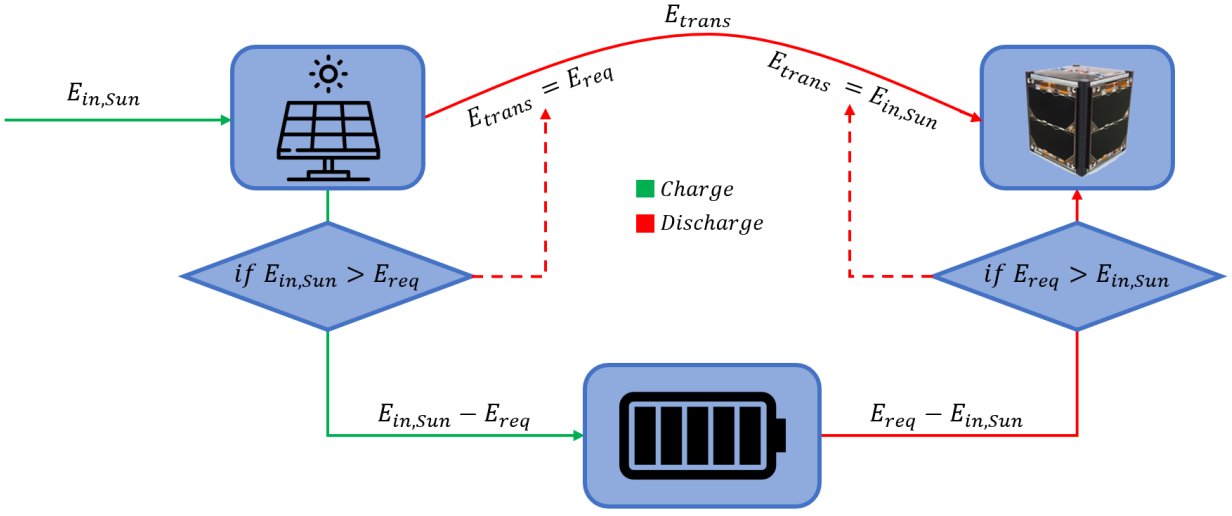


Figure 5.11: Battery Management System of the Amazon Mission CubeSat.

As a model of the CubeSat's constant power consumption, only the reaction wheels are considered. This allows to check if the solar arrays modeled are able to restore all the energy consumed by the reaction wheels, otherwise either the control system or the reaction wheels model must be checked. In advanced stages of the *Amazon Mission*, a refined model of the CubeSat's power consumption can be implemented to obtain more accurate results.

The code responsible for the BMS of the CubeSat is shown below.

```
1 /*SOLAR ARRAYS ENERGY ABSORPTION*/
```

```
if (CSS->Valid == TRUE) {
    MxV(S->B[CSS->Body].CN,S->svn,svb);
    for (i=0;i<3;i++) {if (svb[i] < 0) svb[i] = -svb[i];}
    Irr_fac = svb[0] + svb[1];
    Ei_sun = 1367*Irr_fac*0.9*0.9*0.1*0.1*DTSIM;
}
else Ei_sun = 0;

/*REACTION WHEELS CONSUMPTION*/
for (i=0;i<3;i++) {
    Eo_wheel = Eo_wheel + fabs(2*AC->Whl[i].Tcmd*AC->Whl[i].w*DTSIM);
}

/*FINAL ENERGY BALANCE*/
E_total = Ei_sun - Eo_wheel;
exp_batt = C->Ebatt + E_total;
if (exp_batt < 51840) C->Ebatt = C->Ebatt + E_total;
else C->Ebatt = 51840;
```

The algorithm is divided into three parts: the first computes the energy absorbed by the solar arrays following Eq. (5.11). To obtain the cosine factor, the Sun vector in N frame (`svn`) is transformed to the Body frame using the appropriate DCM. Then, the X, Y components of the Sun vector are made absolute because if a face is hidden from the Sun, its parallel face, which is illuminated, experiences the same inclination relative to the Sun. Doing so, all the cosine factors are obtained. After that, the components associated with X, Y axis are added and stored as a variable called `Irr_fac`. To end, Eq. (5.11) is coded and in case the Coarse Sun Sensor detects that the CubeSat is eclipsed by the Earth, the energy absorbed is null.

The second part of the algorithm, computes the energy consumed by all three reaction wheels following Eq. (5.12). Finally, the batteries are either recharged or consumed depending on the energy values compute before and, if the recharged value exceeds the maximum capacity of the batteries, it is limited to $C_W = 51840$ Ws. It may seem that the complex BMS depicted in Figure 5.11 is not implemented in the code, however, it is. The BMS can be very complex in hardware terms but it is comparatively simpler to code (see Appendix C for full code).

5.4 Data flow

As the CubeSat flyby both ground stations, it exchanges data with them. During a flyby over TGC, the CubeSat transmits data at a rate of 512 kbps with a power of 1 W while during a flyby over AMZ, the CubeSat receives data at the same rate of 512 kbps with a negligible power compared to the consumption in transmission operations.

With that information, a study on the data flow between CubeSat and ground stations as well as the power consumption due to data transmission is done. The CubeSat has a power consumption of 1 W in transmission,

therefore the energy consumed during a flyby over TGC is computed.

$$E_{msg} = P_{ant} \cdot t_{msg} = 1 \cdot t_{msg} = t_{msg}\ Ws \tag{5.13}$$

where t_{msg} is the time in seconds that the CubeSat is within the VCZ and with a pointing angle error ≤ 5 deg as defined in Section 5.2.

For the amount of data exchanged, the same t_{msg} is used, then

$$Data\ exchanged = 512kbps \cdot \frac{1\ MB/s}{8000\ kbps} \cdot t_{msg} = 0.064 \cdot t_{msg}\ MB \tag{5.14}$$

```
1  if (AC->TGCMSG_FLAG == 1) {
2      Eo_msg = AC->elapsed_time;
3      msg_data = 0.064*AC->elapsed_time;
4      msg_batt = 100*(Eo_msg/C->Ebatt);
5      C->Ebatt = C->Ebatt - Eo_msg;
6      AC->tgc_total_data = AC->tgc_total_data + msg_data;
7      AC->tgc_cont = AC->tgc_cont + 1;
8      AC->TGCMSG_FLAG = 0;
9      printf("Visible during T = %fs. Data transmitted: %fMB. Battery consumed: %f%%. Battery
       left: %f%%\n\n",AC->elapsed_time,msg_data,msg_batt,100*C->Ebatt/51840);
10 }
11 else if (AC->AMZMSG_FLAG == 1) {
12     msg_data = 0.064*AC->elapsed_time;
13     AC->amz_total_data = AC->amz_total_data + msg_data;
14     AC->amz_cont = AC->amz_cont + 1;
15     AC->AMZMSG_FLAG = 0;
16     printf("Visible during T = %fs. Data received: %fMB.\n\n",AC->elapsed_time,msg_data);
17 }
```

The code responsible for executing this functionalities, is separated in two main parts, one for each ground station. Each time the CubeSat flyby TGC the energy consumed by the message sent as well as its size are computed (lines 2 and 3). In addition, the battery percentage consumed by the message is obtained and the batteries total energy is updated (lines 4 and 5). To finish, the data size is added to the total amount sent during the simulation, the counter is updated and the flag is reset (lines 6 to 8), with that, a summary of the flyby is printed on the terminal. For the AMZ flyby, the code is simpler, the amount of data received is computed then added to the total amount, the counter is updated and the flag is reset (lines 12 to 15). As for the TGC flyby, a summary message is printed on the terminal (see Appendix C for full code).

Chapter 6

Amazon Mission. Results

Once all the aspects inside the scope of this thesis are fully developed, mission simulations are run and the main outputs analysed. This chapter focuses on the outputs, it analyses the most relevant results and extracts conclusions on the performance of the simulator. It serves to detect improvement areas and to check whether or not the decisions taken were appropriate.

The chapter is divided into four sections, the first one adjusts two parameters that were not set before due to needing flyby simulation to decide them. The three left are different scenarios of the mission which are analysed separately to be able to focus much more on them. It begins with both flyby over the two ground stations that take part in this mission and, after that, a single orbit simulation is done.

6.1 Natural frequency and update rate adjustment

Since the most critical part of the mission is the pointing performance during a flyby, parameters that directly influence this aspect need to be carefully adjusted. This section set final values to those parameters through analysing flyby performance.

As specified in Section 5.1.2, to set a suitable value for the natural frequency (ω_n) of the control system, the angular velocity of the reaction wheels must be taken into account. Low settling times may require high angular velocity changes and, therefore, reaction wheels may achieve too high speeds. Below, the angular velocity of all three reaction wheels is plotted during, what has been considered, a demanding flyby for a natural frequency value of $\omega_n = 1$ rad/s.

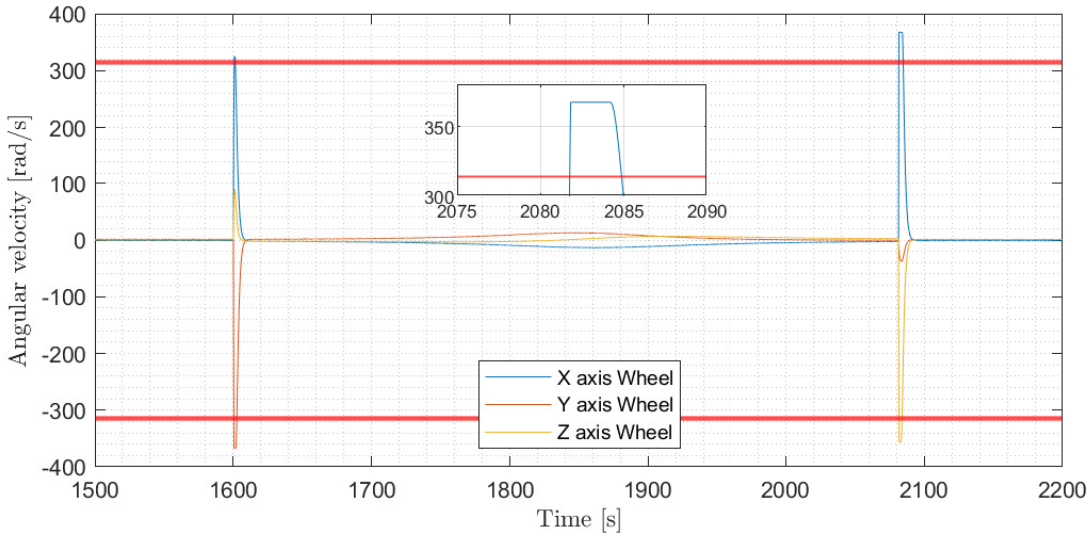


Figure 6.1: Angular velocity of the reaction wheels during a flyby ($\omega_n = 1$ rad/s).

Even though this value does not provide the lowest settling time as seen in Figure 5.3, it is more than enough to exceed maximum angular velocity values with all three reaction wheels. Hence, this natural frequency value cannot be accepted and a lower one must be chosen. For this second iteration, a value of $\omega_n = 0.6$ rad/s is proposed.

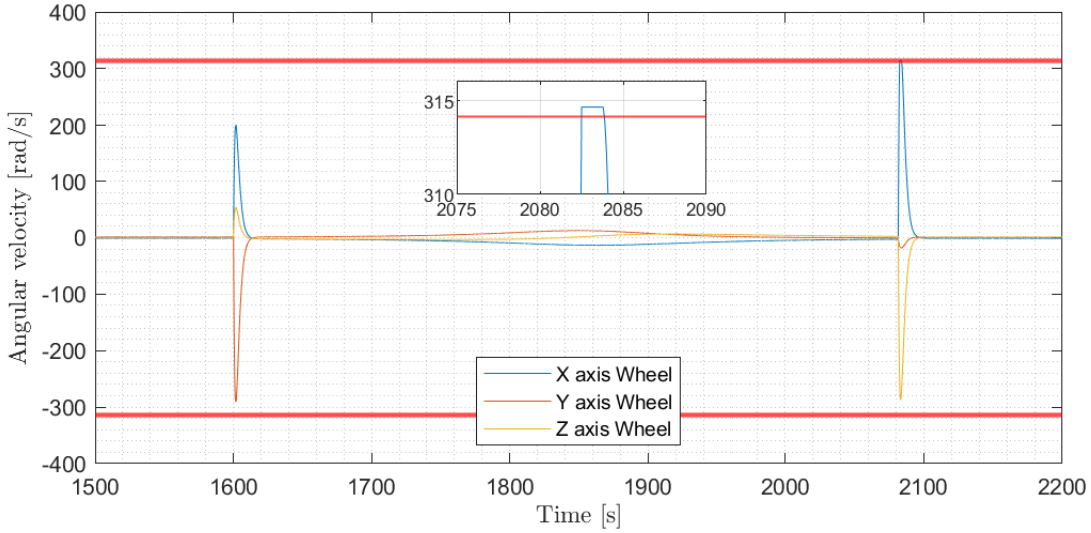


Figure 6.2: Angular velocity of the reaction wheels during a flyby ($\omega_n = 0.6$ rad/s).

For the new natural frequency set, the reaction wheel acting along the X axis, surpasses its maximum angular velocity by about 1 rad/s when exiting the VCZ. As it is seen, the system limits its value (flat peak) and yet continues to surpass it. Although the value is slightly above the maximum, it cannot be accepted since operating at maximum rates deteriorates the components much faster than at nominal rates. Due to that, a lower value of $\omega_n = 0.5$ rad/s is set.

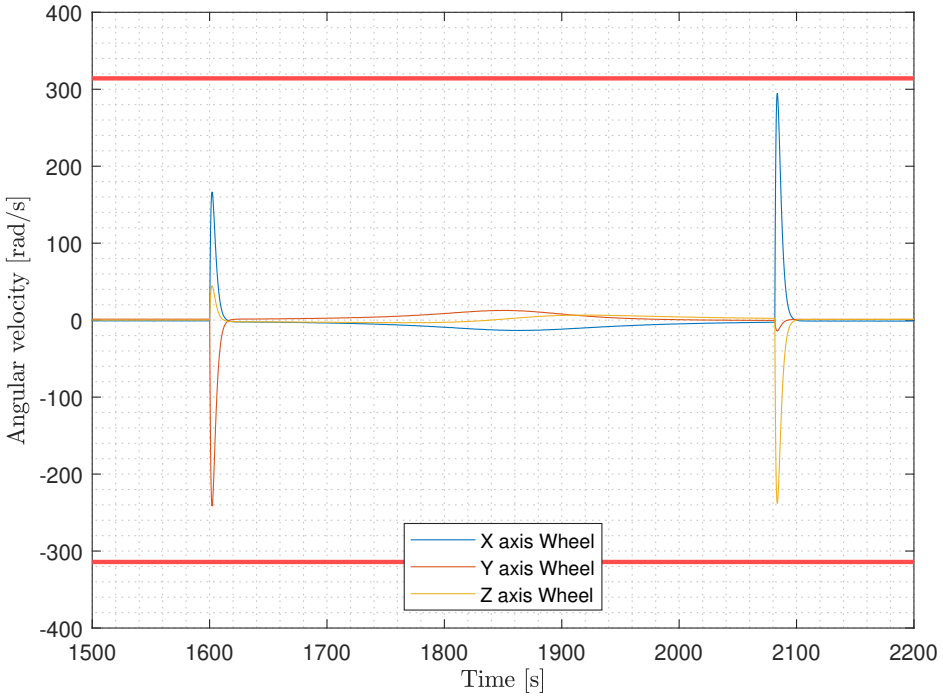
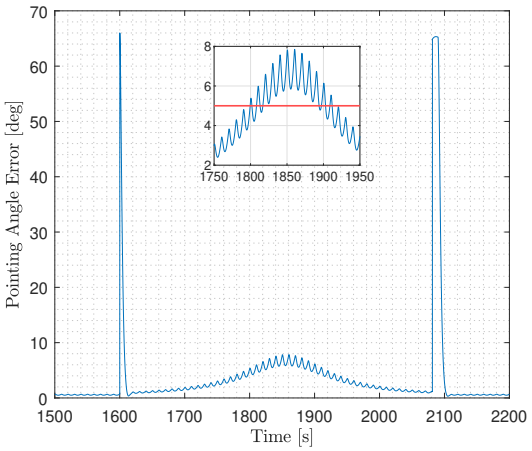


Figure 6.3: Angular velocity of the reaction wheels during a flyby ($\omega_n = 0.5$ rad/s).

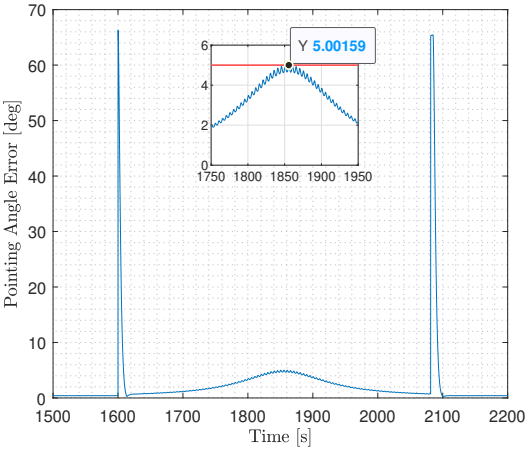
It is clearly seen that this value allows a control of the CubeSat without forcing the reaction wheels to its maximum angular velocity. This ensures a reliable operation during longer periods of time than in the previous cases. In addition, as seen in Figure 5.3, with this new value set the system can reach a setpoint in around 20 seconds. This settling time may be generally lower since the setpoint values given during a flyby are usually shorter.

Another important value to be decided for the flight software is the update rate (UR). It represents the rate at which a new command is sent to the reaction wheels. Since the system has a settling time of about 10 to 20 seconds for an standard pointing command, it makes sense to do a study on the UR and understand if the system needs time to settle and how much is it.

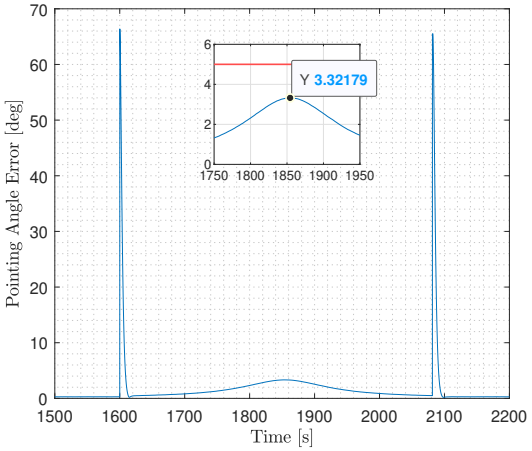
The value to be analysed in this study is the error in the pointing angle because as the CubeSat travels through its orbit, the target vector changes constantly and the error in pointing may vary. Below, the error in the pointing angle during the same flyby analysed for the natural frequency adjustment is plotted for different UR values.



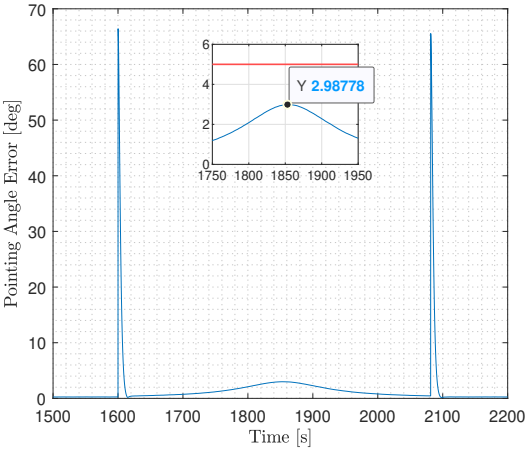
(a) Pointing error for UR = 10 s.



(b) Pointing error for UR = 5 s.



(c) Pointing error for UR = 1 s.



(d) Pointing error for UR = 0.1 s.

Figure 6.4: Pointing angle error during a flyby for multiple UR values.

The Figure above shows a clear feature, the lower the update rate, the smoother the pointing error line and the lower the peak value. Although the system needs between 10 to 20 seconds to stabilise a setpoint, when commanding multiple setpoint the best practice is to reduce the time between them as much as possible. However, a UR = 0.1 s can only be achieved if all the CubeSat sensors can operate at a frequency $\geq$ 10 Hz, otherwise the update rate is limited to the lowest frequency value.

By setting a low UR value, the signal shaking is avoided as well as the lowest peak pointing error is achieved. To understand the oscillations in Figure 6.4a, it needs to be taken into account that the system spends 10 seconds trying to reach a setpoint which is not already pointing where it needs to. Due to passing by near the zenith of the ground station, the setpoint values suffer bigger changes and, if the update rate is not low enough, the pointing error keeps oscillating.

Those oscillations must be avoided in order to maintain the CubeSat pointing accuracy below the 5 degree criteria set, highlighted with a red line in Figure 6.4, an ensure an efficient communication. Therefore, since the sensors selected have a frequency $\geq$ 10 Hz, the update rate set for the *Amazon Mission* is 0.1 seconds, the lowest value achievable limited by the Star Tracker frequency.

6.2 Scenario A. AMZ flyby

This scenario considers an overnight flyby over the Amazon Ground Station, occurring on the morning of the September 20th, 2022. The detailed time and the space view of the flyby can be seen below.

VCZ Entry Time (UTC +00)	VCZ Exit Time (UTC +00)
20/09/2023 at 07:03:32	20/09/2023 at 07:10:27

Table 6.1: AMZ flyby timings.

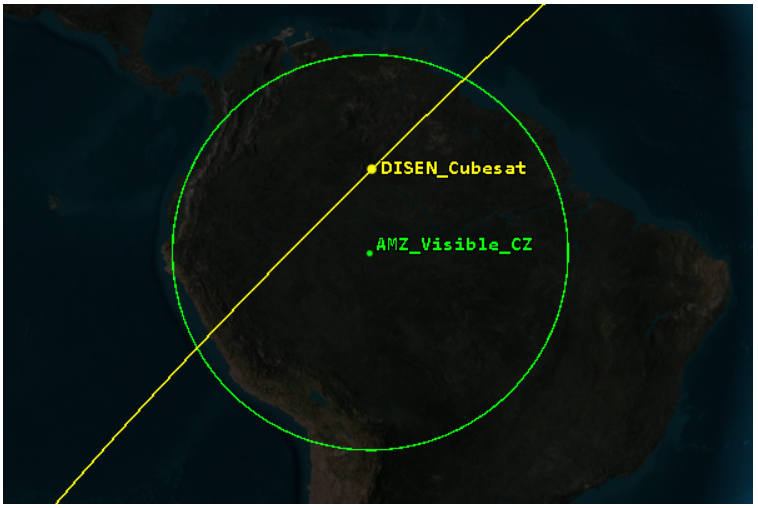


Figure 6.5: AMZ overnight flyby path seen from space.

For this flyby, three critical points are analysed: the pointing accuracy responsible for enabling communication, the battery consumed and the data sent from the ground station to the CubeSat. It is important to notice that sun and moon exclusion angles of the star tracker, have been set to 0 during this simulation to analyse a clean flyby without perturbations. The effects of the exclusion angles are analysed in Scenario C (Section 6.4).

6.2.1 Pointing accuracy

The pointing angle error during the flyby is plotted below. The green line highlights the moment at which the CubeSat enters the ground station visible range and, the red line, the moment at which it exits the ground station visible range. Therefore, the time elapsed between lines is the time when the CubeSat is communicating with the ground station.

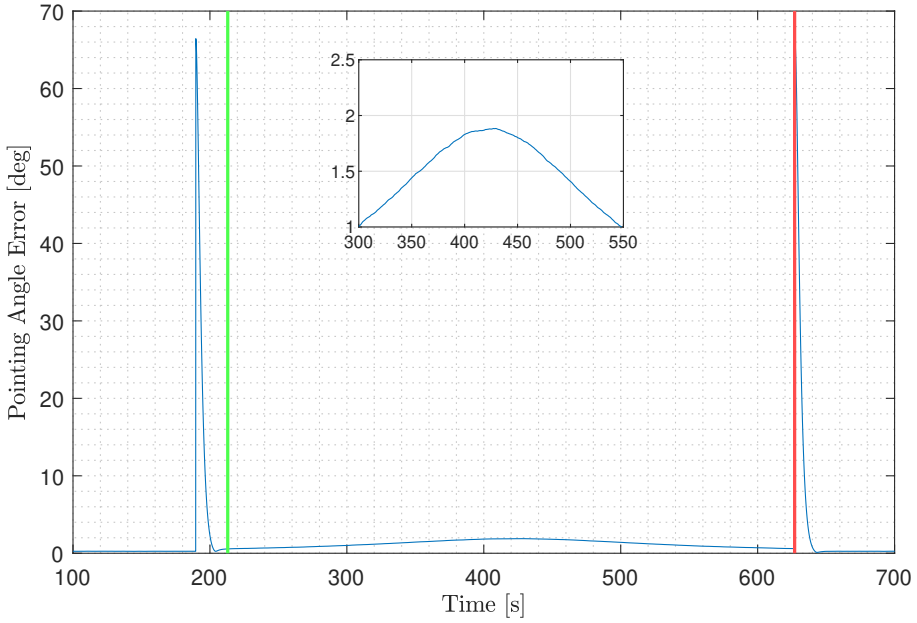


Figure 6.6: Pointing angle error during AMZ flyby.

As it is seen, thanks to the control system and flight schedule designed, the CubeSat maintains its pointing accuracy below 2 degrees during the whole flyby. This allows to communicate with high antenna gains and, therefore, a more efficient communication. To ensure that the reaction wheels do not suffer during its operation and no maximum values are reached, the angular velocity of the reaction wheels is also analysed.

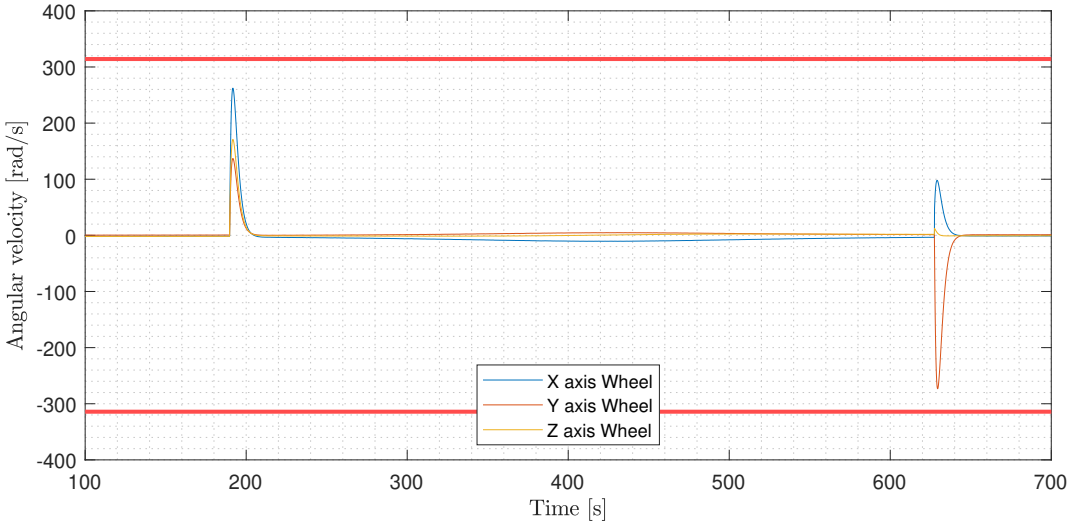


Figure 6.7: Angular velocity of reaction wheels during AMZ flyby.

It is confirmed that the reaction wheels, although being close to the maximum rates at some points, do not surpass at any moment the maximum angular velocity values.

6.2.2 Battery life

Regarding the battery consumed during the flyby, it is considered that the power consumption while receiving data is negligible, hence the reaction wheels are the only consumption source. The reason behind choosing an overnight flyby is to be able to analyse better the absolute wheel consumption without interfering with the energy absorbed by the solar arrays.

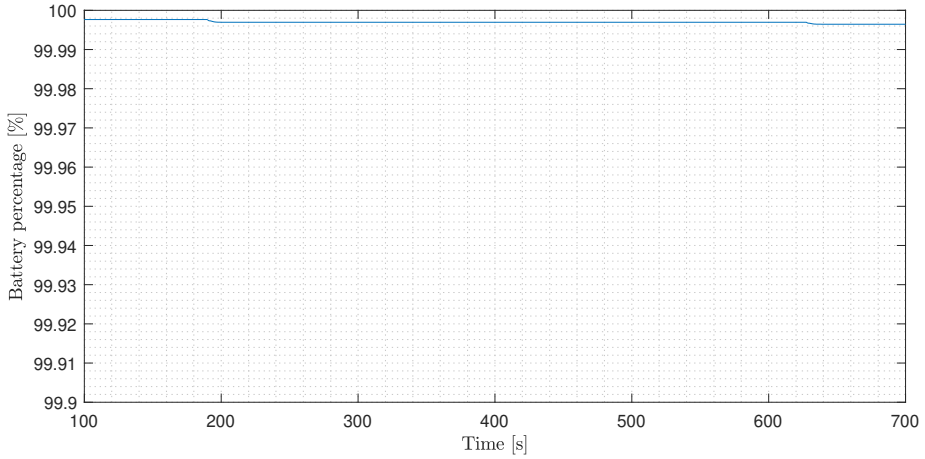


Figure 6.8: CubeSat battery percentage during AMZ flyby.

It is noted that the reaction wheels barely consume battery during its pointing procedures. Due to having relatively small changes in the attitude, consumption is low and it can be stated that reaction wheels are not a limiting factor when looking at battery consumption.

6.2.3 Data transfer

For the data sent from the Amazon ground station to the CubeSat, the visible time value is used. In the flyby analysed, the CubeSat is enabling communication during 414.5 seconds at a rate of 512 kbps which means an amount of data received of about 26.5 MegaBytes[8]. The values are summarised below.

Visible time [s]	Data received [MB]
414.5	26.528

Table 6.2: Data transfer summary during AMZ flyby.

[8]International System of Units, equals to 10^6 bytes.

6.3 Scenario B. TGC flyby

The second scenario considers a flyby over the Terrassa Ground Control, occurring on the morning of the September 19th, 2022. The detailed time and the space view of the flyby can be seen below.

VCZ Entry Time (UTC +00)	VCZ Exit Time (UTC +00)
19/09/2023 at 08:27:01	19/09/2023 at 08:34:41

Table 6.3: TGC flyby timings.

Figure 6.9: TGC flyby path seen from space.

As for the flyby analysed in Scenario A, three points are analysed: the pointing accuracy, the battery consumed and the data sent from the the CubeSat to the ground station. Again, it is important to notice that sun and moon exclusion angles of the star tracker, have been set to 0 during this simulation to analyse a clean flyby without perturbations.

6.3.1 Pointing accuracy

The pointing angle error during the flyby is plotted below. As before, the green line highlights the moment at which the CubeSat enters the ground station visible range and, the red line, the moment at which it exits the ground station visible range.

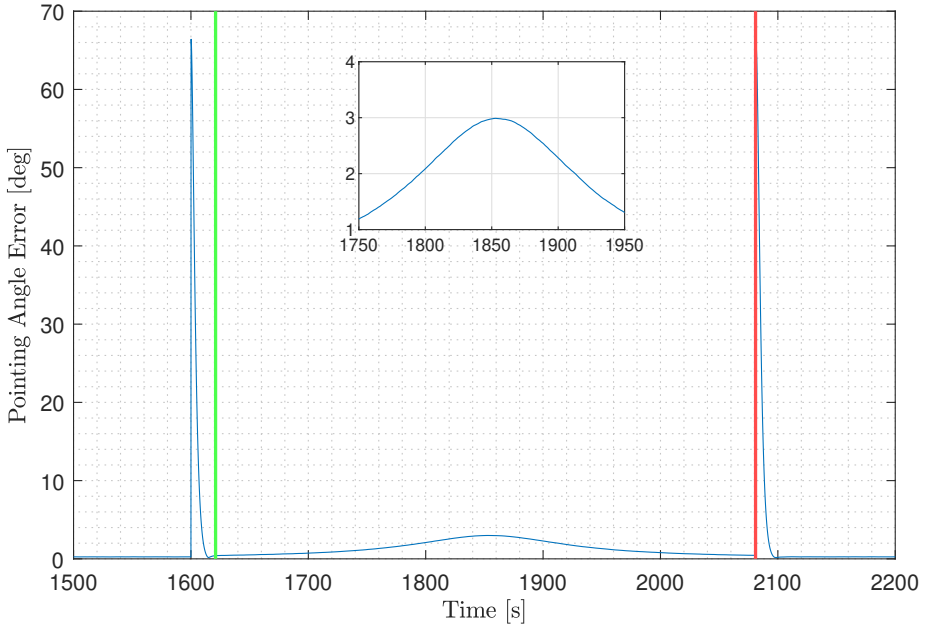


Figure 6.10: Pointing angle error during TGC flyby.

As it can be seen, the CubeSat maintains its pointing accuracy below 3 degrees during the whole flyby. This allows to communicate more efficiently. As a difference from the previous scenario, it is noted that the peak value, corresponding to the moment at which the CubeSat flies over the zenith of the ground station, has increased by 1 degree. This can be explained comparing Figures 6.5 and 6.9 and noting that for the TGC flyby the CubeSat passes closer to the ground station. The shorter the distance to the ground station, the bigger the changes in attitude and the slightly higher the peak in pointing angle error.

To ensure that the reaction wheels do not suffer during its operation and no maximum values are reached, the angular velocity of the reaction wheels is also analysed.

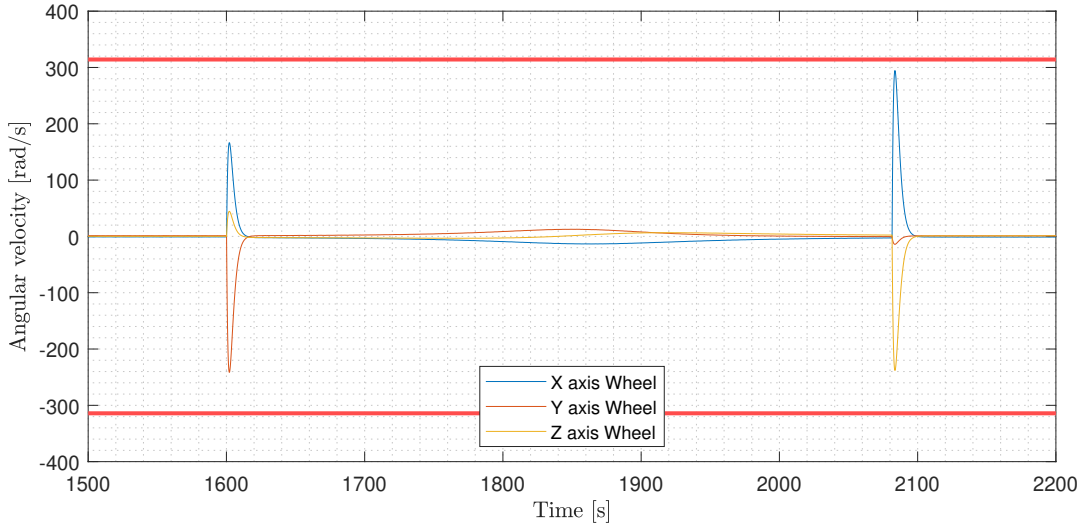


Figure 6.11: Angular velocity of reaction wheels during TGC flyby.

As in Scenario A, it is confirmed that the reaction wheels do not surpass at any moment the maximum angular velocity values.

6.3.2 Battery life

Regarding the battery consumed during the flyby, the solar arrays are responsible for restoring the energy consumed by both the reaction wheels and the communication antenna.

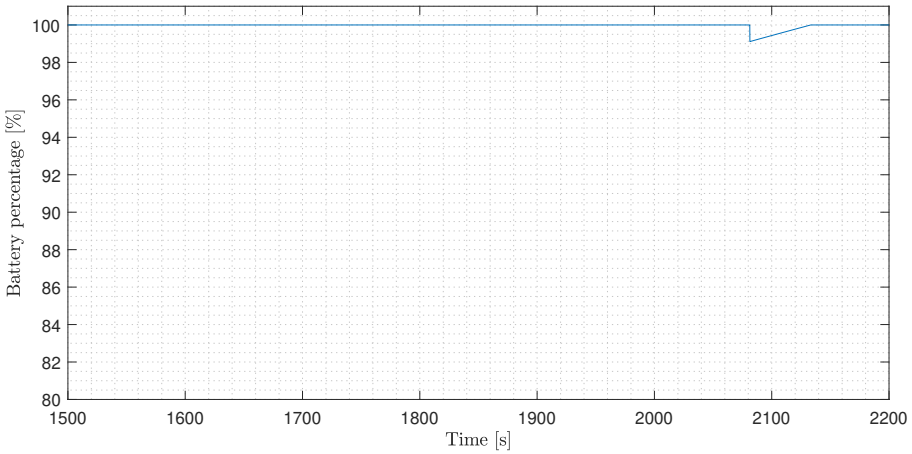


Figure 6.12: CubeSat battery percentage during TGC flyby.

As seen in Figure 6.8, the reaction wheels consumption is so low that the solar arrays easily restore the energy consumed leading to a straight battery percentage line during the flyby. The consumption peak found at about 2100 seconds, corresponds to the antenna consumption due to sending data to the ground station. Since the code is built to compute first the total visible time and then, obtain the amount data being sent along with the energy consumed, it appears as a peak whereas, in reality, it would actually be a much more continuous line. However, this gives a clearer picture of how much sending a message consumes.

6.3.3 Data transfer

For the data sent from the CubeSat to the Terrassa Ground Control, the visible time value is used. In this flyby, the CubeSat is enabling communication during 460.1 seconds at a rate of 512 kbps which means an amount of data sent of about 29 MegaBytes. In addition, as seen in Figure 6.12, the battery consumed due to sending the data is of about 0.9%. All the values are summarised below.

Visible time [s]	**Data sent [MB]**	**Battery consumed [%]**
460.1	29.4464	0.89

Table 6.4: Data transfer summary during TGC flyby.

6.4 Scenario C. Single orbit simulation

This final scenario considers a single orbit of the CubeSat around the Earth. Beginning at 08:00am on September 19th, 2022, the orbit lasts for 1h 34min 30seg according to Eq. (4.2). This specific orbit has been chosen due to doing a flyby over both ground stations and therefore, obtain more significant results.

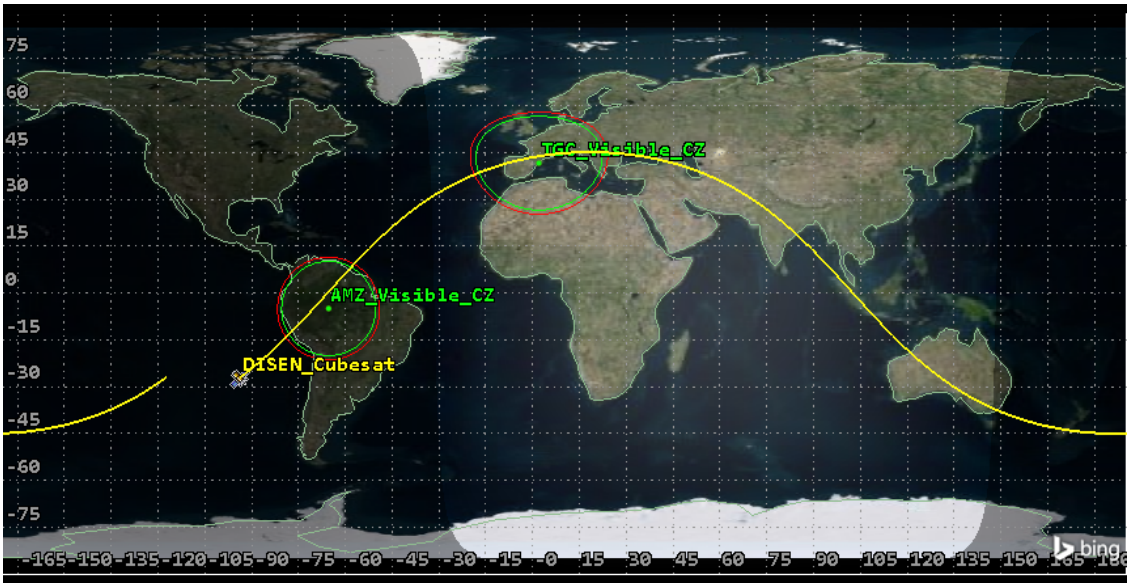


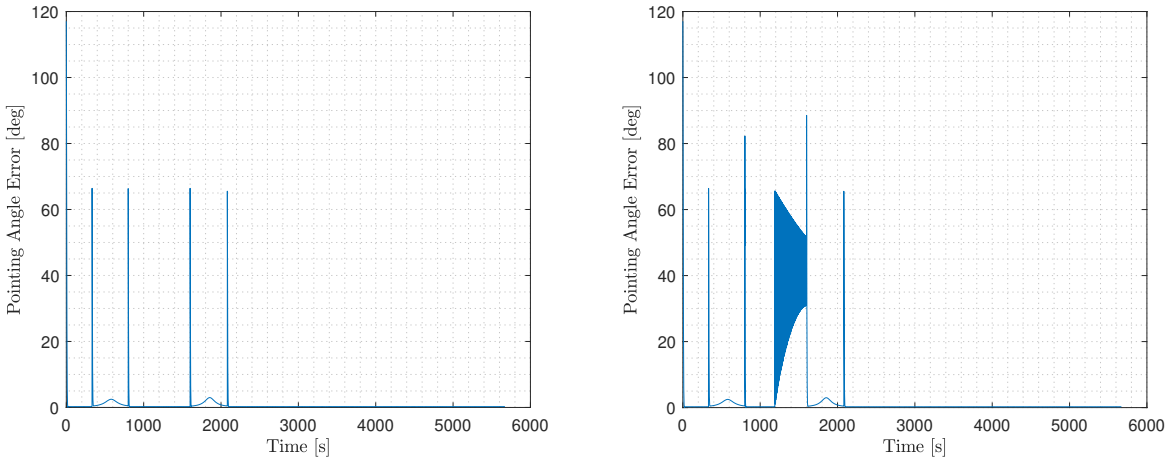
Figure 6.13: Ground track of the CubeSat orbit.

This scenario serves to analyse the effect of exclusion angles values of the star tracker model on the CubeSat behaviour, as well as to assess the battery sizing and obtain the energy margin for the implementation of new components.

6.4.1 Star Tracker. Exclusion angles analysis

The star tracker determine the CubeSat's attitude by comparing the observed star patterns with a star catalog, however, nearby celestial bodies which are significantly brighter than stars can cause confusion and an improper track of the stars when they appear inside the star tracker field of view. Due to that, exclusion angles are defined. These angles represent the angle range inside the field of view where the star tracker avoids the blinding bright bodies.

As specified in Section 4.3.1, the star tracker chosen for the *Amazon Mission* has a Sun exclusion angle of 30 deg and a Moon exclusion angle of 20 deg. It is also pointing in the positive Z body direction. With all that, the pointing accuracy is compared.



(a) Pointing accuracy without exclusion angles.

(b) Pointing accuracy with exclusion angles.

Figure 6.14: Comparison in pointing accuracy over a single orbit for different exclusion angles.

Analysing the Figures above, it is clearly noted where the CubeSat is blinded by a bright object. Using the STK software to look at the relative position of the CubeSat with the Sun and the Moon during the blinding period, it is found that the Moon is the bright body responsible. Due to the inclination of the orbit and the CubeSat's attitude during this specific period, the Moon falls inside the field of view of the star tracker and forces the CubeSat to point away, causing the oscillation depicted in Figure 6.14b. However, as the CubeSat enters the ground station Approximation Zone, the attitude command changes from Earth pointing to ground station pointing and, therefore, the star tracker points at open space and can operate without perturbations. That is the reason behind the sudden stop of the oscillations caused by the moon blinding.

In addition, as the time passes the relative position to the celestial bodies changes and hence, the oscillation periods due to exclusion angles may vary. For this reason, CubeSats which attitude wants to be controlled at every moment, carry multiple sensors for attitude determination (e.g. star trackers plus sun sensors). It is important to avoid these blinding events because components such as the reaction wheels and its motors can deteriorate faster, shortening its lifespan if these oscillations begin to happen in almost each orbit.

6.4.2 Battery life

With the exclusion angles set to 0 deg, the battery consumption of the CubeSat components is computed during a single, worst case considered orbit[9]. This is not yet implemented in the 42 Simulator, however a preliminary study with the idle power consumption of each component is done. In addition, a Raspberry Pi $3B+$ has been considered as the onboard computer of the CubeSat.

[9]For the *Amazon Mission*, a worst case orbit is one with a flyby over both ground stations, which implies a higher consumption due to data transmission and to higher changes in attitude.

Single orbit data

Component	Power [W]	Usage Time	Battery usage
Reaction Wheels	$2 \cdot \lvert T \cdot \omega \rvert$	Full orbit	−0.0056%
Gyroscope	1.5	Full orbit	−16.41%
Star Tracker	0.7	Full orbit	−7.66%
GPS Receiver	0.15	Full orbit	−1.64%
Message Transmission	1	TGC Flyby	−0.89%
Onboard Computer	2 [32]	Full orbit	−21.88%
Solar Arrays	Eq. (5.11)	Full orbit	+68.79%
		Balance	**+20.30%**

Table 6.5: Battery balance over a single orbit.

Summing up, during a single orbit the solar arrays are capable of restoring all the energy consumed by the components plus a 20.3% more. This ensures that even though time passes, the batteries are maintained at full charge and the CubeSat is powered at all times. In order to do a more advanced study in the future, charge and discharge battery models must be implemented to take into account factors such as battery degradation with time.

Chapter 7

Schedule

7.1 Work Breakdown Structure

Below, a visual schematic on how the work has been divided can be seen. Details and dependencies of each task are found in the next section, altogether with a Gantt chart.

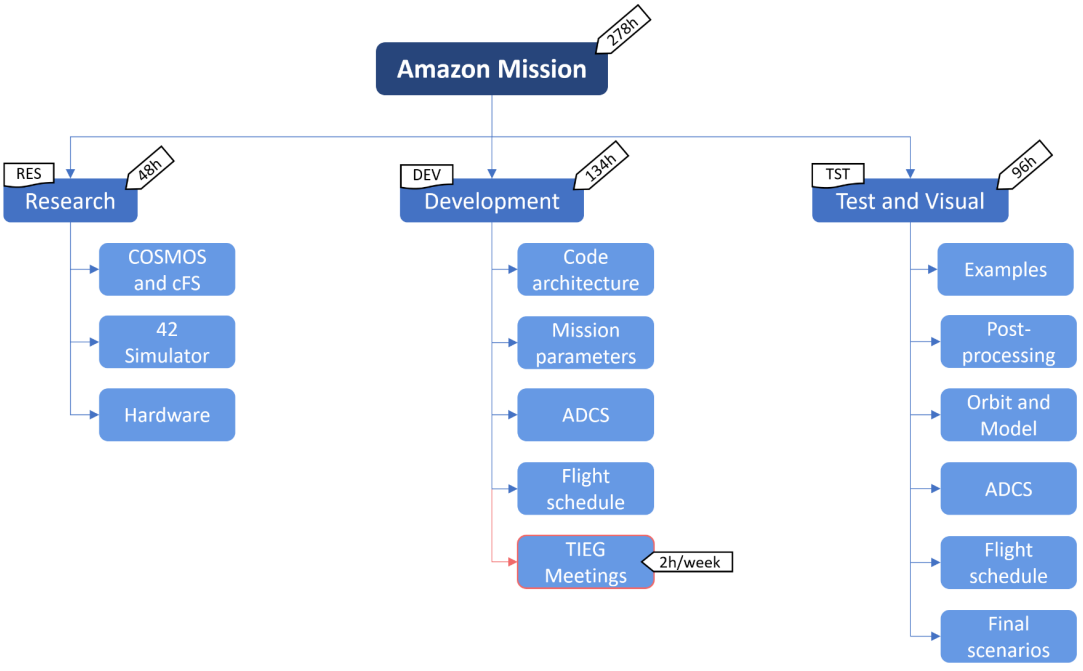


Figure 7.1: Work Breakdown Structure.

7.1.1 Task identification

To organise all the work that needs to be done until the final presentation, four main work packages are defined. Each one has its own specific tasks, so once they are all completed the thesis will be finished. Each task has its own ID which consists of three letters defining the work package and a number.

The first work package defined is the Research and Study (RES), which includes all the information gathering that needs to be done before getting down to work with any software.

- Introduction to COSMOS and cFS platforms (RES-001) [12h]
- Study of 42 Simulator (RES-002) [27h]
- Research on the hardware carried by the CubeSat (RES-003) [9h]

After that, the Design and Development (DEV) work package is defined. This package has the highest amount of working hours since it contains the main work packages of the thesis. In addition, meetings with both director and co-director of the thesis (DISEN Group) have been included.

- Design of the mission characteristics in 42 environment (DEV-001) [15h]
- Design of the code architecture (DEV-002) [15h]
- Development of the attitude control application (DEV-003) [60h]
- Development of the flight schedule, which also includes the BMS and data flow (DEV-004) [45h]
- DISEN Team Meetings, two hours per week during sixteen weeks (DEV-005) [32h]

The third work package to be defined is the Testing and Visualisation (TST). On one hand, the testing part includes every action that will be done to check if the DEV tasks are running properly. And on the other hand, the visualizing part consists on the post-process of the 42 simulation output files.

- Testing of mission examples in 42 (TST-001) [6h]
- Testing of methods of post-processing output files (TST-002) [15h]
- Testing and visualisation of the mission orbit and model and implementation to the final simulation (TST-003) [9h]
- Testing of the attitude control application and implementation to the final simulation (TST-004) [42h]
- Testing of the flight schedule, which also includes the BMS and data flow, and implementation to the final simulation (TST-005) [27h]
- Testing and visualisation of the three final scenarios (TST-006) [15h]

Finally, the Deliverables (DLV) include the parallel work to the previous tasks that must be done to evaluate and present the thesis.

- Project charter (DLV-001) [15h]
- Final report (DLV-002) [60h]
- Presentation slides (DLV-003)

Below, a summary of each work package including task ID, brief description and dependencies with other tasks can be found:

Research and study

ID	Description	Previous
RES-001	Introduction to COSMOS and cFS platforms	-
RES-002	Study of 42 Simulator	-
RES-003	Research on the hardware carried by the CubeSat	-

Table 7.1: RES Work Package summary

Design and Development

ID	Description	Previous
DEV-001	Design of the mission characteristics	RES-003
DEV-002	Design of the code architecture	RES-002, DEV-001
DEV-003	Development of the attitude control application	DEV-002, TST-003
DEV-004	Development of the flight schedule	DEV-003, TST-004
DEV-005	DISEN Team Meetings	-

Table 7.2: DEV Work Package summary.

Testing and Visualisation

ID	Description	Previous
TST-001	Testing of mission examples in 42	RES-002
TST-002	Testing of methods of post-processing output files	RES-002
TST-003	Testing and visualisation of the mission orbit and model	DEV-001
TST-004	Testing of the attitude control application	DEV-002, DEV-003
TST-005	Testing of the flight schedule	DEV-004
TST-006	Testing and visualisation of the final scenarios	TST-005

Table 7.3: TST Work Package summary.

Deliverables

ID	Description	Previous
DLV-001	Project charter	-
DLV-002	Final report	DLV-001
DLV-003	Presentation slides	DLV-002

Table 7.4: DLV Work Package summary.

Adding up all the working hours dedicated to each task, the *Amazon Mission* involves a total of 329 hours, while the development of the whole thesis (*Amazon Mission* plus deliverables) involves a total of 404 hours. This value agrees with the equivalent hours corresponding to 12 ECTS which are 360 hours.

7.1.2 Gantt chart

Regarding the Gantt chart, it can be seen that despite the final report official delivery date being on June 21st, an early end date has been set for June 14th. This is done in order to have enough time to check that everything is properly done and if any mistake or missing part is found, there will be available a whole week to fix it.

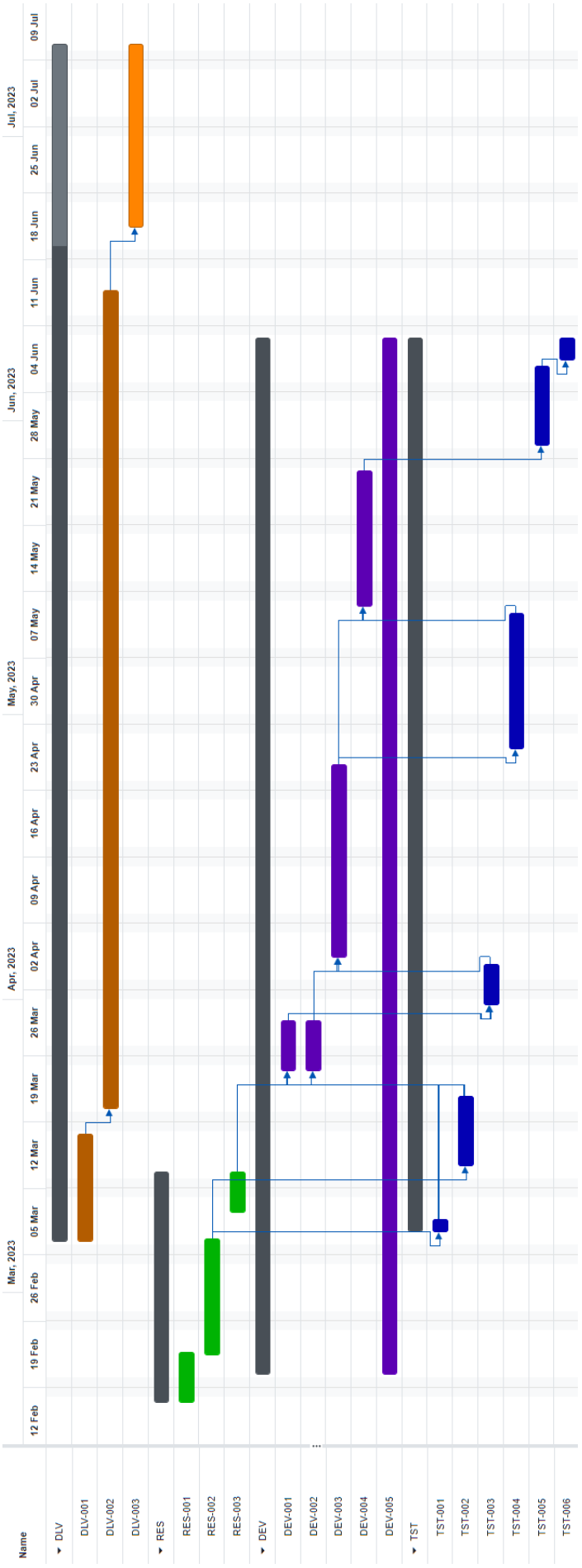


Figure 7.2: Gantt chart.

Chapter 8

Budget Summary

This chapter resumes all the expenses related to the development of this thesis. They are divided in three main groups: human resources, necessary equipment and power consumption. Further detail can be found in the Budget document.

Expense	Total Cost
Human resources	9040€
Hardware	1200€
Power	9.49€
	10249.89€

Table 8.1: Total costs.

It is concluded that this thesis has a total cost of about 10000€, where an 88% is spent on paying salaries and the remaining 12% is invested in the material needed by the engineer.

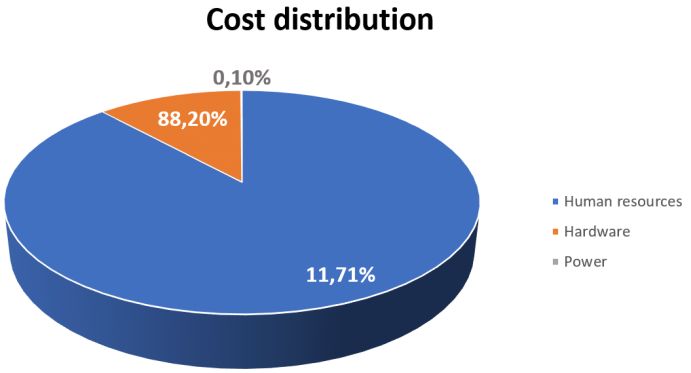


Figure 8.1: Cost distribution.

Chapter 9

Environmental impact

The environmental impact of this thesis is based on the carbon emissions produced by electrical consumption of the computer and the office lamps. Since there is no tasks that involves anything other than computer usage, carbon emissions are quite low.

Concept	Quantity [h]	Power consumption [W]	Carbon emissions [kg CO_2]
Laptop HP Pavilion 15	404	150	16.54
Monitor Samsung Odyssey G3	404	17	1.87
Office lightning	404	8	0.88
		Total	**19.29 kg** CO_2

Table 9.1: Carbon emissions due to development of this thesis.

To obtain the CO_2 kg, the 2022 mean emission factor in Spain has been taken, which is of 0.273 kgCO2/kWh [33].

Chapter 10

Conclusions

Throughout this thesis, each goal set at the beginning has been met successfully as well as all the requirements. Every scope item has been carefully detailed and achieved, resulting in a newer version of the 42 Simulator which can run the *Amazon Mission* CubeSat and fulfill its purpose.

To introduce the 42 Simulator, an entire chapter has been dedicated to help new users reach a solid level of knowledge in short time. In addition, the mission flowchart has been depicted. This will allow to spend less time getting to know the 42 software and the *Amazon Mission* code architecture.

Regarding the hardware, an issue has been encountered with the star tracker. When exposed to bright celestial bodies it is blinded, impeding the CubeSat to point where it should at that specific moment. To solve the issue, a new attitude determination sensor must be added to provide the attitude of the CubeSat when the star tracker is unavailable to do so (e.g. a fine sun sensor or another star tracker).

For the ground station pointing, the reason behind enforcing an accurate response is to maximize the antennas gain. On one hand, in remote areas such as the Amazon, a higher antenna gain can be useful to reduce the power required by the ground antenna, reducing costs and increasing the available spots to place the antenna. On the other hand, a higher antenna gain can provide higher transmission rates as well as higher quality signal, hence more data can be sent with reduced losses.

The fact that the PD control has no optimised gain adjustment procedure, has leaded to the choice of a heuristic procedure. Then, the gains have been obtained through analysing the system response to different values and assessing which performed the best.

Moreover, two preliminary analysis have been done: one for the battery and the other for the data flow. Both give a general idea on how the battery and data transmission performs, while constitute the foundation for future, much more advanced models.

Another interesting result that has been obtained is the command refresh rate value. The CubeSat response has been increasingly better as the refresh rate decreased, until being limited by the star tracker output rate value. Reaching this value, means the system can stabilise a setpoint that changes in each iteration. However,

the refresh rate may vary if the hardware is changed.

As for the performance, the CubeSat is able to point at ground stations with an accuracy lower than 4 degrees over an entire flyby, which allows an efficient communication between parabolic antennas. Additionally, it can send and receive up to 30 MB of data during a flyby and restore all the energy consumed with a 20% surplus.

To conclude, the final simulations results confirm that the CubeSat performs as intended, meeting all the goals set at the beginning of the thesis.

10.1 Future work

In order to achieve a much more realistic approach of the *Amazon Mission*, some aspects that has been identified during the development of this thesis, can be improved. With that, another step will be made towards the HITL simulation.

- The inertia matrix of the CubeSat should be updated with more precise values once the CubeSat is fully assembled.
- CubeSat sensors should be carefully analysed and selected to suit perfectly this case.
- The star tracker should be complemented with at least, one more attitude determination sensor. The second sensor should only operate when the star tracker is blinded.
- Models of actuators and sensors in the 42 Simulator should be analysed looking for improvement areas. This way, models can provide more realistic results.
- Orbital perturbations should be studied and considered enabling them to run a more realistic simulation.
- The Battery Management System should be improved by adding charging and discharging models and be able to determine the battery degradation over time.
- A link budget for the communication procedures should be developed to obtain accurate results on data and antenna power consumption.
- The command refresh rate should be updated in case any sensor has a lower frequency.
- An assessment of the battery sizing should be done to decide whether to add or remove components.

Bibliography

1. GROUP, IMARC. CubeSat Market: Global Industry Trends, Share, Size, Growth, Opportunity and Forecast 2022-2027. *Research and Markets*. 2022.
2. SLO, Cal Poly. CubeSat Design Specification. *The CubeSat Program*. 2022.
3. NANOSATS. *World's largest database of nanosatellites and CubeSats*. 20223. Available also from: https://www.nanosats.eu/. Retrieved 20th April, 2023.
4. WU, Shufan; ZHAO, Tiancheng; GAO, Yuan; CHENG, Xiao. Design and implementation of a Cube satellite mission for Antarctic glacier and sea ice observation. *Acta Astronautica*. 2017, vol. 139, pp. 313–320.
5. NANOSATS. *ALEASAT*. 2023. Available also from: https://www.nanosats.eu/sat/aleasat. Retrieved 25th April, 2023.
6. BURVILLE, Kevin; MEZA, Julian Mentasti; TAJWAR, Zavian Noah; DIVSALAR, Donya Naz; HOLMES, Megan. An Earth Imaging CubeSat as a Disaster Relief Tool for Amateur Radio Operators. In: Dubai, United arab emirates, 2021, vol. B1.
7. SAT, SFU. *ALEASAT*. 2022. Available also from: https://www.sfusat.org/alea. Retrieved 25th April, 2023.
8. ANSYS. *Ansys STK*. 2023. Available also from: https://www.ansys.com/products/missions/ansys-stk#tab1-1. Retrieved 25th April, 2023.
9. SOLUTIONS, a.i. *FreeFlyer Astrodynamics Software*. 2023. Available also from: https://ai-solutions.com/freeflyer-astrodynamic-software/. Retrieved 27th April, 2023.
10. NASA. *OpenSatKit*. 2020. Available also from: https://github.com/OpenSatKit/OpenSatKit. Retrieved 1st May, 2023.
11. OPENC3. *Cosmos*. 2023. Available also from: https://github.com/OpenC3/cosmos. Retrieved 1st May, 2023.
12. NASA. OpenSatKit Quick Start. 2020.
13. NASA. Core Flight System (cFS) Training. 2020.
14. VITTALDEV, Vivek. The Unified State Model. Derivation and Applications in Astrodynamics and Navigation. *TU Delft Master Thesis*. 2010.
15. REBOUND. *Orbital elements*. 2022. Available also from: https://rebound.readthedocs.io/en/latest/orbitalelements/. Retrieved 8th May, 2023.

16. SATÉLITES, Sondas y. *¿Como se construye un satelite artificial?* 2016. Available also from: http://sondasysatelites.blogspot.com/2006/09/sistema-de-determinacin-y-control-de.html. Retrieved 8th May, 2023.
17. MATHWORLD, Wolfram. *Euler Angles*. 2023. Available also from: https://mathworld.wolfram.com/EulerAngles.html. Retrieved 7th May, 2023.
18. GUERRERO-SANCHEZ, Maria; ABAUNZA, Hernan; CASTILLO, Pedro; LOZANO, Rogelio; GARCIA-BELTRAN, Carlos; RODRIGUEZ-PALACIOS, Alejandro. Passivity-based control for a micro air vehicle using unit quaternions. *Appl. Sci. (Basel)*. 2016, vol. 7, no. 1.
19. LABSAT. *GPS Time Calculator*. 2022. Available also from: https://www.labsat.co.uk/index.php/en/gps-time-calculator. Retrieved 15th April, 2023.
20. STONEKING, Eric. An Introduction to Simulation Using 42. *NASA*. 2017.
21. MANZANARES, Mar Bonet. Proyecto de simulador de comunicaciones opticas entre satelites LEO, GEO y Tierra. *UPC Master Thesis*. 2022.
22. DURAN, Ariana Miño. Design and Manufacture of 3DOF reaction wheels as actuators for Attitude Control of a 1U CubeSat. *UPC Master Thesis*. 2022.
23. SATSEARCH. *STIM202 3-axis Gyro Module. Safran*. 2021. Available also from: https://satsearch.co/products/sensonor-stim202-3-axis-gyro-module. Retrieved 28th May, 2023.
24. SATSEARCH. *Star tracker. KAIROSPACE Co., Ltd.* 2020. Available also from: https://satsearch.co/products/kairospace-star-tracker. Retrieved 28th May, 2023.
25. SATSEARCH. *GPS Receiver. WarpSpace*. 2016. Available also from: https://satsearch.co/products/warpspace-gps-receiver. Retrieved 28th May, 2023.
26. DIAZ, Xavier Pozo. Study of attenuation and loss of messages in radiofrequency communication links between Cubesats and Earth. *UPC Degree Thesis*. 2022.
27. Signal Characteristics and Information Extraction. In: *Global Positioning Systems, Inertial Navigation, and Integration*. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2008, pp. 53–110.
28. NASSIFF, Nell Elizabeth; VELEZ, Dianna M; DAWSON, Elizabeth I. Attitude Determination and Control Subsystem Design for a CubeSat. *Worcester Polytechnic Institute*. 2012.
29. ELMAS, Tuba Çigdem. Development of control allocation methods for satellite attitude control. *Middle East Technical University Thesis*. 2010.
30. CHEN, Weiyue; JING, Wuxing; LI, Chaoyong. Attitude recovery for microsatellite via magnetic torque. In: *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*. 2009, pp. 2516–2521. Available from DOI: 10.1109/CDC.2009.5399878.
31. WISSEN, IT. *solar constant*. 2019. Available also from: https://www.itwissen.info/en/solar-constant-123132.html#gsc.tab=0. Retrieved 4th June, 2023.
32. DRAMBLE, Raspberry Pi. *Power Consumption Benchmarks*. 2023. Available also from: https://www.pidramble.com/wiki/benchmarks/power-consumption. Retrieved 10th June, 2023.
33. GENCAT. *Factor de emisión de la energía eléctrica*. 2023. Available also from: https://canviclimatic.gencat.cat/es/actua/factors_demissio_associats_a_lenergia. Retrieved 17th May, 2023.