NANOSAT LAB

TRABAJO FINAL DE GRADO

TFG TITLE: "Implementation and validation of the Attitude and Determination Control System of a pico-satellite"

DEGREE: Degree in Aerospace Systems Engineering (major in Air Navigation)

AUTHOR: Yago Recarey Liste

ADVISORS: Park, Hyuk

Adriano Camps

DATE: September, 2023

TITULO TFG: “Implementation and validation of the Attitude and Determination Control System of a pico-satellite”

GRADO: Grado en Ingeniería de Sistemas Aeroespaciales (mención en Aeronavegación)

AUTOR: Yago Recarey Liste

DIRECTOR: Park, Hyuk

CODIRECTOR: Adriano Camps

FECHA: Septiembre, 2023

Resumen

El presente trabajo de fin de grado se enfoca en el desarrollo del software de un Sistema de Determinación y Control de Actitud de un pico-satélite, haciendo uso de MATLAB, así como su posterior integración en C. El software realizado, será implementado en un pico-satélite de 5 cm de lado, denominado PocketQube.

El proyecto de investigación formará parte de la misión IEEE Open PocketQube Kit desarrollada en el laboratorio NanoSat perteneciente a la UPC.

El sistema de actitud del pico-satélite se puede desglosar en dos ramas bien diferenciadas, la determinación de la actitud y el control de la actitud. La determinación de la actitud implica el proceso de determinar la orientación del satélite utilizando las mediciones adquiridas por sus sensores (incluyen fotodiodos, giroscopios y magnetómetros). Por otro lado, el control de la actitud es el proceso mediante el cual se controla la orientación del pico-satélite utilizando algoritmos de control específicos (como detumbling y nadir pointing) y actuadores, (como los magnetorquers). Para analizar y comprender la actitud del pico-satélite, se desarrolla un modelo que permite simular el entorno en el que se encuentra el satélite. Este modelo abarca la dinámica del pico-satélite, un modelo que describe la dinámica orbital y otro que representa las diversas fuerzas de perturbación que afectan al pico-satélite. De esta manera, se logra describir la órbita seguida por el pico-satélite, incluyendo su posición, velocidad y las fuerzas externas que influyen y afectan en su comportamiento durante el vuelo orbital.

Una vez completada la modelización del entorno de simulación, se procede a la implementación de los algoritmos de determinación y control, así como de los modelos matemáticos que describen el comportamiento de los sensores y actuadores. Estos algoritmos y modelos son diseñados con el objetivo de cumplir con los requisitos establecidos para el Sistema de Determinación y Control de Actitud. Posteriormente, se lleva a cabo un análisis exhaustivo de los resultados obtenidos durante la simulación. Este análisis tiene como finalidad verificar si se cumplen los requisitos previamente establecidos para el Sistema de Determinación y Control de Actitud.

La conclusión de este trabajo permite empezar las campañas de test de este sistema en el hardware del PocketQube, así como comprobar si los resultados obtenidos en las simulaciones corresponden con la realidad.

TFG TITLE: Implementation and validation of the Attitude and Determination Control System of a pico-satellite

DEGREE: Degree in Aerospace Systems Engineering (major in Air Navigation)

AUTHOR: Yago Recarey Liste

DIRECTOR: Hyuk Park

CO-DIRECTOR: Adriano Camps

DATE: September, 2023

Overview

This end-of-degree project focuses on the software development of a pico-satellite Attitude Determination and Control System, using MATLAB, as well as its subsequent integration in C. The software carried out will be implemented in a 5 cm sided pico-satellite, called PocketQube.

The research project will form part of the IEEE Open PocketQube Kit mission developed in the NanoSat laboratory belonging to the UPC.

The pico-satellite attitude system can be broken down into two distinct branches, attitude determination and attitude control. Attitude determination involves the process of determining the orientation of the satellite using the measurements acquired by its sensors (they include photodiodes, gyroscopes, and magnetometers). On the other hand, attitude control is the process by which the orientation of the pico-satellite is controlled using specific control algorithms (such as detumbling and nadir pointing) and actuators, (such as magnetorquers). To analyse and understand the attitude of the pico-satellite, a model is developed that allows simulating the environment in which the satellite is located. This model encompasses pico-satellite dynamics, a model that describes orbital dynamics, and another that represents the various perturbation forces that affect the pico-satellite. In this way, it is possible to describe the orbit followed by the pico-satellite, including its position, speed and the external forces that influence and affect its behaviour during the orbital flight.

Once the modelling of the simulation environment is complete, the determination and control algorithms are implemented, as well as the mathematical models that describe the behaviour of the sensors and actuators. These algorithms and models are designed with the aim of meeting the requirements established for the Determination and Control System. Subsequently, an exhaustive analysis of the results obtained during the simulation is carried out. The purpose of this analysis is to verify if the previously established requirements for the Attitude Determination and Control System are met.

The conclusion of this work allows to start the test campaigns of this system in the PocketQube hardware, as well as to check if the results obtained in the simulations correspond to reality.

Agradecimientos

Finalmente, llego a la meta después de estos 4 años de estudios, esfuerzo y tiempo invertido, que, sin duda, han merecido la pena para alcanzar mi objetivo.

A mi lado en este camino, han estado diversas personas a las cuales quiero dedicar este trabajo de final de grado, y a su vez expresar mi agradecimiento por todo su cariño y apoyo en esta etapa de mi vida.

En primer lugar, a mis padres, por su motivación constante y apoyo incondicional en todos los aspectos, permitiéndome siempre realizar y alcanzar mi objetivo, a pesar del esfuerzo, que ha supuesto estos años tanto para mí, como para ellos; así como a mis abuelos, tíos y primos.

Gracias por vuestro apoyo.

A mis amigos, los de siempre, que seguimos acompañándonos año tras año aun cuando la distancia nos tiene largas temporadas sin vernos; y a los nuevos que he tenido el placer de conocer durante esta nueva etapa universitaria, a todos, gracias por estar siempre ahí.

Por último, pero no menos importante, me gustaría agradecer a mis tutores y compañeros involucrados en este proyecto. En primer lugar, a Hyuk, quien amablemente accedió a ser mi director a pesar de su apretada agenda y horarios. A continuación, también considero fundamental agradecer el inestimable apoyo de todos mis compañeros del laboratorio NanoSat, que han estado involucrados desde el inicio de nuestra relación y a los cuales he recurrido en diversas ocasiones. En particular, quisiera extender una mención especial a mis dos compañeros del departamento de ADCS, Youssef El Kaisi Rahmoun y José Faneca Ruedas, quienes trabajaron codo con codo conmigo durante estos cuatro meses, y como no, a Luis siempre dispuesto a atenderme y ayudar en todo lo que estuviera en su mano. Gracias a todos los integrantes del laboratorio por toda la ayuda, apoyo y conocimientos que me han brindado durante estos meses de investigación, donde me han enseñado sobre todo que no debo rendirme, escuchado y he aprendido mucho de toda su experiencia.

A todos, muchas gracias por formar parte de esta etapa de mi vida.

Yago Recarey Liste

Barcelona 2023

Special Thanks

Finally, I reach the goal after these 4 years of studies, effort and time invested, which, without a doubt, have been worth it to achieve my goal.

By my side on this path, there have been various people to whom I want to dedicate this final degree project, and in turn express my gratitude for all their love and support at this stage of my life.

First of all, to my parents, for their constant motivation and unconditional support in all aspects, always allowing me to carry out and reach my goal, despite the effort that these years have meant for both me and them; as well as my grandparents, uncles and cousins.

Thanks for your support.

To my friends, the usual ones, who continue to accompany us year after year even when the distance keeps us from seeing each other for long periods of time; and to the new ones that I have had the pleasure of meeting during this new university stage, to all of you, thank you for always being there.

Last but not least, I would like to thank my tutors and colleagues involved in this project. First of all, to Hyuk, who kindly agreed to be my director despite his busy schedule and schedule. Next, I also consider it fundamental to thank the invaluable support of all my colleagues in the NanoSat laboratory, who have been involved since the beginning of our relationship and to whom I have turned on various occasions. In particular, I would like to extend a special mention to my two colleagues from the ADCS department, Youssef El Kaisi Rahmoun and José Faneca Ruedas, who worked closely with me during these four months, and of course, to Luis, who is always willing to assist me and help in everything that was in his hand. Thanks to all the members of the laboratory for all the help, support and knowledge that they have given me during these months of research, where they have taught me above all that I should not give up, listened and I have learned a lot from all their experience.

To all, thank you very much for being part of this stage of my life.

Yago Recarey Liste

Barcelona 2023

INDEX

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

ADCS	Attitude Determination and Control System.
AWGN	Additive White Gaussian Noise
CMoS	Control Margin of Safety.
CoM	Centre of Mass.
CuPID	CubeSat PocketQube Integrated Deployer.
D	Detumbling.
ECI	Earth-Centred Inertial Frame
EOP	Early Orbital Operations.
ESA	European Space Agency.
GPS	Global Positioning System.
GSP	Simplified General Perturbation
IEEE	Institute of Electrical and Electronics Engineers.
IGRF	International Geomagnetic Reference Field.
IMF	Interplanetary magnetic field
LEO	Low Earth Orbits.
LVLH	Local-Vertical-Local-Horizontal.
MOI	Moment of Inertia
MRFOD	Morehead Rome Femto Orbital Deployer
MSU	Morehead State University
NP	Nadir Pointing.
OBC	On Board Computer.
SGP	Simplified General Perturbations.
SRP	Solar Radiation Pressure.
TLE	Two-Line Element.
UPC	Polytechnic University of Catalonia.
UPC	NanoSat Lab UPC Nano-Satellite and Payload Laboratory.
VHF	Very High Frequency.

CHAPTER 1. INTRODUCTION

1.1 UPC NanoSat Lab

The Nano-Satellite and Payload Laboratory (UPC NanoSat LAB) [1], located in the Technical University of Catalonia, is an interdepartmental initiative belonging to the Barcelona School of Telecommunications Engineering. The main activity of the lab is the design and development of nano-satellite missions focusing on the exploration of innovative small spacecraft system concepts and developing and integrating subsystems and payloads for Earth Observation.

Currently, the UPC is developing different research projects with the aim of carrying out 3 launches in the year 2023. In this final degree project, we will focus on the IEEE Open PocketQube Kit, which is the mission in which I am involved.

1.2. Open PocketQube Kit

The “IEEE Open-PocketQube Kit” is a multidisciplinary IEEE Geoscience and Remote Sensing Society [2] initiative. This kit is based on the PocketQube standard [3], which defines a small satellite of just 5 cm x 5 cm x 5 cm, and allows for the creation of highly capable and cost-effective satellites.

The IEEE Open-PocketQube Kit provides a complete solution for building PocketQube satellites, including a standardised interface, a set of open-source design files and software tools. The main objective of this project is to develop different educational PocketQubes for space research and to facilitate access to the space environment for institutions and organisations that cannot afford the cost of developing a satellite.

The laboratory has started designing the PocketQubes in the spring of 2020 and nowadays, 3 PocketQubes have been designed and implemented. Mainly, the participants in these projects are UPC students who are already finishing their studies helped by graduates who have decided to continue their studies by doing research at the laboratory.

Within the “IEEE Open PocketQube Kit” project three different payloads will be developed and tested in three different PocketQubes:

PoCat-1: carries a VGA camera,

PoCat-2: carries a L-band RFI monitoring payload,

PoCat-3: carries a Ka-band RFI monitoring payload to measure the possible interferences caused by the 5G communications system deployment.

This work focuses on the last two payloads.

Mission Objectives and Requirements

It is essential to define the objectives and requirements. These serve as the foundation on which the mission is built and guide the design, development, and implementation of mission components and systems. In the picosatellite context, the objectives can be categorised into general and specific objectives for each subsystem. The general objectives describe the general goals of the mission, while the specific objectives define the more detailed goals for each area of the picosatellite.

To provide a clear and concise overview of these objectives, they have been presented in a table format. This table not only describes the general and specific objectives of the mission, but also serves as a reference point throughout the development of the mission. By clearly defining these objectives, the project team can ensure that all efforts are aligned with the ultimate mission goals and objectives:

Table 1.1. Mission Objectives and requirements

Index	Domain	Description
M-0010	General	The satellite will be launched in a LEO orbit corresponding to both its purpose and the international regulations.
M-0020	General	The satellite must not contain entry resistant materials or completely detachable sections or appendages.
M-0030	General	The satellite will have a 2.7 lifespan until passivization.
M-0040	General	The satellite must be able to control its attitude in case of payload acquisition requirements as well as high temperatures.
M-0100	Structural	The satellite must abide toboth its class' available standard, as well as the regulations imposed by the deployer entity.
M-0110	Structural	The satellite must have all its deployable elements safely stowed during transport, storage and launch.
M-0120	Structural	The satellite must be capable of of starting and using all its systems and appendices in a controlled manner.
M-0210	Electronic	The satellite must keep its power source disconnected from all its subsystems during transport, storage and launch.
M-0220	Electronic	The satellite must be capable of satisfactory harvesting all the necessary power during its entire lifespan.
M-0230	Electronic	The satellite must regulate all its power and data lines, providing protection against potential electrical hazards.
M-0300	Computational	The satellite must maintain complete control of its circuitry, data processes, communications, payloads and physical interfaces at all times.
M-0310	Computational	The satellite must be able to store both the information obtained from payloads as well as telemetry: location, date, battery level, temperature, current and voltage in key nodes.
M-0400	Communications	The satellite must be able to communicate and receive telecommands from both the ground station and other satellites.
M-0410	Communications	The satellite must be able to maintain a satisfactory link budget and to allow control of the satellite by way of telecommands.
M-0420	Communications	The satellite must be able to transmit and receive without an attitude requirement.

1.3. Scope

The aim of this project is to develop an Attitude and Determination Control System Software and achieve the goals of the ADCS. For this purpose, a complete study of the spacecraft configuration, the environmental model, the disturbances that affected the satellite and finally the determination and control algorithms is done. The development of the simulation software is done with MATLAB, thanks to the Princeton CubeSat Toolbox [4] which will be modified in order to be operational for the PocketQube.

In the first part of the work, the main objective is to understand how to create a good environmental model in order to obtain the most realistic results in our simulations. For this, we would stand the frames in which we will work and we would study the different propagation algorithms in order to simulate the orbit of the picosatellite, the geomagnetic field that affects the satellite, the sun position which would be essential for the determination and the effects of the atmosphere in a LEO orbit.

Secondly, we would learn the different external disturbances that affect the satellite in the space. For example, in this work we would study the aerodynamic, gravity and optical force and we would see their effects on the pico-satellite.

Then, we would study the different algorithms that are used to determine and control the picosatellites and that are essential to achieve the goals of the mission. In this work the main goals are to be able to control the rotation of the picosatellite due to the implementation of a detumbling algorithm and to be able to point to a specific location due to the nadir pointing algorithm. Moreover, a study of how to create a determination algorithm it will be done.

Finally, once the development of the algorithm is done, it would be implemented in C in order to be operational with the other subsystems of the PocketQube.

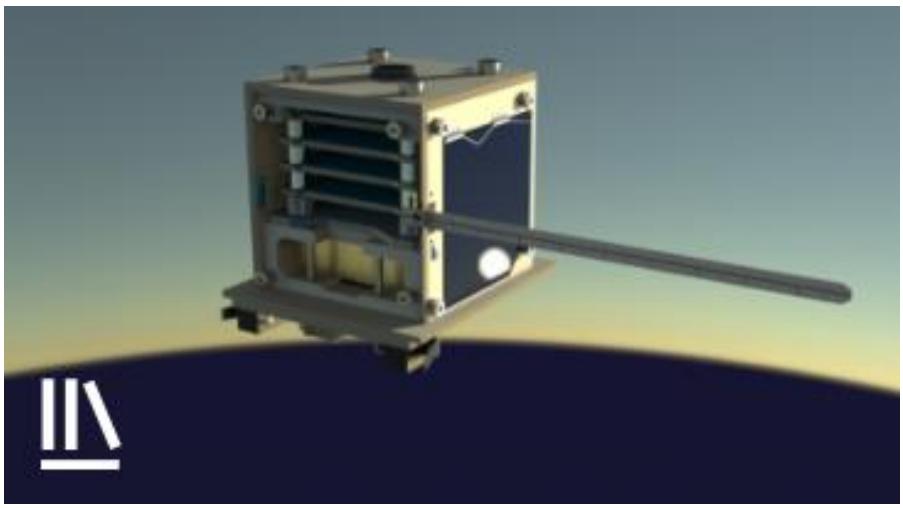

Figure 1.1 Open PocketQube Kit mission

CHAPTER 2. STATE OF ART

2.1. PocketQubes Background

The PocketQube concept was first proposed in 2009 by Professor Robert J. Twiggs, as a result of a collaboration between Morehead State University (MSU) and Kentucky Space which developed some specifications regarding this new class of spacecraft. The PocketQube Standard [3] proposed a picosatellite with dimensions 50x50x50 mm called a 1P unit which is half the size of a CubeSat, with a mass not exceeding 250 g. The first PocketQube was launched using the Morehead Rome Femto Orbital Deployer (MRFOD) installed inside the UniSat5 microsatellite by MSU in collaboration with Kentucky Space. The main objective of the PocketQube was to further reduce the spacecraft cost and complexity, while trying to maintain as much functionality as possible. This has been possible thanks to the improvement in commercially off-the-shelf (COTS) components. However, some disadvantages have raised. For example, the lifespan of a PocketQube mission is typically less than 2 years. In addition, by reducing the size and cost of the spacecraft, its liability is affected and its power consumption is reduced seeing that the batteries cannot take as much space and the number of usable solar cells is very limited. Consequently, starting to develop a PocketQube still requires some knowledge on which parts need to be kept for the success of the mission, and how the spacecraft needs to be optimised to reduce its requirements.

Nonetheless, this new type of spacecraft has become very popular as an educational tool for academic institutions and associations. The two leading companies in the use of PocketQubes for space missions have been Alba Orbital [5] and FOSSA Systems [6]. For the project, the PocketQube 1P is used as a reference. The following Figure shows the PocketQube external dimensions and its sliding backplate dimensions:

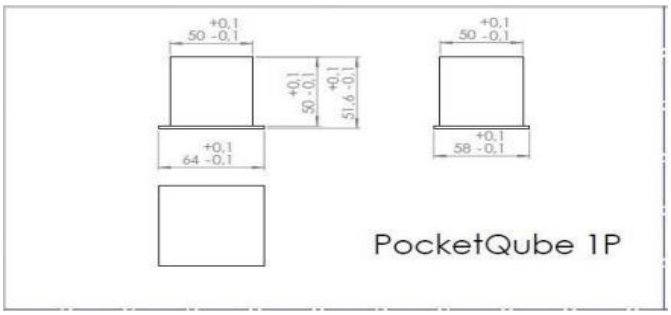

Figure 2.1 PocketQube 1P external dimensions and sliding backplate dimensions

2.2. Attitude Determination and Control System

The Attitude Determination and Control System (ADCS) is the spacecraft subsystem whose main responsibility is to determine the spacecraft's orientation and position, and control it to achieve a desired orientation. These main functionalities can be separate in two different groups:

Attitude Determination: It is a critical component of spacecraft operation, responsible for orienting the satellite precisely in a specific location. In the context of our specific mission, the camera serves as the payload and requires accurate positioning to capture images of targeted locations on Earth. The performance of this function relies on the data extracted from the sensors. The data collected is then processed and transmitted to the ground station, where we can take appropriate actions based on the measurements obtained. The successful execution of this process is essential for ensuring that the satellite remains accurately oriented and pointing in the right direction, allowing us to obtain high-quality images of Earth's surface.

Control the spinning of the satellite: It is a function in charge of positioning the spacecraft and adjusting its orientation to point towards specific locations signalled by the ground station. This process is achieved through the use of actuators, such as magnetorquers, which can change the satellite's rotation and stop it if necessary.

The ADCS is divided into 4 functional blocks which are the following:

Sensors Block: This block consists of a series of sensors with the aim of collecting as much data as possible from space to fulfil ADCS duties. The sensors used in the PocketQube are the photodiodes, magnetometers and gyroscopes. The photodiodes estimate the sun vector with respect to the PocketQube depending on the amount of energy it receives from the sun. The magnetometers measure the magnetic field experienced by the satellite. Finally, the gyroscopes measure the angular velocity of the satellite.

Processing Data: The goal of data processing is, once we have collected data from the sensors, applying a series of algorithms, to obtain the actions necessary to perform the required tasks. It is also responsible for processing and converting the data obtained from the sensors into valuable results that can be later sent to the ground station.

Actuators Block: The Actuators Block consists of a set of magnetorquers that are responsible for orienting the satellite towards the location specified by the ground station. Typically, these magnetorquers consist of electromagnetic coils that, when a current is applied, generate a magnetic dipole. This dipole interacts with the Earth's magnetic field, producing a torque that causes the satellite to rotate.

Interface Block: This module is in charge of sending and receiving the data obtained from the sensors to the OBC and COMMS subsystems.

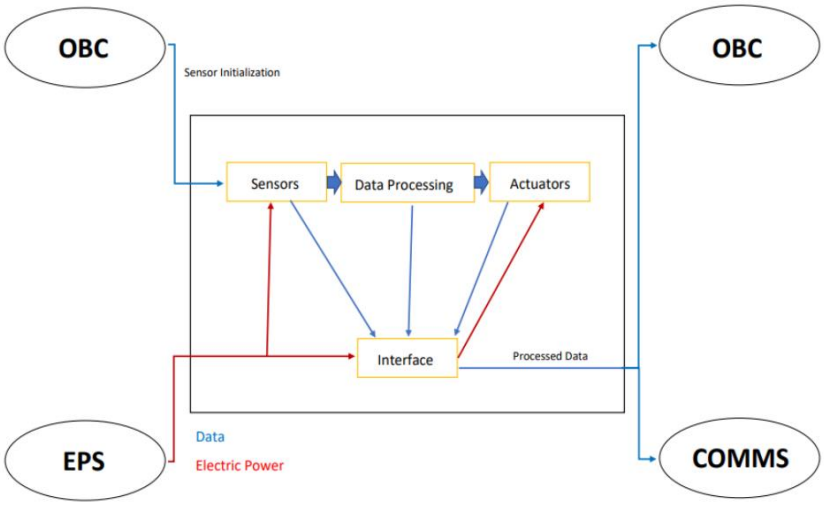

Figure 2.2 Schematic of the ADCS

Figure 2.2 illustrates the two primary sections of the subsystem: the electrical component and the coding component. The coding aspect is represented by the blue lines, while the red lines correspond to the electrical section.

Finally, all these blocks are implemented on the ADCS and Breakout Board which is represented in the following Figures.

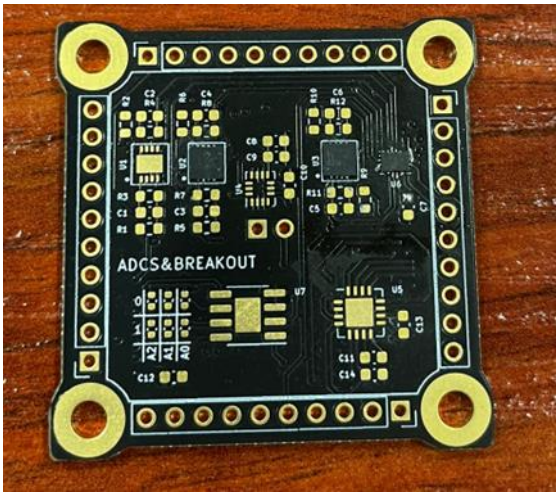

Figure 2.3 ADCS and Breakout board, top side

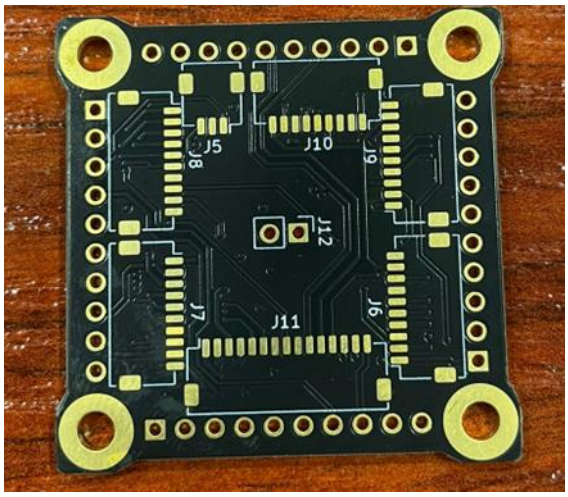

Figure 2.4 ADCS and Breakout board, bottom side

2.3. ADCS requirements

The subsystem requirements established for the mission are presented below.

Table 2.1 ADCS system requirements

ADCS-0010	Be able to extract data from the photodiode.
ADCS-0020	Be able to recollect data from the magnetometer.
ADCS-0030	Be able to control the rotation of the satellite.
ADCS-0040	Be able to recollect data from the gyroscope.
ADCS-0050	Be able to point at a specific location using the nadir algorithm.
ADCS-0060	Be able to know the location of the satellite using the SGP
ADCS-0070	Be able to communicate correctly with the OBC so that the satellite can control its rotation.

2.4. ADCS Components Specifications

Table 2.2. ADCS components Specifications

Name	Model	Supply Voltage [V]	Dissipated Power	Size [mm]
Gyroscope	IIM-42652	1.71 - 3.6	0.3 µW	2.5 x 3 x 0.91
Magnetometer	MMC5983MA	2.8 - 3.6	30 µW	3 x 3 x 1
I2C Mux.	TCA9548ARGER	1.65 - 5	450 mW	7.8 x 4.4 x 1
Op. Ampl. (1x4)	LT6005CDHC#TRPBF	1.6 - 16	26.4 µW	5 x 3 x 1
Op. Ampl. (1x2)	LT6004IMS8#PBF	1.6 - 16	13.2 mW	3 x 3 x 1
Or	SN74AUP1G08DBVT	0.8 - 3.6	12 mW	2.9 x 1.6 x 1.1
Driver mux.	TMUX1204DGSR	1.08 - 5.5	2.25 µW	3 x 3 x 1.1
Current Driver	BD2606MVV	2.7 - 5.5	780 mW	4 x 4 x 1
Photodiode	SLCD-61N8	-	-	3.4 x 1.3 x 0.4

CHAPTER 3. ADCS SIMULATOR DEVELOPMENT

3.1 Spacecraft Configuration

First of all, it is necessary to understand the physical characteristics of the PocketQube, as they serve as the fundamental basis for every simulation we may undertake. In particular, we will focus on its geometry, mass properties (including mass, centre of mass, and Moment of Inertia (MOI) tensor), as well as the optical properties of the materials and surface components. In this work, for every simulation we would consider that the PocketQube is in Stowed Configuration.

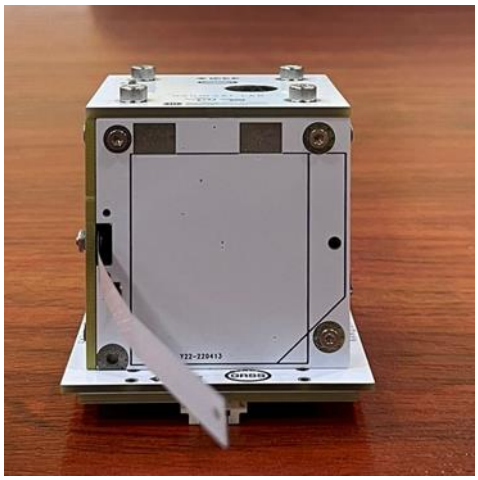

Figure 3.1 PocketQube in Stowed Configuration

For the sake of simplicity, the following assumptions are made for the calculations:

We define the spacecraft as a solid rigid, that means, the spacecraft maintains a constant shape and does not deform when subjected to external forces.

The centre of mass is equal to the solid's geometric centre (uniform mass distribution).

Homogeneous consideration of the surface materials.

The stowed configuration of the PocketQube refers to its folded or compressed state. As we said before, for all the simulation processes we use this configuration

but it is important to mention that in PoCat-2 there is another configuration state that is called deployed configuration in which an antenna is deployed once the storage, launch and detumbling phase have been completed.

The dimensions of a 1U PocketQube are 50 x 50 x 50 mm as we have mentioned in the State of Art.

The mass considered for a 1U PocketQube is 250 g. As we assume that the centre of mass coincides with the geometric centre of the solid, then its location can be expressed as the intersection of the diagonals of the cube at coordinates (0, 0, 0).

Since the spacecraft definition has been done using a simple and well-known geometry, the definition of its inertia tensor with respect to its centre of mass in kg/m^2 comes directly from the following equation where M represents the total mass and L the side length:

$$\mathrm{I_{stowed}} = \begin{pmatrix} \mathrm{M*L\hat{}2} & 0 & 0 \\ 0 & \mathrm{M*L\hat{}2} & 0 \\ 0 & 0 & \mathrm{M*L\hat{}2} \end{pmatrix} = \begin{pmatrix} 1.0417\mathrm{e}-4 & 0 & 0 \\ 0 & 1.0417\mathrm{e}-4 & 0 \\ 0 & 0 & 1.0417\mathrm{e}-4 \end{pmatrix} \quad (\mathbf{3.1})$$

3.2. Environment Model

3.2.1 Coordinate Frames

The definition of the coordinate systems is one of the most important parts for the determination and control of a satellite seeing that for all the calculations it should be defined the reference system being used. The coordinate systems used in the work are the Earth-Centred Inertial Frame (ECI), the Orbital axes (x0, y0, z0) and the Spacecraft body frame. In order to explain this subsection, the following astrodynamics book is used [7].

3.2.1.1. Earth-Centred Inertial Frame (ECI)

The Earth-Centred Inertial Frame (ECI) is a coordinate system used in aeronautical applications to describe the position and motion of objects relative to the Earth. In this system, the Earth is considered to be stationary and the coordinates are fixed to the centre of the Earth. The ECI system is an inertial frame of reference, which means that it is a non-accelerating frame of reference. This allows for accurate calculations of the motion of objects in space without the effects of the Earth's rotation and orbital motion.

The coordinates in the ECI system are typically given in terms of the Cartesian coordinate system, where the X-axis points towards the vernal equinox, the Y-axis points in the direction of the Earth's rotation, and the Z-axis points towards the North Pole.

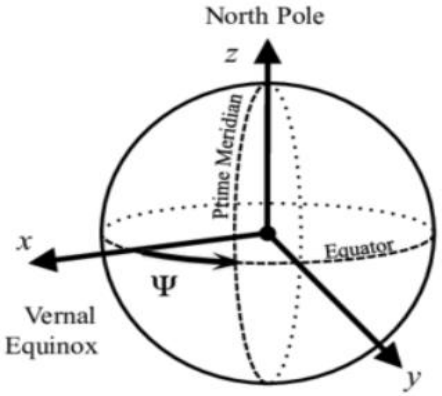

Figure 3.2 ECI frame

3.2.1.2. Orbital Frame

The Orbital System is a non-inertial frame. It is also known as the Local Vertical Local Horizontal (LVLH). This system has its origin in the centre of the spacecraft and it rotates with the spacecraft as it orbits around the Earth in this case. The x-axis is aligned with velocity vector and tangent to the orbit, the z-axis is perpendicular to the orbital plane and points towards the centre of the Earth, usually referred to as the Nadir. Finally, the y-axis is perpendicular to both axes.

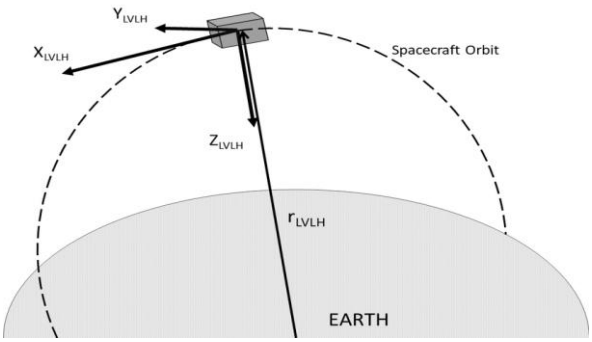

Figure 3.3 Orbital Frame

3.2.1.3. Spacecraft body frame

Satellite body frame is fixed to the satellite's body, with its origin at the centre of mass of the satellite. The body frame's principal axes are aligned with the satellite's principal axes, including its axis of rotation and axes of symmetry. All calculations are performed within this body frame as it directly belongs to the spacecraft's orientation and motion. Furthermore, utilising the body frame allows for a more convenient representation of the forces and moments acting on the satellite, such as the various torques experienced in space.

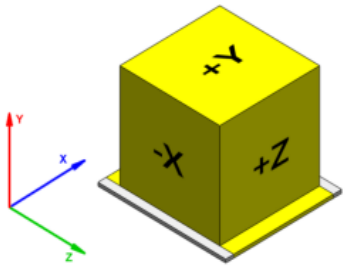

Figure 3.4 Body Frame

From Figure 3.4 is important to mention that the camera is located in the -Z axis so it would be the axis that points towards the Nadir while the Z axis will point towards the Zenith.

3.2.1.4. Representation of the 3 coordinate frames

The following Figure shows the 3 coordinate frames join together so once we have read the sections above, we could understand the whole coordinate system we will use in the simulations:

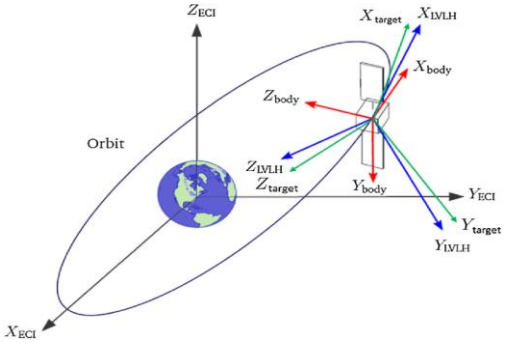

Figure 3.5 Coordinate Frames

3.2.2 Attitude Representation

Given the reference frames presented above, it can be defined the satellite's attitude as the orientation of the Spacecraft Body Frame with respect to the Orbital frame. In the Attitude Determination and Control algorithms developed during the final degree project Euler angles and Quaternions are used. In the following subsections it would be explained in detail the importance of these two forms of representing the orientation of the spacecraft.

3.2.2.1. Euler Angles

The orientation between any set of two different reference frames (for example, A and B) can be defined as the product of three consecutive rotations about the axes of one of the systems. The order of rotations about each axis depends on the transformation sequence so there exist multiple solutions which end in the same result. We start off with the standard definition of the rotations about the three principal axes apply in [8].

A rotation of ψ radians about the x-axis is defined as:

$$R_x(\psi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & -sen(\psi) \\ 0 & sen(\psi) & \cos(\psi) \end{pmatrix} \quad (\mathbf{3.2})$$

Similarly, a rotation of θ radians about the y-axis is defined as:

$$Ry(\theta) = \begin{pmatrix} \cos(\theta) & 0 & sen(\theta) \\ 0 & 1 & 0 \\ -sen(\theta) & 0 & \cos(\theta) \end{pmatrix} \quad (\mathbf{3.3})$$

Finally, a rotation of φ radians about the z-axis is defined as:

$$Rz(\phi) = \begin{pmatrix} \cos(\phi) & -sen(\phi) & 0 \\ sen(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (\mathbf{3.4})$$

Where φ, θ and ψ are the Euler angle known as Roll, Pitch and Yaw respectively. From the equations above and following the ZYX transformation sequence, the product of the three consecutive rotations can be defined as the following rotation matrix:

$$R = R_z(\phi)Ry(\theta)Rx(\psi)$$

$$\begin{pmatrix} c(\theta)\,c(\phi) & s(\psi)s(\theta)\,c(\phi) - c(\phi)\,s(\phi) & s(\psi)s(\theta)\,c(\phi) + s(\phi)s(\phi) \\ c(\theta)\,s(\phi) & s(\psi)s(\theta)s(\phi) + c\,(\phi)c(\phi) & s(\psi)s(\theta)s(\phi) - s(\phi)\,c(\phi) \\ -s(\theta) & s(\psi)\,c(\theta) & c(\psi)\,c(\theta) \end{pmatrix} \quad (\mathbf{3.5})$$

This is the rotation matrix for a specific transformation sequence, which is useful to transform a vector represented in a reference frame A, in its representation in a reference frame B. This transformation sequence is one of the most widely used parameterizations used to describe the attitude of an Earth pointing satellite.

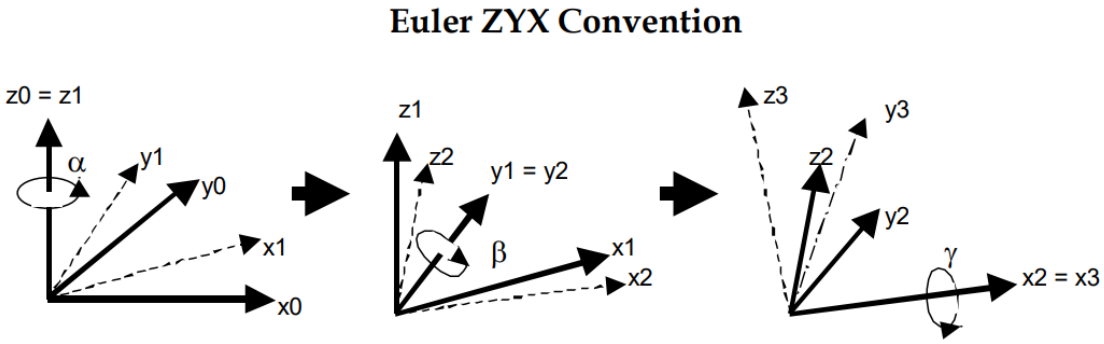

Figure 3.6 Transformation sequence ZYX

The main disadvantage of using Euler angles to represent the orientation of the spacecraft in space is the appearance of the phenomenon called Gimbal Lock [9]. Gimbal lock is the loss of one degree of freedom in a three-dimensional system, that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space. For example, considering a satellite that employs Euler angles to represent its orientation. As the satellite performs pitch, roll, and yaw movements, all three angles change accordingly. However, if the spacecraft completes a full 90-degree pitch, the pitch axis and the yaw axis will align. In this configuration, the roll axis becomes useless since any further rotation will not affect the spacecraft's orientation. This is called gimbal lock. This can lead to

unexpected results or errors in the system. To avoid gimbal lock, in this work we use an alternative method of orientation representation, which are the quaternions.

3.2.2.2. Quaternions

We consider a quaternion to be a four-component vector with some additional operations defined on it. A quaternion has a three-vector part and a scalar part. Hamilton in 1844 has defined a quaternion as [10]:

$$q = q_0 + iq_1 + jq_2 + kq_3 \qquad (\mathbf{3.6})$$

This is an hypercomplex extension of a complex number z=x+iy, with i, j and k obeying the following relationships:

$$i^2 = j^2 = k^2 = -1; \qquad ij = -ji = k; \qquad jk = -kj = i; \qquad ik = -ki = j \qquad (\mathbf{3.7})$$

Apart from the mathematical definition mentioned earlier, quaternions have an algebraic structure that relates to rotations in three-dimensional space. Specifically, they serve as a valuable tool for describing the orientation of a reference frame relative to another frame, rotating vectors, and combining successive rotations. Instead of using Equation 3.6, quaternions can be employed as an intuitive parameterization of attitude by representing them as a column vector with four parameters. The first term represents the scalar component, while the remaining three terms represent the vector component.

$$q = \begin{bmatrix} s \\ \vec{v} \end{bmatrix} = \begin{bmatrix} q_s \\ q_x \\ q_y \\ q_z \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\theta}{2}\right) \\ sen\left(\frac{\theta}{2}\right)\cos\left(\beta_x\right) \\ sen\left(\frac{\theta}{2}\right)\cos\left(\beta_y\right) \\ sen\left(\frac{\theta}{2}\right)\cos\left(\beta_z\right) \end{bmatrix} \qquad (\mathbf{3.8})$$

where θ is the transformation angle and $\cos(\beta_x), \cos(\beta_y), \cos(\beta_z)$ are the direction cosines of the angles between the three coordinate axes and the axis of rotation. Specifically, the vectorial part gives information on which axes the transformation angle is being applied while the scalar part gives information on how close are both frames.

The three quaternions that relate the different frames defined in previous sections are defined as:

$\mathrm{q_{e2o}} = \text{Quaternion that define the orientation of ECI frame with respect to orbit frame}$

$\mathrm{q_{o2b}} = \text{Quaternion that define the orientation of orbit frame with respect to body frame}$

$\mathrm{q_{e2b}} = \text{Quaternion that define the orientation of ECI frame with respect to body frame}$

The main advantage of the use of quaternions instead of Euler angles is the fact that quaternions have only four elements while the rotation matrix from Euler angles have nine elements. Moreover, quaternions do not have singularities so there would be no loss of degrees of freedom. The main drawback of quaternions is that the visualization is not clear and they are difficult to understand.

3.2.2.2.1. Quaternion identities

In order to work with quaternion, we need to explain the quaternion algebra that is implemented in the code.

Normalization: The quaternions must be normalized when we use it as attitude parameterization. This consists on the division of the quaternion by its norm, which must be unitary.

$$\|q\| = \frac{q}{|q|} \;\; where \;\; |q| = \sqrt{{q_s}^2 + {q_x}^2 + {q_y}^2 + {q_z}^2} = 1 \qquad (\mathbf{3.9})$$

Conjugate: The conjugate of a quaternion is obtained by changing the sign of the vectorial part of the quaternion. This operation would be useful if we want to swap from qe2o into qo2e for example (See Annex 2).

$$q^* = \begin{bmatrix} q_s \\ -q_x \\ -q_y \\ -q_z \end{bmatrix} \qquad (\mathbf{3.10})$$

Inverse: The inverse of a quaternion is essentially the conjugate of the quaternion divided by its magnitude. The operation of dividing by the magnitude normalizes the quaternion, which means that the resulting quaternion has a magnitude of 1.

$$q^{-1} = \frac{q^*}{|q|} \tag{3.11}$$

Multiplication: In this work, the use of the quaternion multiplication has two main functions. First, we can determine the rotation of a three-component vector as:

$$\overrightarrow{v_B} = q_{A2B} \otimes \vec{v}_A \otimes {q^{-1}}_{A2B} \tag{3.12}$$

The second application we use is transforming other quaternions by multiplying between them as:

$$q_{A2C} = q_{A2B} \otimes q_{B2C} \tag{3.13}$$

where $\otimes$ represents the multiplication between two different quaternions (e.g., q and q1), defined as:

$$q \otimes q_1 = \begin{bmatrix} q_s q_{1s} - q_s q_{1v} \\ q_{1s} q_v + q_s q_{1v} + q_v \; x \; q_{1v} \end{bmatrix} \tag{3.14}$$

Finally, is important to mention that quaternion multiplication is associative and distributive but it is not commutative so the order of the factors is essential in quaternion multiplication. (See Annex 3)

Derivative: The process of updating the PocketQube's attitude state involves determining the orientation of the Earth-Centred Inertial (ECI) frame relative to the Spacecraft Body frame at each simulation time step. This is achieved through

the derivation of the kinematic equation using quaternion parametrization. The derivation of the quaternion is defined as:

$$\dot{q} = \frac{1}{2} w \otimes \mathrm{q} = \frac{1}{2} \begin{bmatrix} -q^T{}_v \\ q_s I_3 + [q_v x] \end{bmatrix} w = \frac{1}{2} \begin{bmatrix} -q_2 & -q_3 & -q_4 \\ q_1 & -q_4 & q_3 \\ q_4 & q_1 & -q_2 \\ -q_3 & q_2 & q_1 \end{bmatrix} w \qquad (\mathbf{3.15})$$

3.2.3. Propagation Algorithms

3.2.3.1. Orbit Propagator

The Orbit Propagator is a model whose main objective is to determine the position of the satellite at any moment of time. If we assume that the Earth is a perfect sphere and only influenced by its gravitational field, solving this problem would be relatively straightforward. However, complications arise when considering additional factors such as the Earth's ellipticity, gravitational forces from the Moon and Sun, atmospheric drag, and solar pressure. Certain factors have a greater impact on satellite propagation than others. For instance, a satellite in a LEO orbit around the Earth is significantly affected by the Earth's ellipticity, while the gravitational forces exerted by the Sun and Moon, due to their considerable distance, have less influence. There are many propagators in existence, varying in computational time and accuracy. Ideally, the most accurate one would be the preferred choice, but in the work, we consider the limitations in the memory size of the OBC. First, the simulator uses the Two Lines Elements (TLE) [11] to define the initial state of the PocketQube. The TLE contains the following information about the picosatellite:

Card #	Satellite Number	Class	International Designator			Yr	Epoch Day of Year (plus fraction)	Mean motion derivative (rev/day /2)	Mean motion second derivative (rev/day2 /6)	Bstar (/ER)	Ephemeris type	Element number	Chk Sum
			Year	Lch#	Piece			S	S.	S. S E			
1	16609	U	86017A			93352.53502934		.00000788 9	00000-0	10529-3	0	342	
	Inclination (deg)		Right Ascension of the Node (deg)			Eccentricity		Arg of Perigee (deg)	Mean Anomaly (deg)	Mean Motion (rev/day)		Epoch Rev	Chk
2	16609		51.6190			134.3340		0005770	102.5680	257.5950		15.591140704478	69

Figure 3.7 Two Line Elements Format

Secondly, we implement the SGP4 model [12] which from the initial state define by the TLE set compute the PocketQube at any instant of time. SGP4 was created by NORAD [13] and NASA [14], is part of the Simplified General Perturbation (GSP) model, and only applies to satellites whose orbital period is less than 225 minutes. This model has a relatively good accuracy and it considers the perturbations mentioned previously.

Finally, it is important to mention that the outputs of the propagator are the position vector in [m] and the velocity vector in [m/s], relative to ECI frame. In the simulator, in order to define the state vector, we perform a unit conversion to [km] and to [km/s]. In order to see the orbit propagator simulated see Annex 4.

3.2.3.2. Geomagnetic field models

The Geomagnetic field models are also essential as they help us to know the magnetic field which acts on the pico-satellite. Even though magnetometers give us information about the magnitude of the magnetic field, they do not give any information about its direction and this is the reason why these models are very useful. The Geomagnetic field model predicts the magnetic field in each position and date. In this work two different model would be discussed.

International Geomagnetic Reference Field (IGRF) [15]: The IGRF is a standard mathematical description of the Earth's main magnetic field. It is used widely in studies of the Earth's deep interior, crust, ionosphere, and magnetosphere. It is updated every 5 years, and the current 13th edition of the IGRF model (IGRF-13), which was released in December 2019, is valid from 1900 until 2025.

Geomagnetic Dipole model of the Earth [16]: The dipole model of the Earth's magnetic field is a first order approximation of the rather complex true Earth's magnetic field. Due to effects of the interplanetary magnetic field (IMF), and the solar wind, the dipole model is particularly inaccurate at high L-shells, but may be a good approximation for lower L-shells. In our case, as the satellite is in a Low Earth Orbit (LEO), where it orbits at relatively low altitudes close to the Earth's surface, the dipole model of the Earth's magnetic field can be considered acceptable. This is because the effects of the interplanetary magnetic field (IMF) and the solar wind are relatively minor compared to higher altitudes.

The model that will ultimately be used is the most accurate one that can fit in the OBC’s memory. However, for the simulations in MATLAB we use the Geomagnetic Dipole model of the Earth (See Annex 5).

3.2.3.3 Sun position estimator

As the magnetic field, it is relevant to estimate the position of the sun with relative to the spacecraft. For this we use the approximation known as parallel vector approximation, which is a method that approximates the satellite´s position to the Earth´s centre. We consider this acceptable seeing that the distance between the Sun and the Earth is much greater than the altitude of the satellite. This means that, from the perspective of the Sun, the Earth and the satellite are almost in the same point of the space.

In conclusion, by knowing the date, we know in which part of the ecliptic the Earth is so we know its position. For the simulation, we use the MATLAB Toolbox of Meysam Mahooti [17] to compute the sun position (See Annex 6).

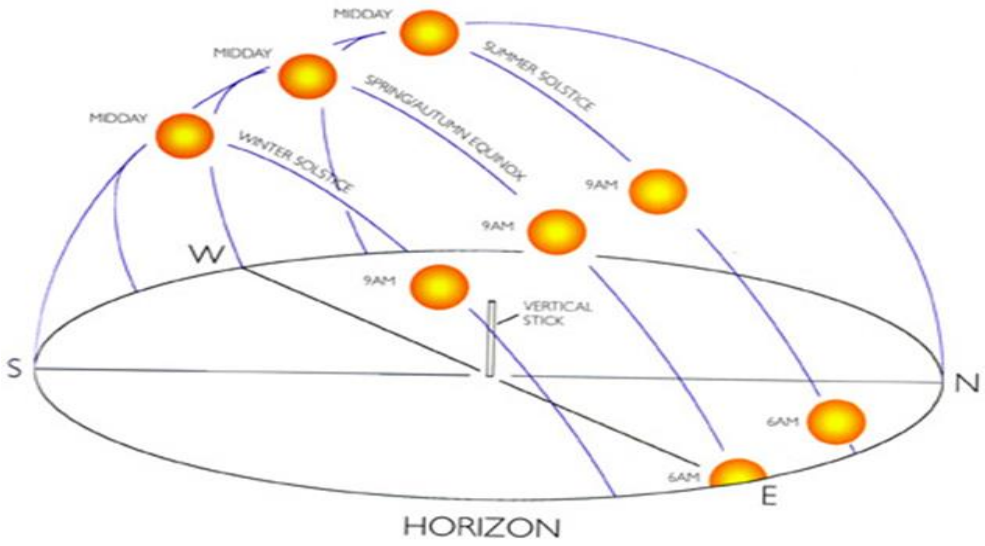

Figure 3.8 Sun position estimation

3.2.3.4. Atmospheric density

Since the PocketQube will be orbiting around a Low Earth Orbit (LEO), it will be affected by the aerodynamic perturbation force, explained in detailed in next section. This perturbation force is proportional to the atmospheric density so it is essential to know how it varies along the orbit.

In the simulator we use the NRLMSISE-00 (NRL Mass Spectrometer and Incoherent Scatter Radar Extended) [18] model which is an empirical model used to describe the Earth's atmosphere. It is specifically designed to estimate the density and composition of the neutral atmosphere from ground level to space

altitudes. The model considers various factors such as solar activity, latitude, longitude, season, and time of day to provide accurate atmospheric profiles.

NRLMSISE-00 incorporates data from multiple sources, including satellite observations and ground-based measurements, to create a comprehensive representation of the Earth's atmosphere. The last update done in the code use in the simulator was on the 15/08/2015. Nowadays, some coefficients could have changed as the NRLMSISE-00 is continuously updated and refined based on new data and research findings (See Annex 7).

3.3. External Disturbances

In this work, as external disturbances we only consider the gravity gradient torque, the aerodynamic torque and the optical torque. These torques would affect the picosatellite control and they are considered in every step of the simulation code.

3.3.1. Gravity Gradient Force and Torque

An object in low Earth orbit experiences different gravitational attraction in distinct parts of it, depending on their distance from the centre of the Earth. Although these forces are relatively weak, they are sufficient to stabilize certain satellites in a vertical orientation relative to the local horizon. This stabilization is not inertial but rather occurs in relation to a rotating frame. The moon serves as an example of an Earth satellite that is stabilized through the influence of gravity gradient. As a result, the Moon consistently presents the same side to Earth while orbiting around it. The gravity gradient torque arises from the variation in gravitational force acting on a body. First, we need to model the gravity force on a mass element dm of a body, which is located at a distance R from the centre of the Earth:

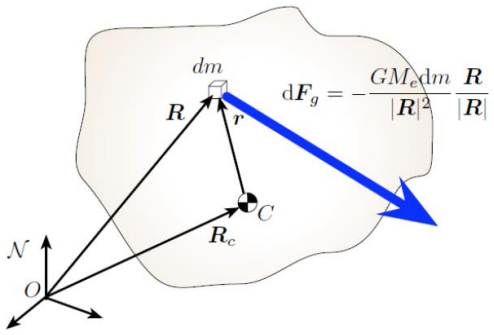

Figure 3.9 Gravity force over a body

The gravity force applied over the body could be define as:

$$dF_g = -\frac{\mu * R}{|R|^3} dm \qquad (\mathbf{3.16})$$

Where μ=GM is the gravitational parameter of the Earth. From the equation (3.16) we obtain the gravity gradient torque applied in the simulation code. For the whole process of the development of the equation written above in order to get the torque it is recommended to read [19]:

$$\tau_{gg} = \frac{3\mu}{|R_c|^5}(R_c \times I_c R_c) \qquad (\mathbf{3.17})$$

This is the most general form for the torque components collected in the (3x1) column vector (See Annex 8).

3.3.2. Aerodynamic Force and Torque

The aerodynamic drag force is generated when atmospheric particles collide with the satellite, resulting in a reduction of its orbital velocity. This force acts in the opposite direction to the satellite's movement [20]. These collisions can be simplified as purely elastic, meaning that the total energy of the incident particles is transferred to the satellite upon collision. The frequency of collisions depends on the density of the atmosphere. In the work, we use a simple algorithm that only considers the variations of density due to altitude. In LEO orbits, the atmospheric density is still large enough so that the drag force predominates over the rest of external disturbance forces. In the project, an approximation of the total aerodynamic force acting on the spacecraft is often used to obtain a conservative estimate of the aerodynamic torque. To determine the total aerodynamic force for specific spacecraft orientations, we employ the following expression obtain from [21]:

$$F_a = \frac{1}{2} C_D \rho V^2 A \qquad (\mathbf{3.18})$$

Where Cd is the drag coefficient, which for the purposes of a conservative estimate of the force is generally assumed to have a value of 2.7, ρ is the atmospheric density, V spacecraft velocity and A is the projected area of spacecraft element normal to the incident flow. The aerodynamic force presented above is exerted on the centre of pressure of the body, which as a simplification is considered coincident with the geometric centre. Since the centre of pressure can be at a distance r from the centre of mass, the aerodynamic torque is defined as (See Annex 9):

$$\tau_a = r \; x \; F_a \tag{3.19}$$

3.3.3. Optical Force and Torque

The Solar Radiation Pressure (SRP) torque is produced by the force exerted by solar radiation on the surface of a body. The sources of electromagnetic radiation that cause forces and possible torques to act on a spacecraft are defined in [22] as:

- Direct solar photon radiation.
- Solar radiation reflected by the Earth and its atmosphere.
- Radiation from Earth and its atmosphere.
- Radiation from the spacecraft.

The main cause of radiation torques is the direct radiation from the solar photon. The forces from other types of sources are usually at least one order of magnitude smaller. For the computation of the optical torque, we consider the solar wind negligible.

Solar photons coming from the Sun produce a pressure on the spacecraft surfaces that are illuminated. The effective pressure for Earth orbiting objects is shown in [23] as:

$$p = \frac{S}{c} = \frac{1367}{3e-8} = 4.557e-6\frac{N}{m^2} \tag{3.20}$$

Where S is the mean solar flux (1367 W/m^2) and c is the speed of the light. Earth-emitted radiation can be assumed as uniform over the surface of the earth. The flux is:

$$q = \frac{400}{\left(\frac{r}{R_{earth}}\right)} \qquad (\mathbf{3.21})$$

where r is the distance between the centre of the Earth and the satellite's altitude, and Rearth is the earth radius that is 6378.165 km. The 400 W/m^2 assumes that the Earth is a black body and has a temperature of 289.8 K. The contribution of albedo, which is the reflection of sunlight from the Earth's surface and clouds, presents a complex scenario as the Earth cannot be treated as a point source due to variations in reflectivity across its surface. In this study, the reflection from the Earth's surface and clouds is assumed to be diffuse, meaning that the reflected light scatters in multiple directions. To calculate the flux resulting from albedo, it is necessary to integrate over the entire surface of the Earth. The equation describing the flux is as follows:

$$q_a = \frac{S * a}{\left(\frac{r}{R_{earth}}\right)^2} \qquad (\mathbf{3.22})$$

Where a is the albedo factor which updates in every loop. As we mention before, the solar flux is the main source of the solar radiation pressure.

To model the solar radiation forces, incident radiation is assumed to be either absorbed, specular reflected and reflected diffusely. In terms of the fractions of the incoming radiation, the following is true for a surface:

$$1 = p_a + p_s + p_d \qquad (\mathbf{3.23})$$

Where p_a is the coefficient of absorption, p_s the coefficient of specular reflection and p_d the coefficient of diffuse reflection.

Once we define the different coefficients, we can model the force caused by the solar radiation as:

$$F_0 = -pA\hat{s}^T n \left(2\left(p_s \hat{s}^T n + \frac{p_d}{3}\right) n + (p_a + p_d)\hat{s}\right) \; for \; (\hat{s}^T n) > 0 \qquad (\mathbf{3.24})$$

Where $\hat{s}$ is the unit sun vector, T denotes its transpose, n is the unit normal to the surface, and A is the area of the surface. Moreover,$\hat{s}^T$ n is the dot product which corresponds to the cosine angle between sˆand n. In addition, the positive value means that the surface faces the sun direction. It is important to mention that the $\hat{s}$ could change due to the Earth`s radiation and albedo.

Finally, the torque of the optical force can be expressed as (See Annex 10):

$$\tau_o = r \; x \; F_0 \qquad (\mathbf{3.25})$$

3.4. Determination

3.4.1. Sensors

Sensors play a crucial role in the Attitude Determination as they provide essential data for accurately determining the attitude of a spacecraft. Various sensors are commonly employed in an ADCS to measure different parameters related to the spacecraft's attitude.

In this work, all the simulations have been done under noiseless conditions so the sensors are ideal. However, it is important to mention that in reality, all the sensors are noisy. This noise will be mainly Additive White Gaussian Noise (AWGN), and, assuming a correct calibration, it will have a null mean.

In the following sections the sensors used are explained.

3.4.1.1 Magnetometers

A magnetometer is a device that is used in attitude estimation to measure the local magnetic field. By comparing the measured magnetic field with the Earth's magnetic field at the current location, the attitude vector can be estimated. In order to compute the Earth's magnetic field, a model of the Earth's magnetic field is implemented, see section (Propagation Algorithms).

The magnetometers used in the Open PocketQube mission are the MMC5983MA [24] which is a fully integrated 3-axis magnetic sensor.

3.4.1.2 Photodiodes

A photodiode is a semiconductor device designed to detect sunlight and provide accurate information about the direction and intensity of incident solar radiation. The main objective of the photodiode in this work is to measure the position vector of the Sun, detecting the radiation emitted, in the body reference system.

The primary source of received power will be predominantly from the Sun. At any given instant, only three out of the six faces of the PocketQube will be in direct line of sight with the Sun, and these three faces will be perpendicular to each other. However, it is important to consider the Earth's reflectance caused by factors such as ice and clouds, also known as albedo. This could cause some faces that are not facing the Sun to receive almost as much power as the ones that are. In order to solve this problem, the criteria used in the implementation is to select, from each pair of parallel faces, the ones with the greatest reading in absolute terms. It is important to acknowledge that the functionality of photodiodes is contingent upon their dependence on solar radiation. Therefore, for the PocketQube to operate effectively, it must maintain direct visual contact with the Sun. During periods of eclipse when the satellite is shadowed from solar illumination, the photodiodes will be rendered incapable of carrying out their intended functions.

In Open PocketQube Kit mission, the photodiodes used are the SLCD-61N8 [25]. The SLCD-61N8 has high sensitivity in the spectral range of sunlight, which mainly spans from ultraviolet to near-infrared and this allows the photodiode to detect and measure solar radiation across a wide range of wavelengths. Moreover, the photodiode has directional response, meaning its sensitivity to light can vary with the angle of incidence. In the following diagram, we can see the radiation pattern of the photodiode where the directional sensitivity characteristics are shown:

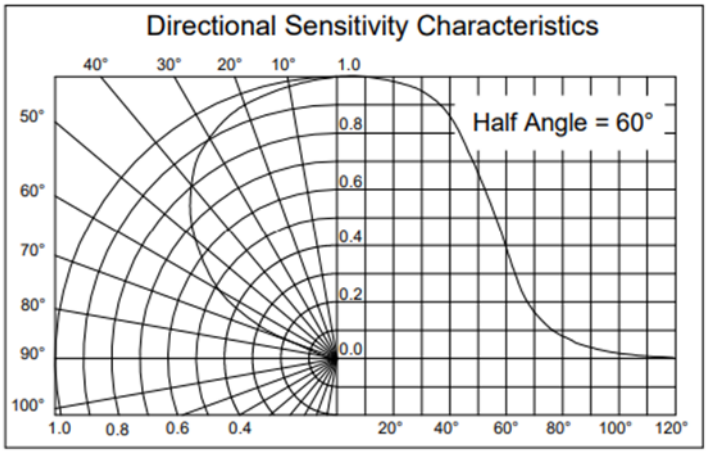

Figure 3.10 Radiation Pattern of the photodiode

3.4.1.3 Gyroscopes

A gyroscope is a device that measures angular velocity. These sensors are not only noisy, but also have a bias related to the random walk phenomenon.

The gyroscope use in the Open PocketQube Kit mission is the IIM-42652 [26] which is a 6-axis device that combines a 3-axis gyroscope and a 3-axis accelerometer.

3.4.2. Determination Algorithm

The purpose of the determination algorithm is to determine the orientation of the satellite with respect to a reference from two or more vectors. This is equivalent to obtain the rotation matrix, also called attitude matrix, from the reference frame to the body frame. In order to establish the attitude matrix uniquely, it is necessary to have minimum two sets of vector representations: one in the reference frame and another in the body frame. This means that a total of four representations (two for each vector) are needed. It is important to note that the magnitude of the vectors is not currently relevant, so, for simplicity, it is assumed that these vectors are unit vectors.

Considering two vectors A and B and their respective representations in reference and body frames Ar, Ab, Br and Bb. The attitude matrix M is defined as:

$$\left.\begin{aligned} M * Ar = Ab \\ M * Br = Bb \end{aligned}\right\} \quad (\mathbf{3.26})$$

Where M is orthonormal, by solving the system, the attitude would be determined.

However, the vectors A and B that will be used will have an error. The reason why this error appears could be seen in previous sections seeing that, for example, the propagation algorithms are models that are not perfect. Moreover, the sensors of the satellite will have also noise and systematic errors that cannot be control. These is the reason why we need to implement very good approximations so we reduce this error. Therefore, the matrix M would not be the attitude matrix, but it would approximate it.

In order to solve this problem, we study two methods explained in [27] which are the TRIAD Algorithm and the Wahba´s Problem.

TRIAD is based on the following idea. It is highly probable that one of the vectors used in the attitude matrix estimation may have more noise or uncertainty compared to the other vector. In such cases, the previous approximation may not be reliable or useful. However, the TRIAD algorithm offers a solution by prioritizing the more accurate vector during the estimation process. This prioritization leads to a significantly improved and more accurate estimation of the attitude matrix. According to the TRIAD algorithm, the attitude matrix can be estimated as follows:

$$M = AbAr^T + (Ab\ x\ Cb)(Ar\ x\ Cr) + CbCr^T \qquad (\mathbf{3.27})$$

where $$Cr = \frac{(Ab\ x\ Bb)}{\|Ab\ x\ Bb\|}, \qquad Cb = \frac{(Ar\ x\ Br)}{\|Ar\ x\ Br\|} \qquad (\mathbf{3.28})$$

It is important to note that this estimation prioritizes A over B, meaning that while M*Ar = Ab is true, M*Br = Bb is only true if the vectors are error-free. Variants of TRIAD have been proposed that use other combinations of the reference and body vectors to form the orthonormal triads. A more advanced algorithm, which is the Wahba´s problem, has replaced the previously mentioned method. This algorithm incorporates an arbitrary scalar weighting of the two vectors, allowing for more flexibility in the estimation process. By assigning appropriate weights to the vectors, the algorithm achieves improved accuracy and better adaptability to different scenarios. The upcoming derivation of this algorithm will provide a detailed understanding of how to utilize this scalar weighting approach effectively. For this reason, Wahba´s problem has been selected for the determination of the satellite and now it would be explained in detail.

In order to explain this method, we take as a reference the Study and development of attitude determination and control simulation software and control algorithms for 3Cat-4 mission of Carlos Díez García [28] as the use this method for the determination.

This method was proposed by Grace Wahba in 1965 [29] as a cost function defined by:

$$L(A) = \frac{1}{2}\sum_{i=1}^{n} a_i \|b_i - Ar_i\|^2 \qquad (\mathbf{3.29})$$

where n is the number of observation vectors, a_i is a weighting variable assigned to each observation, b_i are the observation vectors from the attitude sensors, A is the attitude matrix and r_i are the reference vectors corresponding to each observation. The main objective of this method is to minimize the cost function detailed in equation (3.29) in order to minimize the error of the common least squares curve fitting.

For solving this problem, we decide that the more convenient method was the q-method proposed by Davenport in 1966. The reason why we chose this method is because we are using quaternions for the attitude parametrization and this method is usually apply to compute the quaternion q corresponding to the optimal matrix A of the equation written above. For this, we apply the following equation:

$$\frac{1}{2}\sum_{i=1}^{n} a_i \|b_i - Ar_i\|^2 = \lambda_0 - q^T K q \qquad (\mathbf{3.30})$$

Where λ_0 the summatory of the weighting variables assigned to each observation and K is the Davenport matrix. The optimal quaternion is obtained by computing the unit eigenvector of the matrix, corresponding to its larger eigenvalue.

In the project, we use the Optimal-REQUEST [30] algorithm as the numerical implementation of the q method. This algorithm is an optimization of the REQUEST procedure [31]. The reason why this method will be implemented in the attitude simulator is because it uses measurements from the past so it gives a better quaternion estimation than other methods. Optimal-REQUEST recursive algorithm that propagates and updates the K matrix using the rotation velocity of the satellite, measured by the gyroscopes in the following way:

$$K_{k+1/k} = \emptyset_k K_{k/k} {\emptyset_k}^T \qquad (\mathbf{3.31})$$

Where $K_{k+1/k}$ the K matrix related to the satellite's attitude in the time instant tk+1, $\emptyset_k$ the rotation matrix computed using the angular velocity measured by the gyroscopes and $K_{k/k}$ the K matrix related to the satellite's attitude in the time instant tk.

Moreover, by using the vectors measured by the attitude sensors b_i and the reference vectors r_i, which are known, the matrix ∂K_{k+1}can be obtained. Once we have the estimated K matrix from the sensor measurements and the predicted K matrix, it can be computed the updated state as:

$$K_{(k+1)/(k+1)} = p_k \frac{m_k}{m_{k+1}} K_{k+1/k} + \alpha_{k+1} \frac{1}{m_{k+1}} \delta K_{k+1} \qquad (\mathbf{3.32})$$

Where p_k heuristic variable used to determine the percentage of application of the predictive block taking values between 0 and 1 (for example, p_k is equal to 1 if the predictive block is error free or 0 if the gyroscope measurements are very noisy), m_k and m_{k+1} are scalar coefficients that keep the highest eigenvalue close to one and αk+1 is the weighting coefficient of sensor measurements.

The Optimal-REQUEST algorithm is an adaptation of the REQUEST algorithm that incorporates Kalman filter techniques to determine the optimal value of the ρ gain. By considering the uncertainty caused by sensor noise in the measurement and the predictive blocks of the K-matrix system, the algorithm aims to find the best combination of both blocks for calculating the updated status. This update stage is referred to as the proposed update stage, where the goal is to account for the effects of sensor noise and determine the optimal approach for determining the updated status. Consequently, the update state would be defined as:

$$K_{(k+1)/(k+1)} = (1 - p_{k+1}) \frac{m_k}{m_{k+1}} K_{k+1/k} + \rho_{k+1} \frac{\delta m_{k+1}}{m_{k+1}} \delta K_{k+1} \qquad (\mathbf{3.33})$$

In this case, the value of the gain ρ is computed using the P, Q and R matrices, which are a measure of the uncertainty in gyroscope and attitude sensors measurements. Its value is defined by:

$$\rho_{k+1} = \frac{{m_k}^2 tr(P_{k+1/k})}{{m_k}^2 tr\left(P_{k+\frac{1}{k}}\right) + \partial {m_{k+1}}^2 tr(R_{k+1})} \qquad (\mathbf{3.34})$$

The value of ρ will be close to one if the confidence in the estimation due to sensor measurements is high or close to zero if, on the contrary, there is greater confidence in the predictive block (See Annex 11).

Finally, it is important to mention that this algorithm is already done but it cannot be implemented seeing that the calibration of the sensors is not done yet.

3.5. Control

In this section, two control algorithms which their respective control laws are discussed. Their main goal is to achieve the requirements established in the State of Art as ADCS-0030 and ADCS-0050. The first requirement is associated to the detumbling algorithm, which makes use of the measured magnetic field variation onboard the satellite to produce a magnetic moment whose resulting torque stabilizes the satellite. Regarding the second requirement mentioned, it is associated to the Nadir pointing algorithm which oversees orienting the camera of the PocketQube to the Nadir.

However, before we explain the control algorithms, we must explain in detail which are the actuators that generate the necessary external torques to fulfil these objectives and that are crucial for the control of a satellite.

3.5.1. Actuators: Magnetorquers

The actuators are the hardware components that generate the external torques, momentum transfer or physical interaction with the environment in order to execute the attitude control. When designing a satellite, the choice of actuator for various components, such as antennas, solar panels, or thrusters, is influenced by several factors including consumption, space, mass restrictions, and the required pointing accuracy. The performance of these actuators plays a crucial role in determining the overall performance of the satellite's control system. In Open PocketQube Kit mission the unique actuator we use are magnetorquers. Magnetorquers, also known as magnetic toques are a conductive wire winding that generate a magnetic dipole moment when excited electrically. The magnetic dipole moment generated by a magnetorquer is defined as:

$$m = NIA \tag{3.35}$$

Where N is the number of turns of the wire, A is the area of each coil and I is the electric current in the wire.

Attitude control using magnetic torquers was first proposed in the early 1960s. One of the main uses of magnetic torquers is to dump excess momentum induced by external disturbances [32]. Magnetic control torques offer numerous benefits for near-Earth missions. These advantages include the smooth application of

torques, virtually unlimited mission lifespan (as they do not rely on expendable resources), and the absence of catastrophic failure modes.

Magnetic torquers use the Earth's magnetic field to produce a torque. This torque could be defined as:

$$\tau = m\ x\ B \tag{3.36}$$

Where m is the magnetic dipole moment explained in Equation (3.35) and B is the local geomagnetic field expressed in body-frame coordinates.

The main disadvantage of magnetorquers compared to other actuators such as reaction wheels, used primarily by spacecraft for three-axis attitude control, is their limited ability to provide torque. Magnetorquers generate torque using the Earth's magnetic field, which means their effectiveness decreases as the satellite moves away from Earth or enters a region of space with a weak magnetic field. In contrast, reaction wheels can independently generate torque using the conservation of angular momentum, giving them greater manoeuvrability and a faster response compared to magnetorquers. In addition, magnetorquers are only able to produce perpendicular torques while the reaction wheels can produce torques in any direction. In the simulator, the magnetic moment generated by the magnetorquers is calculated from the control law, which returns the necessary torque on each axis to achieve the desired pointing, by applying the following equation:

$$m = \frac{B\ x\ \tau}{\|B\|^2} \tag{3.37}$$

This expression comes from introducing the orthogonality condition between the vectors m and B detailed in reference [33]. The resulting magnetic moment is just a theoretical result as we must assure that the magnetic moment is not bigger than the maximum magnetic dipole moment. In order to consider that, we model its saturation by limiting the value of m for the thee axes of the magnetorquer as in the following expression:

$$m = m_{min}(\max(m, -m_{max}), m_{max}) \tag{3.38}$$

Where m_{max} is the maximum magnetic dipole moment that the magnetorquers can generate.

Finally, I will show the design of the bottom Board coil and the Lateral Board of the magnetorquer:

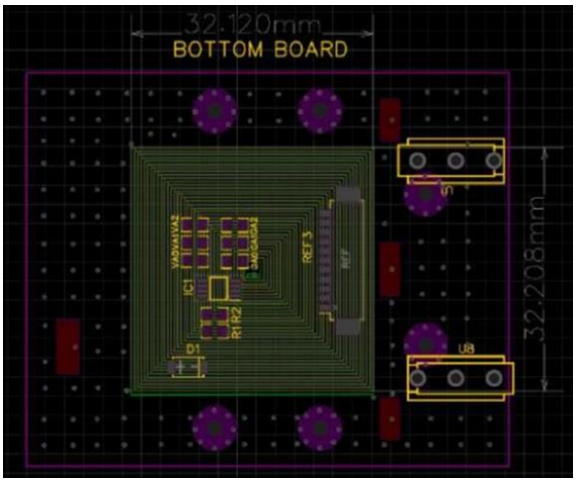

Figure 3.11 Bottom Board Coil Design

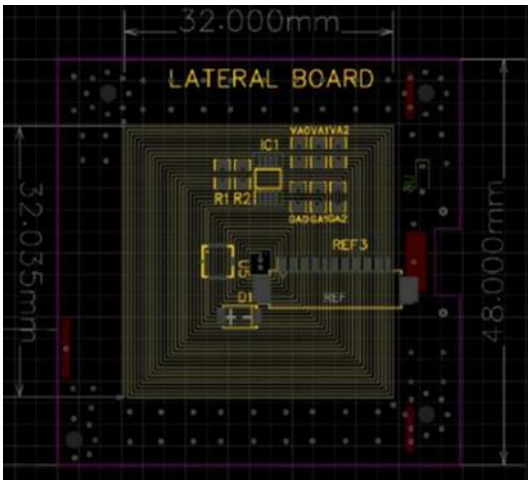

Figure 3.12 Lateral Board Design

3.5.2. Control Algorithm

3.5.2.1. Detumbling Algorithm

The detumbling algorithm uses the Bdot algorithm to check if the satellite is stabilized. B-dot algorithm uses the variation of the magnetic field over a small time period to determine the control torque to be employed on the body. It relates the change in magnetic field observed in the body frame to the magnetic moment required in space to counter the angular rates. In this work, the main objective of this algorithm is to reduce the angular velocity of the pico-satellite from 30º/s (worst case consideration of the initial state) to 2º/s on each axis using magnetic actuation.

The magnetic moment produced under B-dot actuation:

$$m = -k * \dot{B} \qquad (\mathbf{3.39})$$

Where $\dot{\vec{B}}$ is the rate of change of the measured magnetic field and k is a positive scalar gain. Ideally, k is inversely proportional to the elements of the rotational inertia. In order to find the optimal value of k, trial and error is used. In our case, k has a value of 1e6.

A different approach to the proportional controller is the Bang-Bang algorithm mentioned in [34], which operates by constantly applying the maximum torquer strength instead of calculating the commanded magnetic moment with a proportional gain factor. Using this controller offers the advantage of faster spin rate decay when compared to proportional B-Dot control. Consequently, the Bang-Bang controller exhibits higher efficiency since it always generates the maximum achievable torque, unlike the proportional controller.

Finally, the torque is given by the cross product of the magnetic dipole moment and the external magnetic field (See Annex 12):

$$\tau = m\ x\ B \qquad (\mathbf{Fig.\ 3.40})$$

3.5.2.2. Nadir Pointing Algorithm

Nadir pointing belongs to the regulation control which is defined as bringing the attitude to some fixed location, usually the identity quaternion, and the angular velocity to zero. In this work the main goal is to point the satellite´s camera towards the Earth. In order to do that, we use the non-linear controller defined in [35]. First, as we want to point towards the nadir, we must align the body frame with the orbit frame seeing that the z-axis of the orbit frame points towards the centre of the Earth. When expressed in terms of quaternions, this implies that the quaternion representing the rotation of the satellite relative to the orbit frame is desired to be $q = [1\ 0\ 0\ 0]^T$. This means, the vector part of the quaternion is cero so both frames are aligned. Finally, the controller use in this work is a PD controller based on the work of Kristian Lindgard Jenssen and Kaan Huseby Yabar in [31] and is given by:

$$m = \frac{1}{\|B\|}\left(-k_p(B\ x\ \varepsilon) - k_d(B\ x\ w)\right) \qquad (\mathbf{Fig.\ 3.41})$$

Where B is the magnetic field in body reference, w is the angular velocity, k_p and k_d are the proportional and derivative gains which are positive and ε is the vector part of the quaternion error between the actual quaternion and the desired one.

In the attitude simulator the values of kp and kd obtained are respectively 4.8805e-4 and 0.0355.

This control law is very suitable for the purpose of the mission as this law will always calculate a gain assuming the magnetorquers are perpendicular to the local geomagnetic field. Moreover, this law is projected it to a plane always being perpendicular to the local geomagnetic field. Consequently, we ensure that the control gains obtained are the best we can obtain.

Finally, the magnetic control torque is defined as (See Annex 13):

$$\tau = \frac{1}{\|B\|}\left(-k_p(B\ x\ \varepsilon) - k_d(B\ x\ w)\right)\ x\ B \qquad (\mathbf{Fig.\ 3.42})$$

CHAPTER 4. RESULTS

In this section will be presented the results of the different simulations performed with the developed attitude simulation software.

4.1. Detumbling Mode

In this subsection, we present the results obtained from the detumbling algorithm. As we state in the ADCS requirement ADCS-0030 the satellite we must be able to control the rotation of the satellite. For this reason, we have made the following test with the next initial conditions:

- The starting date of the simulation will be the first of January of 2023 at 00:00.
- The pico-satellite describes a polar orbit starting at altitude of 500 km.
- The number of orbit revolutions simulated are 10 with a time step of 0.5 s.
- The PocketQube is in stowed configuration.
- The perturbation torques simulated are the aerodynamic force, the gravity gradient force and the optical force.
- The sensors are considered ideal.
- The initial state attitude is random with an angular velocity of 30 degrees/second in each axis.

Considering the following assumptions, the results obtained when executing the simulation are:

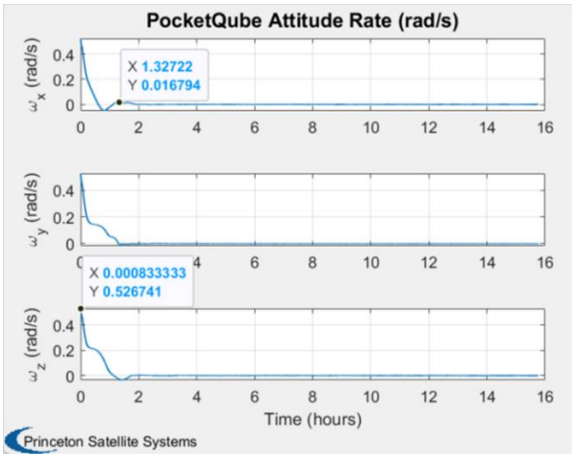

Figure 4.1 PocketQube Attitude Rate

The Figure above represents the most important result for the detumbling algorithm. In the Figure we observe how the angular velocity decrease from 30 degrees/s, equivalent to 0.527 rad/s, to almost 0 rad/s. This means the detumbling algorithm can encounter the most critical angular velocity so we can assure the algorithm works well. In this simulation, we can see that the detumbling is done successfully in 1 hour and a half more a less.

Another important aspect are the magnetic dipole moment and the current apply by the magnetorquer. As we already mention in the subsection, the magnetic dipole moment and the current are proportional so we should expect similar graphs.

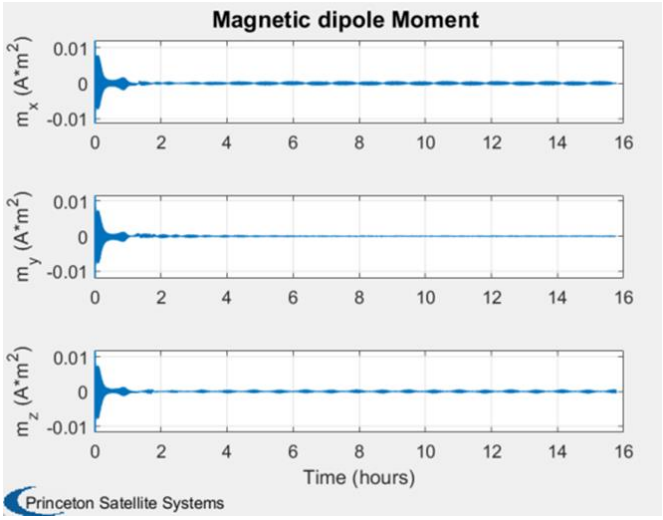

Figure 4.2 Magnetic dipole moment

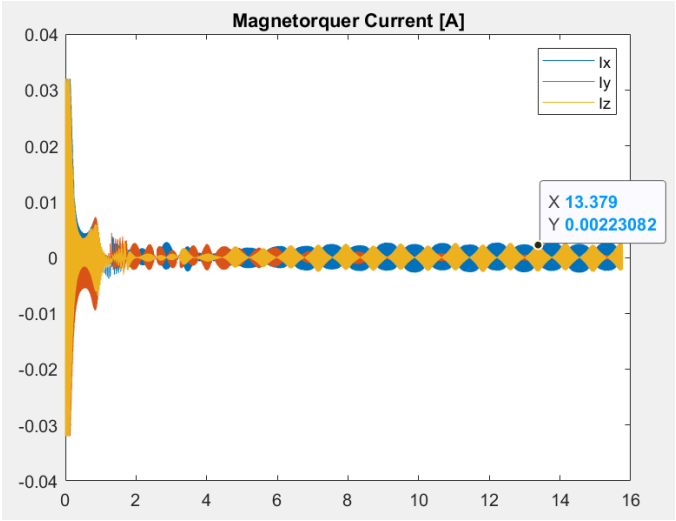

Figure 4.3 Current apply by the magnetorquers

From this two Figures, we observe similar behaviours as we have expected. First, when the angular velocity is higher, the satellite is not control so the dipole moment generated by the magnetorquers must be greater to decelerate the satellite. Consequently, the current applied to generate that moment should be higher. In fact, we can observe that the maximum current is applied which is 32 mA. Finally, when the satellite is almost control, which means that the satellite angular rate is almost 0, the moment and the current generated decrease significantly. Finally, we will compare the magnitude of the disturbance torques simulated with the magnitude of the magnetic control torque generated by the magnetorquer in order to see if they really affect the satellite control.

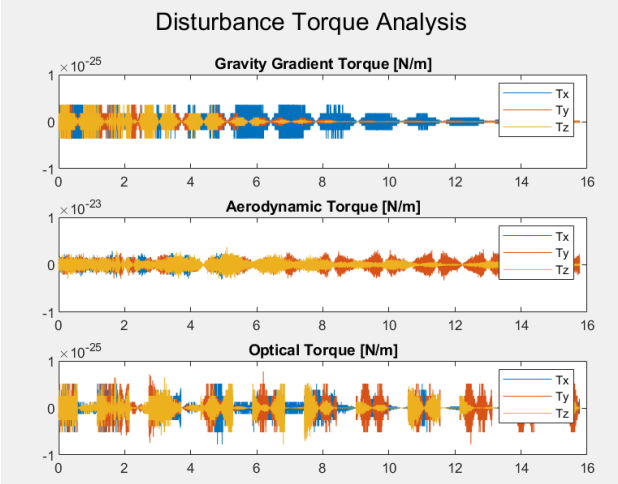

Figure 4.4 Disturbance Torques

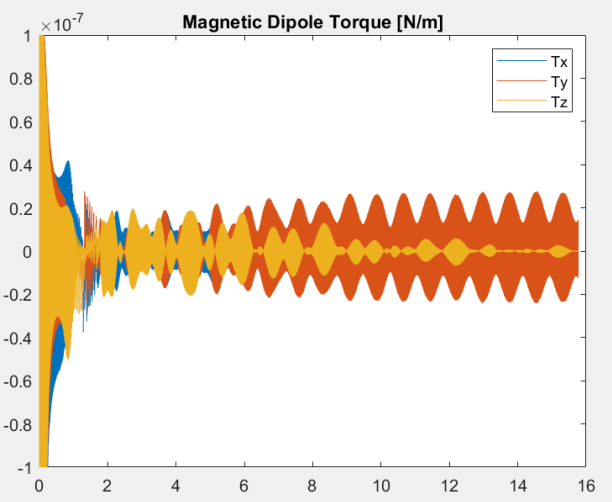

Figure 4.5 Magnetic Dipole Torque

From the Figure, we can conclude that the Disturbance torques are insignificant relative to the torque generated by the magnetorquers. The torque generated by the magnetorquers exceeds the total disturbance torque by a factor of 10^6, allowing the PocketQube to effectively overcome these torques. Furthermore, the control system exhibits increased activity during specific time intervals, particularly at the initial stages when the difference between the satellite's angular velocity and the desired target velocity is more pronounced.

4.2. Nadir Pointing Mode

In this subsection, we present the results obtained from the nadir pointing algorithm. As we state in the ADCS requirement ADCS-0050 the satellite must be able to point at a specific location using the nadir algorithm. Moreover, the ADCS department has established that the maximum error angle we can have in order to consider the algorithm successful is 10º. For this reason, we have made the following test with the next initial conditions:

- The starting date of the simulation will be the first of January of 2023 at 00:00.
- The pico-satellite describes a polar orbit starting at altitude of 500 km.
- The number of orbit revolutions simulated are 10 with a time step of 0.5 s.
- The PocketQube is in stowed configuration.
- The perturbation torques simulated are the aerodynamic force, the gravity gradient force and the optical force.
- The sensors are considered ideal.
- The initial state attitude is known with null angular velocity in each axis.

Considering the following assumptions, the results obtained when executing the simulation are:

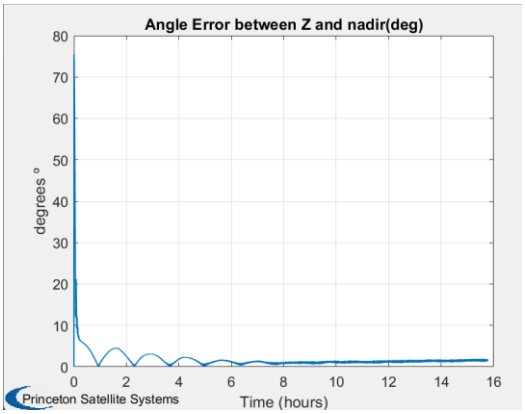

Figure 4.6 Angle error

The first Figure represents the error angle between the z-axis and the nadir pointing. This is the most important result as it shows if the pointing is done correctly. From the Figure, we observe that the algorithm works perfect with an angle error of almost cero.

Moreover, as we are analysing a specific attitude target such as nadir pointing, it is necessary to show the attitude of the satellite. For this reason, the next two figures represent the angular velocity of the pico-satellite and the ECI to Body quaternion.

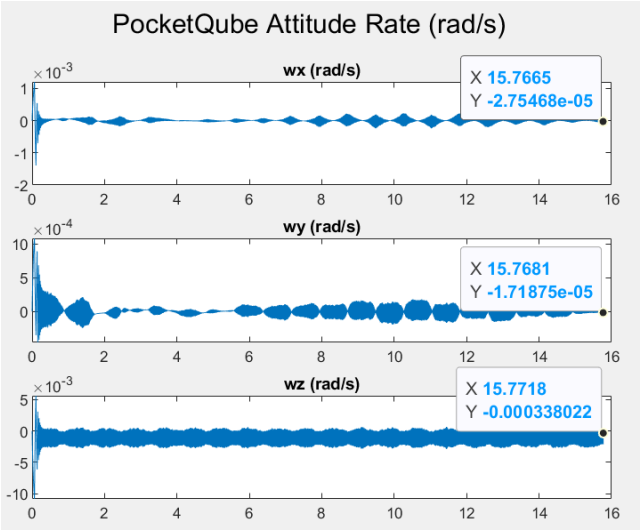

Figure 4.7 PocketQube Attitude rate

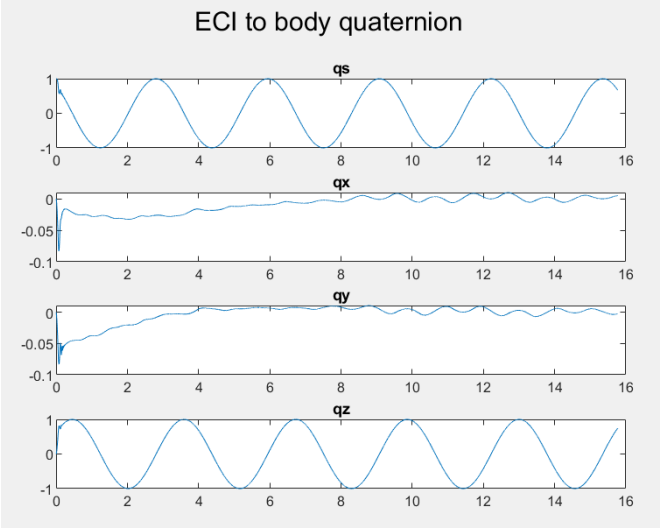

Figure 4.8 ECI to Body quaternion

From these two figures we could notice that the z-axis rotates freely since has no pointing requirements while the other two axis are always in control seeing that their vector quaternions parts are almost 0. In addition, in the Figure that shows the attitude rate, we can also see this effect as the angular velocity in the z-axis is 10 times bigger than in the rest of the axis.

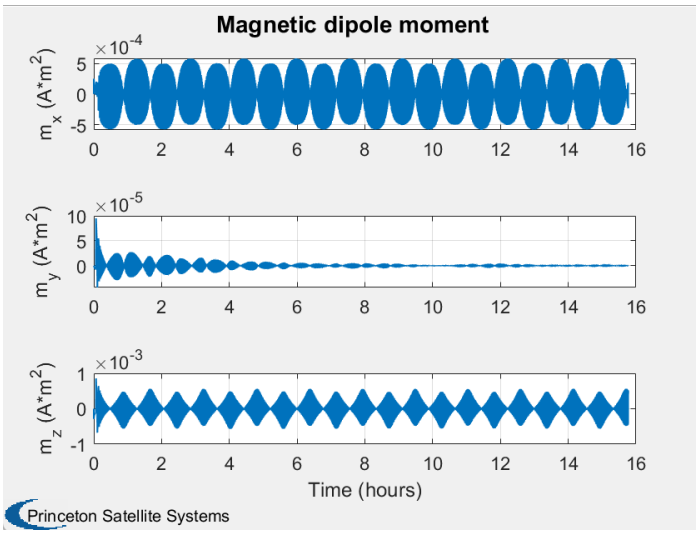

Figure 4.9 Magnetic Dipole moment

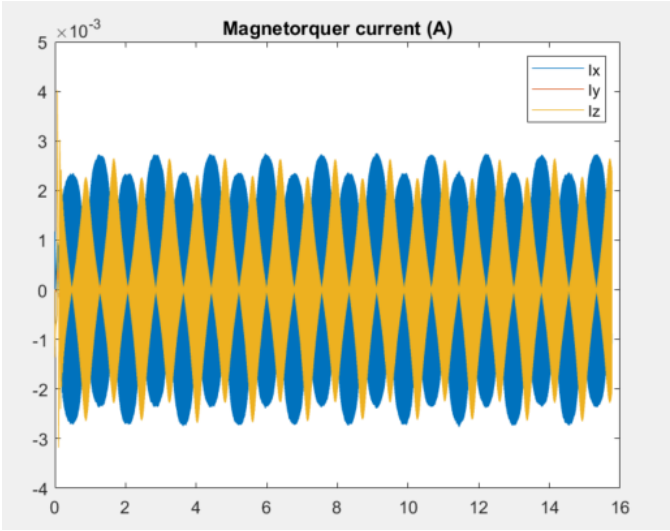

Figure 4.10 Magnetorquer Current

In this Figures we can observe that the magnetic dipole moment and the current generated by the magnetorquers are lower than the ones obtain from the detumbling. The reason why this happens is because as the initial angular rate is 0 rad/s, is easy to achieve complete control.

Moreover, as it was explained in section (3.5.2.2) nadir pointing algorithm control law also brings angular velocity to 0 so it should be able to do detumbling. In order to see this we would maintain the initial conditions except the angular velocity that would be change to 30 degrees/s and this are the results:

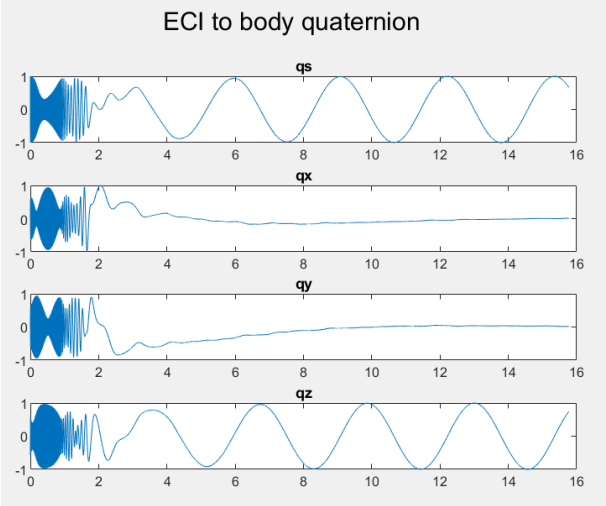

Figure 4.11 ECI to Body quaternion

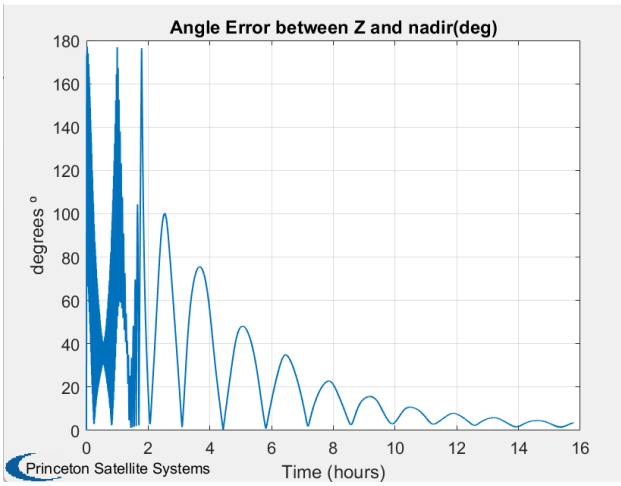

Figure 4.12 Angle error

From this two Figures we can observe that at the beginning the satellite is in detumbling so the satellite is not under control and consequently, the angle error and the $q_{eci2body}$ oscillates. Once the angular velocity of the satellite starts decreasing, the angle error does the same and finally we obtain the nadir pointing with an error less than 10 º so the objective is fulfilled.

Just in case with this explanation is not clear that the nadir pointing algorithm does detumbling, the following graph represents the attitude rate of the PocketQube:

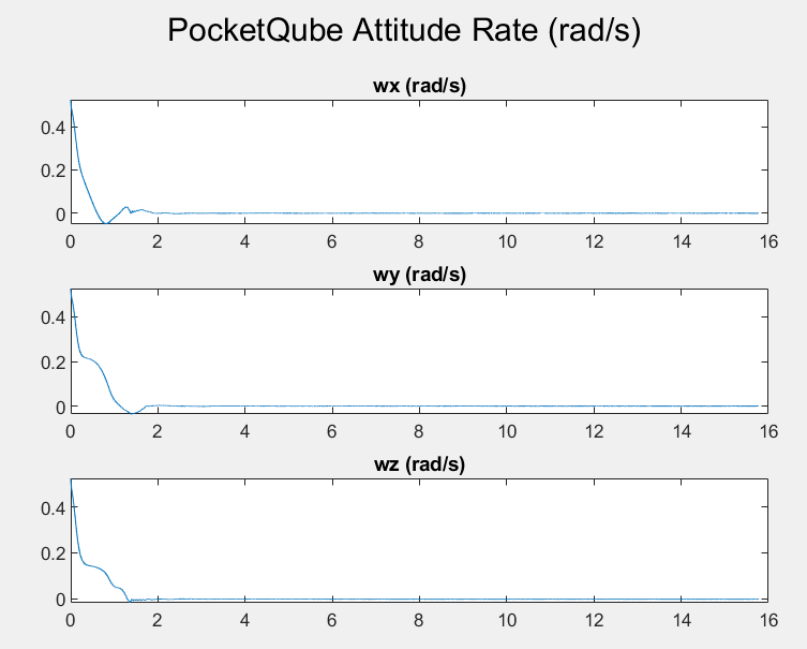

Figure 4.13 PocketQube Attitude rate

CHAPTER 5. BUDGETS

The project budget serves as a representation of the projected expenditure needed to successfully carry out our project. It offers a comprehensive perspective on the monetary resources required to complete each phase of the project. Within the project budget, there are two distinct categories of expenses: direct costs and indirect costs. Direct costs encompass the materials and labour necessary for constructing the project. On the other hand, indirect costs encompass various expenditures such as equipment depreciation, facilities, and software licenses. On the one hand, as the direct costs we only will mention the labour costs seeing that almost all the project is done by MATLAB and C. In order to do that, we will only consider the ADCS department which was formed by two PAE students and me. The average salary of an Engineer student in Barcelona is of 10 €/hour while the average salary of a TFG student is of 11 €/hour.

Given that the dedication of this project has been 4 months and considering that the TFG are 18 credits where each credit are 25 hours a total of 450 hours of work have been spend in the project. Finally, this are the total direct cost of the project:

Table 5.1. Direct costs

Role	Employees	Hours	€/hour	Base	Social Security	Total
Engineer Student	2	235	10	4.700 €	1.316 €	6.016 €
TFG student	1	450	20	9.000 €	2.520 €	11.520 €
Total						

On the other hand, as indirect costs the PAE student have computed a study in order to notice the equipment depreciation and the facilities costs. Finally, the software cost we will also be considered.

Table 5.2 Facilities costs

Facilities	Costs/month	Costs/week	Weeks	Total
Light	200 €	50 €	14	700 €
Internet	60 €	15 €	14	210 €
Premises	1.080 €	270 €	14	3.780 €
Total				4.690 €

Throughout the project's development, various equipment will be utilized, including personal computers, welding tools, power supply units, RF equipment, and more.

These items were acquired for a total cost of 40.000 €, and they are expected to remain in service for a duration of 15 years. The subsequent table depicts the expenditure incurred over the course of the project's 14-week timeline.

Table 5.3. Equipment depreciation

Initial cost	Useful life (years)	Salvage value	Expense per year	Expense per week	Weeks	Total
40.000	15	3.000 €	2.000 €	41,6 €	14	583 €
Total						583 €

The software cost are basically the MATLAB licence and the toolbox use which is the CubeSat Control Toolbox.

Table 5.4. Software Costs

Software	Cost per year	Cost per week	Weeks	Total
MATLAB	250 €	5 €	14	70 €
CubeSat Toolbox	485 €	10 €	14	140 €
				1.372 €

Finally, a table with all the indirect costs is added:

Table 5.5. Indirect Costs

Expense	Total
Facilities	4.690 €
Equipment Depreciation	583 €
Software	1.372 €
Total	6.645 €

CHAPTER 6: ENVIRONMENTAL SUSTAINABILITY

The purpose of conducting a Sustainability analysis is to evaluate the environmental impact of the project. The NanoSat Lab group is committed to minimizing the CO2 emissions. This commitment extends to the satellite itself, as conventional satellite launches contribute significantly to CO2 emissions. To address this issue, the group has partnered with Fossa System to launch the PocketQube with the Fossasat, resulting in a significant reduction in CO2 emissions seeing that the highest amount of CO2 emissions comes directly from the launching of the satellite. The satellite design itself focuses on reducing size and material usage. NanoSat Lab has also taken a selective approach to purchasing new materials and has been cautious in the management of energy and sources. Although the exact quantification of emission reduction is unavailable, it is estimated that the launch alone will reduce CO2 emissions by 100,000 kg. Moreover, many suppliers publish environmental reports. For example, STMicroelectronics [36] is the supplier of all the nucleus boards used in the project and environmental reports are found in [37]. However, implementing a cradle-to-cradle approach for the PocketQube is very difficult seeing that most of the components of the satellite cannot be reused, and even those that can be reused can suppose risks according to our standards. For this reason, once satellite reaches the end of its lifespan, it will naturally disintegrate due to atmospheric conditions upon re-entry into Earth. Another important environmental issue is the increase of space garbage due to the increase of educational satellites launches to the space. As we already mention in the introduction, there has been an increase in the accessibility to space so many university research groups have been able to develop their own missions with educational and scientific objectives such as Earth observation, interference studies and climate change.

In order to see graphically this increase of satellites launched, the following graph is shown:

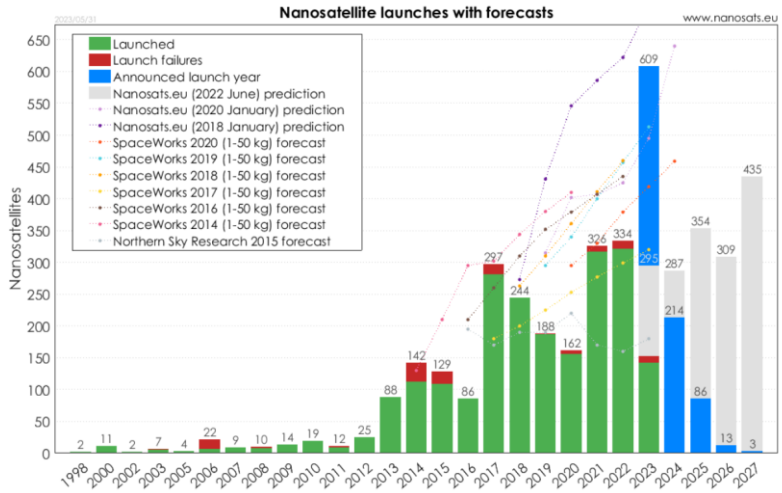

Figure 6.1 Nanosatellites launches

From the Figure we observe that this increase is exponential so in the future we might expect more space garbage due to congestion of LEO orbits and consequently, a higher probability of collisions.

The most efficient way to immediately reduce the growth rate of space debris is by preventing in-orbit explosions through the passivation of space objects once they reach the end of their operational life. Likewise, avoiding collisions through proactive collision avoidance manoeuvres while the objects are still active is crucial. Furthermore, long-term stability and safety in the space junk environment can be achieved by strictly adhering to post-mission disposal guidelines. Compliance with these guidelines is considered the most effective approach for maintaining a sustainable and secure space debris environment over time.

In fact, according to ESA's internal research, delaying the implementation of continuous removal actions until 2060 would result in a 25% decrease in effectiveness compared to starting them immediately.

To prevent uncontrolled growth of the garbage environment, it is essential to apply both types of mitigation measures extensively and promptly. Insufficient or delayed implementation of these mitigation concepts could lead to a collisional cascading process, particularly in orbit regions such as the crucial 800-1400 km altitude range. This could make these regions extremely hazardous for space activities within a few decades. Therefore, it is crucial to take timely and comprehensive action to safeguard the sustainability of space operations. More information about this problem could be found on [38].

CHAPTER 7: FUTURE WORK

In this work, a complete study of the Attitude Determination and Control System of a picosatellite has been done. Moreover, an ADCS software has been implemented for the Open PocketQube Kit mission. This means, it has been tested in MATLAB and implemented in C so it can communicate with the other subsystems of the satellite. The future work that must be done to continue the complete development and integration of the ADCS will be explained in this subsection.

- Calibration of the sensors (magnetometers, photodiodes and gyroscopes) in order to know the different bias parameters that affect the measurements.
- Add the determination algorithm done in the attitude simulator and see how the error of the sensors affected the determination and control of the PocketQube.
- Discuss with the other members of the mission which parameters we want to send and receive from the satellite in relation to the ADCS.
- Do functional and performance tests of the algorithms done in order to know how it works on reality.

CHAPTER 8: CONCLUSIONS

The goal of this project, focused on researching and creating a simulation software tool for the attitude determination and control subsystem, as well as designing and implementing control algorithms for the Open PocketQube Kit mission, has been effectively accomplished.

The Attitude Determination and Control Subsystem (ADCS) plays a crucial role in satellite missions with specific pointing needs. For instance, in the Open PocketQube Kit mission, it is of great importance for the camera to accurately align with the Nadir direction to capture Earth images. To fulfil this requirement, the ADCS of the Open PocketQube Kit mission utilizes attitude sensors such as magnetometers, gyroscopes, and photodiodes, along with magnetorquers as actuators.

To assess the performance of this subsystem and validate compliance with the mission's specifications, a simulator has been developed as part of this thesis. The simulator serves the purpose of analysing the subsystem's functionality and ensuring that the desired objectives are achieved.

To design and develop the ADCS simulator, a research work has been carried out, analysing the state of the art of the ADCS subsystems and the simulation software used for this type of systems. With that, it has been defined a table of requirements in which is detailed the characteristics that shall have the project in order to meet the defined objectives.

Taking into consideration the block diagram illustrated in Figure 3 of subsection 2.2, which represents each of the sections that are part of this project, the physics behind each block have been studied and referenced in this document so that it serves as a valuable reference for the simulator developed. Specifically, in this project the following points have been covered:

• The geometry and configuration of the PocketQube has been defined using simple geometries and in accordance with the "PocketQube Standard" document.

• The simulation environment has been generated by implementing algorithms for the propagation of the orbit (SGP4 algorithm), Earth’s magnetic field (Geomagnetic Dipole model of the Earth), Sun position (Mahooti algorithm) and atmospheric density (NRLMSISE-00), and developing models of external perturbation forces (Gravity Gradient torque, Aerodynamic drag torque and Optical torque) that affect the satellite.

• To develop the determination system in the simulator, the Optimal-REQUEST algorithm has been created. This algorithm would consider the error in the measurements of the sensors.

• To develop the control system in the simulator, two control algorithms have been developed, corresponding to each of the control modes of the Open PocketQube mission: detumbling mode and nadir pointing mode.

• The simulator has been used to analyse the performance of the ADCS subsystem for the Open PocketQube Kit mission. The results obtained (see section 5) show that the subsystem requirements are met. On one hand, during the detumbling mode the PocketQube is capable to reduce the speed of each of the axes below 2°/s in almost 1.5 hours. On the other hand, during the Nadir pointing mode the PocketQube is capable to reach the Nadir pointing within a maximum deviation margin of ±10°and an angular velocity lower of almost cero.

REFERENCES AND BIBLIOGRAPHY

[1] NanoSat Lab-UPC. https://nanosatlab.upc.edu/en/presentation

[2] IEEE GRSS. https://www.grss-ieee.org

[3] Alba Orbital, TU Delft, and Gauss. The PocketQube Standard, Issue 1. http://www.albaorbital.com/pocketqube-standard

[4] Princeton CubeSat Toolbox. https://www.psatellite.com/products/sct/cubesat-toolbox

[5] Alba Orbital. http://www.albaorbital.com

[6] Fossa Systems. https://fossa.systems/es/home-espanol

[7] Fundamentals of Astrodynamics and Applications, 4th ed. (Space Technology Library) by David A. Vallado (2013-03-29)

[8] Computing Euler angles from a rotation matrix - Gregory G. Slabaugh

[9] Avoiding Gimbal Lock in a Trajectory Simulation - David Hosier

[10] On Quaternions By William Rowan Hamilton

[11] TLE format https://celestrak.org/NORAD/documentation/tle-fmt.php

[12] SGP4 Orbit Determination. https://celestrak.org/publications/AIAA/2008-6770/AIAA-2008-6770.pdf

[13] NORAD https://www.norad.mil

[14] NASA https://www.nasa.gov

[15] International Geomagnetic Reference Field https://www.ncei.noaa.gov/products/international-geomagnetic-reference-field

[16] Geomagnetic Dipole model of the Earth https://en.wikipedia.org/wiki/Dipole_model_of_the_Earth%27s_magnetic_field

[17] Meysam Mahooti (2023). Sun Position (https://www.mathworks.com/matlabcentral/fileexchange/56042-sun-position), MATLAB Central File Exchange. Retrieved June 22, 2023.

[18] NRLMSISE-00 Empirical Model Of The Atmosphere: Statistical Comparisons and Scientific Issues

[19] Gravitational torque. http://www.ingaero.uniroma1.it/attachments/1978_5_EnvironmTorques.pdf

[20] J. R. Wertz, Spacecraft Attitude Determination and Control, J. R. Wertz, Ed. D. Reidel Publishing Company, 1978, vol. 73, no. September

[21] Spacecraft aerodynamic torques by National Aeronautics and Space Administration https://ntrs.nasa.gov/api/citations/19710016459/downloads/19710016459.pdf

[22] Spacecraft radiation torques by National Aeronautics and Space Administration https://ntrs.nasa.gov/api/citations/19710014836/downloads/19710014836.pdf

[23] Comparison of Solar Radiation Torque and Power Generation of Deployable Solar Panel Configurations on Nanosatellites by Syahrim Azhan Ibrahim and Eiki Yamaguchi https://www.mdpi.com/2226-4310/6/5/50

[24] MMC5983MA. https://eu.mouser.com/datasheet/2/821/Memsic_09102019_MMC5983MA_Datasheet_Rev_A-1635338.pdf

[25] SLCD-61N8. https://www.farnell.com/datasheets/125186.pdf

[26] IIM-42652. https://invensense.tdk.com/products/smartindustrial/iim-42652

[27] Fundamentals of Spacecraft Attitude Determination and Control F. Landis Markley John L. Crassidis

[28] Study and development of attitude determination and control simulation software and control algorithms for 3 Cat-4 mission of Carlos Díez García.

[29] G. Wahba, “A Least Squares Estimate of Satellite Attitude,” SIAM Review, vol. 7, no. 3, pp. 409–409, jul 1965. [Online]. Available: http://epubs.siam.org/doi/10.1137/1007077

[30] D. Choukroun, I. Y. Bar-Itzhack, and Y. Oshman, “Optimal-REQUEST Algorithm for Attitude Determination,” Journal of Guidance, Control, and Dynamics, vol. 27, no. 3, pp. 418–425, 2004.

[31] I. Y. Bar-Itzhack, “REQUEST: A recursive QUEST algorithm for sequential attitude determination,” Journal of Guidance, Control, and Dynamics, vol. 19, no. 5, pp. 428–437, 1996.

[32] Magnetic Detumbling of a Rigid Spacecraft by Giulio Avanzini and Fabrizio Giulietti

[33] Attitude Control System of UWE-3 by Florian Reichel.

[34] Design of Attitude Control System of a Double CubeSat. Gaute Bråthen. https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/260799/617039_FULLTEXT01.pdf?sequence=1&isAllowed=y

[35] K. L. Jenssen and K. H. Yabar, “Development, implementation and testing of two attitude estimation methods for cube satellites,” Master’s thesis, Department of Engineering Cybernetics (ITK), Norwegian University of Science and Technology (NTNU), Trondheim, Norway, 2011.

[36] STMicroelectronics: https://www.st.com/content/st_com/en.html

[37] STMicroelectronics Sustainability report 2023 - Home

[38] Mitigating space debris generation. ESA. https://www.esa.int/Space_Safety/Space_Debris/Mitigating_space_debris_generation#:

ANNEX

Annex 1: Spacecraft configuration

```
%-------------------------------------------------------------------------
% Default data structure for the PocketQube
%-------------------------------------------------------------------------

function d = DefaultStruct
% Code that initialize the parameters of the Satellite structure.
% This code is made with theoretical values for the stowed version
% without displacement in the CM.
%-------------------------------------------------------------------------

d                       = struct;
d.jD0                   = Date2JD(2010);

% Dimensions of the satellite X Y Z[m]
%-------------------------------------------------------------------------
d.dimensions = [0.05, 0.05, 0.05];
d.side= 0.05;

% Mass of the PocketQube [kg]
%-------------------------------------------------------------------------
d.mass                  = 0.250;

% Location from geometric center of spacecraft
%-------------------------------------------------------------------------
r = diag([d.dimensions(1) d.dimensions(2) d.dimensions(3)]/2)*eye(3);
d.r = [r -r];

% Initial magnetic dipole moment
%-------------------------------------------------------------------------
d.dipole                = [0;0;0];

% Power for a solar power system.
%-------------------------------------------------------------------------
d.power                 = SolarCellPower;
d.power.consumption     = 0.5;
d.power.batteryCapacity = 100;

% Aerodynamic model for a PocketQube
%-------------------------------------------------------------------------
d.surfData              = CubeSatAero;
d.surfData.att.type     = 'eci';
d.aeroModel             = @CubeSatAero;
```

```
% Distance from the center of masses to the geometrical center [m]
%-------------------------------------------------------------------------
d.CM = [0, 0, 0];

% Inertia of the PocketQube
%-------------------------------------------------------------------------
d.inertia=(d.mass*d.side*d.side/6)*eye(3);

% Normals of the faces of the satellite [unitary vectors]
%-------------------------------------------------------------------------
d.normals      = [1,0,0,-1,0,0; ...
                  0,1,0,0,-1,0; ...
                  0,0,1,0,0,-1];

% Area of the PocketQube
%-------------------------------------------------------------------------
d.area = [ d.dimensions(3)*d.dimensions(2),... % X+
           d.dimensions(1)*d.dimensions(3),... % Y+
           d.dimensions(1)*d.dimensions(2),... % Z+
           d.dimensions(3)*d.dimensions(2),... % X-
           d.dimensions(1)*d.dimensions(3),... % Y-
           d.dimensions(1)*d.dimensions(2)];   % Z- We have the camara


% Location from geometric center of spacecraft
%-------------------------------------------------------------------------
r = diag([d.dimensions(1) d.dimensions(2) d.dimensions(3)]/2)*eye(3);
d.r = [r -r];

% Initial magnetic dipole moment
%-------------------------------------------------------------------------
d.dipole                = [0;0;0];

% Power for a solar power system.
%-------------------------------------------------------------------------
d.power                 = SolarCellPower;
d.power.consumption     = 0.5;
d.power.batteryCapacity = 100;

% Aerodynamic model for a PocketQube
%-------------------------------------------------------------------------
d.surfData              = CubeSatAero;
d.surfData.att.type     = 'eci';
d.aeroModel             = @CubeSatAero;

% Radiation pressure model for a PocketQube around the Earth
%-------------------------------------------------------------------------
d.opticalModel          = @CubeSatRadiationPressure;
d.surfData              = CubeSatRadiationPressure(d.surfData);
[aP, f, fHat, fHat400]  = SolarFluxPrediction( d.jD0, 'nominal' );
d.atm.aP                = aP(1);
d.atm.f                 = f(1);
d.atm.fHat              = fHat(1);
d.atm.fHat400           = fHat400(1);

% In case there are reaction wheels add information here
%-------------------------------------------------------------------------
d.kWheels               = [];
d.inertiaRWA            = [];
d.tRWA                  = [];
```

Annex 2: QPose.m

```
function qp = QPose( q )

%-------------------------------------------------------------------------
%   This function compute the transpose of a quaternion.
%
%-------------------------------------------------------------------------
%   Form:
%   qp = QPose( q )
%-------------------------------------------------------------------------
%   Inputs
%-------------------------------------------------------------------------
%   q                   (4,:)  quaternion from a to b
%-------------------------------------------------------------------------
%   Outputs
%-------------------------------------------------------------------------
%   qp                  (4,:)  quaternion from b to a
%-------------------------------------------------------------------------
%   Copyright (c) 1993 Princeton Satellite Systems, Inc.
%-------------------------------------------------------------------------
qp(1,:)   =   q(1,:);
qp(2:4,:) = - q(2:4,:);
```

Annex 3: QMult.m

```
function Q3 = QMult( Q2 ,Q1 )

%-------------------------------------------------------------------------
%   Multiply two quaternions.
%   Q2 transforms from A to B and Q1 transforms from B to C
%   so Q3 transforms from A to C. If only one argument is input
%   it produces the 4x4 matrix equivalent:
%   q3 = Q2*q1;
%-------------------------------------------------------------------------
%   Form:
%   Q3 = QMult( Q2 ,Q1 )
%-------------------------------------------------------------------------
%   Inputs
%-------------------------------------------------------------------------
%   Q2              (4,1)  Quaternion from a to b
%   Q1              (4,1)  Quaternion from b to a
%-------------------------------------------------------------------------
%   Outputs
%-------------------------------------------------------------------------
%   Q3              (4,1)  Quaternion from a to c or 4x4 version of q
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%   Copyright (c) 1993 Princeton Satellite Systems, Inc.
%-------------------------------------------------------------------------

if( nargin == 2 )

  Q3 = [Q1(1,:).*Q2(1,:) - Q1(2,:).*Q2(2,:) - Q1(3,:).*Q2(3,:) - Q1(4,:).*Q2(4,:);...
        Q1(2,:).*Q2(1,:) + Q1(1,:).*Q2(2,:) - Q1(4,:).*Q2(3,:) + Q1(3,:).*Q2(4,:);...
        Q1(3,:).*Q2(1,:) + Q1(4,:).*Q2(2,:) + Q1(1,:).*Q2(3,:) - Q1(2,:).*Q2(4,:);...
        Q1(4,:).*Q2(1,:) - Q1(3,:).*Q2(2,:) + Q1(2,:).*Q2(3,:) + Q1(1,:).*Q2(4,:)];

else

  Q3 = [Q2(1), -Q2(2), -Q2(3), -Q2(4);...
        Q2(2),  Q2(1), -Q2(4),  Q2(3);...
        Q2(3),  Q2(4),  Q2(1), -Q2(2);...
        Q2(4), -Q2(3),  Q2(2),  Q2(1)];

end
```

Annex 4: Orbit Propagator

```
%% Orbit Propagator
%--------------------------------------------------------------------------
% This code initialize the parameters of the orbit that will be simulated.
%--------------------------------------------------------------------------

% Starting date of the simulation
Year   = 2023;
Month  = 1;                % [0-12]
Day    = 1;                % [0-31]
Hour   = 0;                % [0-24]
Minute = 0;                % [0-60]
Second = 0;                % [0-60]
Orbit.Date = [Year Month Day Hour Minute Second];%Building the date vector

refl = load('Reflectivity_Data.mat'); % Albedo Computation (using Toolbox)

% TLE of the orbit to simulate
% TLE Guidelines:
]%{
    LINE 1:
        Col 1:     Line Number
        Col 3-7:   Satellite Number: 5 digit number assigned by NORAD in
                                     its Satellite Catalog
        Col 8:     Classification letter: U=Unclassified, C=Classified,
                                          S=Secret
        Col 10-11: Last two digits of launch year
        Col 12-14: Number of launch of that year
        Col 15-17: Piece of the launch
        Col 19-20: Last two digits of the current year
        Col 21-32: Epoch: Day of the year > xxx/365.
                          Portion of the day > hour/24 = xxxxxxxx
        Col 34-43: First time derivative of the mean motion [rev/day^2]
        Col 45-52: Second time derivative of the mean motion: [rev/day^3]
        Col 54-61: BSTAR (B*) Friction coefficient from aerodynamic
                              disturbance
        Col 63:    Orbital model used to calculate elements: Currently 0
        Col 65-68: Element set number. Incremented when a new TLE
                   is generated for the object
        Col 69:    Checksum (Module 10): The sum of all the values of the
        line ignoring the blank spaces, the letters and the "+" signs, and
        assigning a value of -1 to all the "-" signs. The checksum is the
        last digit of the sum.
    LINE 2:
        Col 1:     Line Number
        Col 3-7:   Satellite Number: 5 digit number assigned by NORAD
                   in its Satellite Catalog
        Col 9-16:  Inclination i [degrees]
        Col 18-25: Right Ascension of the Ascending Node RAAN [degrees]
        Col 27-33: Eccentricity e (only decimal part)
        Col 35-42: Argument of the perigee w [degrees]
        Col 44-51: Mean Anomaly M [degrees]
```

```
        Col 53-63: Mean Motion n [revs/day]
        Col 64-68: Number of revolution from the Epoch [revs]
        Col 69:    Checksum (Module 10): Same procedure as Col 69 of Line 1
%}

% First line of the TLE
%-------------------------------------------------------------------------
L1 = '1 00001U 16001A   19000.00000000  .00000746  00000-0  16862-3 0  9993';

% Second line of the TLE
%-------------------------------------------------------------------------
L2 = '2 00001  90.0000 000.0000 0000000  00.0000 000.0000 15.21694435 00000';

% Second line of the TLE (Changed to get an orbit of h ~= 630km)
%-------------------------------------------------------------------------
%L2 = '2 00001  90.0000 000.0000 0000000  00.0000 000.0000 14.81000000 00000';

Orbit.TLE = [L1; L2];

% Orbital revolutions and delta time
%-------------------------------------------------------------------------
Orbit.Simulation_Revs = 1;  % Number of orbital revolutions to simulate
Orbit.dt = 0.1;             % Delta time to generate the orbital points
                            % to simulate [s]

% Computing the number of orbital points to simulate (N)
%-------------------------------------------------------------------------
Orbit.MeanMotion = str2double(L2(1, 53 : 63));      % n[revs/day]
Period = 1/Orbit.MeanMotion;                        % t[days]
Period = Period*(24*60*60);                         % t[seconds]
% Number of points to simulate
Orbit.Npoints = floor((Period*Orbit.Simulation_Revs)/Orbit.dt);

% PROPAGATION
%-------------------------------------------------------------------------

% Date of simulation: Calculation of Epoch date into an array
%-------------------------------------------------------------------------
Epoch_Date = fun_EpochDate(Orbit.TLE);

Orbit.Propagation_tSinceEpoch = fun_tSinceEpoch(Epoch_Date, Orbit.Date,...
    Orbit.dt, Orbit.Npoints);  % Time elapsed since the TLE Epoch [s]
Propagation_tJulian = fun_JulianDate(Epoch_Date,...
    Orbit.Propagation_tSinceEpoch); % Julian Date [days]
Propagation_GST = gstime(Propagation_tJulian); %Greenwich Sideral Time[rad]

% Position and velocity
%-------------------------------------------------------------------------

Propagation_ECIPos = zeros(3, Orbit.Npoints);
Propagation_ECIVel = zeros(3, Orbit.Npoints);
altitude = zeros(1,Orbit.Npoints);
```

```
velocity = zeros(1,Orbit.Npoints);
% Satellite info prepared fot the third party tools
Satellite_Info = twoline2rvMOD(Orbit.TLE(1,:), Orbit.TLE(2,:));

% Results of the arrays calculated in this loop are in km and km/s
for i = 1 : Orbit.Npoints
    [Satellite_Info, Propagation_ECIPos(:, i), Propagation_ECIVel(:, i)]...
        = sgp4(Satellite_Info, Orbit.Propagation_tSinceEpoch(i)/60);
    altitude(1,i) = norm(Propagation_ECIPos(:, i));
    velocity(1,i) = norm(Propagation_ECIVel(:,i));
end
[Propagation_ECEFPos, Propagation_ECEFVel] = fun_ECI2ECEF...
    (Propagation_GST, Propagation_ECIPos * 1000,...
    Propagation_ECIVel * 1000); % [m] & [m/s]
Orbit.LatLongHeight = ecef2lla(Propagation_ECEFPos.').';


% Memory allocation of the previous calculations
%--------------------------------------------------------------------------
% Position vector in ECI frame [m]
Orbit.Propagation_ECIPos = Propagation_ECIPos * 1000;
% Velocity vector in ECI frame [m/s]
Orbit.Propagation_ECIVel = Propagation_ECIVel * 1000;
% Position vector in ECEF frame [m]
Orbit.Propagation_ECEFPos = Propagation_ECEFPos;
% Velocity vector in ECEF frame [m/s]
Orbit.Propagation_ECEFVel = Propagation_ECEFVel;
% Angular rate vector of the orbit in ECI frame [rad/s]
Orbit.Propagation_OrbitW = fun_PerifocalAngRate...
    (Orbit.Propagation_ECIPos,Orbit.Propagation_ECIVel);
Orbit.Altitude = altitude*1000;
Orbit.Velocity = velocity*1000;

% Orbit enviroment
--------------------------------------------------------------------------

SolarActivity = 'Nominal';

%{
    The solar activity index will be automatically selected as a function
    of the date of the simulation, but for each date there is a high
    estimation, a low estimation and a nominal estimation.

    Select between: 'High','Low' and 'Nominal' and the program will
    automatically assign the correct F107 and F107A values.

    When selectiong 'Nominal' or 'Low', the Ap magnetic index is computed
    via a quadratic relation [Ap = 3.9312e-04*F107^2 + 0.1194*F107 -
    9.4226].

    When selecting 'High', it considers a short-term high solar activity
    and the Ap index is set to 240 (ISO 14222, 2013) and it is independent
```

```
    from the F107.
%}

[density_input] = fun_nrlmsiseinput(Orbit.Npoints, SolarActivity,...
    Propagation_tJulian, Orbit.LatLongHeight);

% Density of the atmosphere in the orbit [kg/m^3]
Orbit.Propagation_density = fun_density(density_input);
% Magetic field vector expresed in the ECI frame [T]
Orbit.Propagation_ECIMag = fun_ECIMag(Orbit.Date,...
    Orbit.LatLongHeight, Propagation_GST);
% Solar irradiance without noise [W/m^2]
Orbit.Propagation_EAM0Theory = ...
    fun_SolarIrrad(Orbit.Date, Orbit.Propagation_tSinceEpoch);
% Solar irradiance with an additive gaussian random noise [W/m^2]
Orbit.Propagation_EAM0 = Orbit.Propagation_EAM0Theory +...
    normrnd(0, 5/3, Orbit.Npoints, 1);
% Sun vector in ECI frame [au]
Orbit.Propagation_ECISunVec = sunPosition(Propagation_tJulian);

% Sun vector to ECEF
[Propagation_ECEFSunVec, Non_Use] = fun_ECI2ECEF(Propagation_GST,...
    Orbit.Propagation_ECISunVec * 1.496e11, [0;0;0]); % [m] & [m/s]
Orbit.Propagation_ECEFSunVec = Propagation_ECEFSunVec;

% Moon vector in ECI frame
Orbit.Propagation_ECIMoonVec = moonpos(Propagation_tJulian,...
    'jd', 'xyz', 'm')';
% Random Eclipse info
Orbit.Propagation_Eclipse = Eclipse_ECSM(Orbit.Propagation_ECIPos,...
    Orbit.Propagation_ECISunVec);


% Saving and clearing
%--------------------------------------------------------------------------
save("../Data/Orbit_Theory", 'Orbit')
clear
```

Annex 5: BDipole.m

```
function [b, bDot] = BDipole( r, jD, v )
```

```
%% Computes the geocentric magnetic field based on a tilted dipole model.
% The output is in geocentric inertial coordinates (ECI). This function
% includes the effect of dipole motion on the earth.
%
% Type BDipole for a demo of a satellite in a low earth inclined orbit.
%-------------------------------------------------------------------------
%   Form:
%   [b, bDot] = BDipole( r, jD, v )
%-------------------------------------------------------------------------
%   Inputs
%-------------------------------------------------------------------------
%   r           (3,:)   Position vector in the ECI frame
%   jD          (1,:)   Julian days
%   v           (3,:)   Velocity
%
%-------------------------------------------------------------------------
%   Outputs
%-------------------------------------------------------------------------
%   b           (3,:)   Magnetic field in the ECI frame (T)
%   bDot        (3,:)   Derivative of b in the ECI frame (T/s)
%-------------------------------------------------------------------------
%   Reference:  Wertz, J., ed. "Spacecraft Attitude Determination and
%               Control," Kluwer, 1976, 783-784.
%
%   Includes 1995 IGRF coefficients as of Jan. 1999
%-------------------------------------------------------------------------

%-------------------------------------------------------------------------
%   Copyright (c) 1995-1999, 2014 Princeton Satellite Systems, Inc.
%-------------------------------------------------------------------------

% Demo
%-------------------------------------------------------------------------
if( nargin < 2 )
    a       = 7000;
    el      = [a 55*pi/180 0 0 0 0];
    p       = Period(a);
    t       = linspace(0,p,1000);
    [r, v]  = RVFromKepler(el,t);
    jD      = JD2000 + t/86400;
    BDipole( r, jD, v );
    return;
end
```

```
jD1995     = -1826.5;
jDFrom2000 = jD - JD2000;

dJD        = (jDFrom2000(1)-jD1995)/365.25;

g10        = -29652   + 22.4*dJD;
g11        =  -1787.5 + 11.3*dJD;
h11        =   5367.5 - 15.9*dJD;

h0         = sqrt( h11^2 + g11^2 + g10^2 );
a          = 6371.2;

cosThetaM = g10/h0;
thetaM    = acos( cosThetaM );
sinThetaM = sin( thetaM );
phiM      = atan2( h11, g11 );
uDipoleEF = [sinThetaM*cos( phiM ); sinThetaM*sin( phiM ); cosThetaM];

aCuH       = a^3*h0*1.e-9;
n          = length(jD);
b          = zeros(3,n);

% First path includes the derivative
%-------------------------------------------------------------------------
if( nargin > 2  )
    bDot       = zeros(3,n);
    for k = 1:n
        cECIToEF    = ECIToEF( JD2T( jD(k) ) );
        rEF         = cECIToEF*r(:,k);
        rMag        = Mag( rEF );
        uR          = rEF/rMag;
        bEF         = (aCuH/rMag^3)*(3*(uDipoleEF'*uR)*uR-uDipoleEF);
        b(:,k)      = cECIToEF'*bEF;
        bDot(:,k)   = DerivativeOfB( bEF, rEF, r(:,k), rMag, v(:,k),...
            aCuH, uDipoleEF, cECIToEF', jD(k) );
    end
else
    for k = 1:n
        cECIToEF    = ECIToEF( JD2T( jD(k) ) );
        rEF         = cECIToEF*r(:,k);
        rMag        = Mag( rEF );
        uR          = rEF/rMag;
        bEF         = (aCuH/rMag^3)*(3*(uDipoleEF'*uR)*uR-uDipoleEF);
        b(:,k)      = cECIToEF'*bEF;
    end
end
```

```
% Plotting
%-------------------------------------------------------------------------
if( nargout == 0 )
    Plot2D(jDFrom2000, r, 'Time from J2000', {'r_x (km)' 'r_y (km)'...
        'r_z (km)'}, 'ECI Position')
    if( nargin < 3 )
        Plot2D(jDFrom2000, b*1.e9,'Time from J2000',{'B (nT)'},...
            'Dipole Field');
    else
        Plot2D(jDFrom2000, [b;bDot]*1.e9,...
            'Time from J2000',{'B (nT)' 'dB/dt (nT/s)'},...
            'Dipole Field','lin',{'[1 2 3]' '[4 5 6]'});
    end
    legend('x','y','z')
    clear b
end

%-------------------------------------------------------------------------
%   Time derivative of b
%-------------------------------------------------------------------------
function bDot = DerivativeOfB( bEF, rEF, r, rho, v, a, m, c, jD )

uEF     = rEF/rho;

rhoDot  = r'*v/rho;

omega   = [0;0;EarthRte(jD)];
cDot      = c*Skew(omega);

uEFDot  = (c'*v + cDot'*r)/rho - rEF*rhoDot/rho^2;

bEFDot  = -3*a*rhoDot*(3*(m'*uEF)*uEF - m)/rho^4 ...
        +  3*a*((m'*uEFDot)*uEF + (m'*uEF)*uEFDot)/rho^3;

bDot    = c*bEFDot + cDot*bEF;
```

Annex 6: sunPosition.m

```
function [rsun] = sunPosition(jd)

%% SUN POSITION PROPAGATION ALGORITHM
%  This function calculates the geocentric equatorial position vector of
%  the sun in AU's given the julian date.  This is the low precision
%  formula and is valid for years from 1950 to 2050.  Accuaracy of apparent
%  coordinates is 0.01  degrees.
%-----------------------------------------------------------------------
%  Author        : david vallado                  719-573-2600   27 may 2002
%-----------------------------------------------------------------------
%  Inputs
%-----------------------------------------------------------------------
%    jd          - julian date                    days from 4713 bc
%-----------------------------------------------------------------------
%  Outputs
%-----------------------------------------------------------------------
%    rsun        - ijk position vector of the sun au
%    rtasc       - right ascension                 rad
%    decl        - declination                     rad
%-----------------------------------------------------------------------
%  Locals
%-----------------------------------------------------------------------
%    meanlong    - mean longitude
%    meananomaly - mean anomaly
%    eclplong    - ecliptic longitude

%    obliquity   - mean obliquity of the ecliptic
%    tut1        - julian centuries of ut1 from
%                  jan 1, 2000 12h
%    ttdb        - julian centuries of tdb from
%                  jan 1, 2000 12h
%    hr          - hours                          0 .. 24              10
%    min         - minutes                        0 .. 59              15
%    sec         - seconds                        0.0  .. 59.99     30.00
%    temp        - temporary variable
%    deg         - degrees
%-----------------------------------------------------------------------
%  REFERENCES    :
%    vallado       2007, 281, alg 29, ex 5-1
%
%    "D. A. Vallado and W. McClain, Fundamentals of Astrodynamics and
%     Applications, 4th editio. Hawthorne, CA: Microcosm Press, 2013".
%
% [rsun,rtasc,decl] = sun ( jd );
%
% ----------------------------------------------------------------------

    twopi      =     2.0*pi;
    deg2rad    =     pi/180.0;
```

```
    npts = length( jd );

    rsun = zeros(3,npts);

    % --------------------  Implementation   ----------------------
    % ------------------  Initialize values   --------------------
    tut1 = ( jd - 2451545.0  )/ 36525.0;

    meanlong = 280.460  + 36000.77*tut1;
    meanlong = rem( meanlong,360.0  ); % deg

    ttdb = tut1;
    meananomaly = 357.5277233  + 35999.05034 *ttdb;
    meananomaly = rem( meananomaly*deg2rad,twopi ); % rad
    if ( meananomaly < 0.0  )
        meananomaly = twopi + meananomaly;
    end

    eclplong = meanlong + 1.914666471 *sin(meananomaly) ...
                + 0.019994643 *sin(2.0 *meananomaly); % deg
    eclplong = rem( eclplong,360.0  ); % deg

    obliquity = 23.439291  - 0.0130042 *ttdb; % deg

    eclplong = eclplong*deg2rad;
    obliquity = obliquity*deg2rad;

    eclplong = eclplong*deg2rad;
    obliquity = obliquity*deg2rad;

    % --------- Find magnitude of sun vector, )   components ------
    magr = 1.000140612  - 0.016708617 *cos( meananomaly ) ...
                          - 0.000139589 *cos( 2.0 *meananomaly ); % in au's

    rsun(1,:) = magr.*cos( eclplong );
    rsun(2,:) = magr.*cos(obliquity).*sin(eclplong);
    rsun(3,:) = magr.*sin(obliquity).*sin(eclplong);


end
```

Annex 7: fun_density.m

```
function [rho] = fun_density(input)

%-------------------------------------------------------------------------
% Function that computes the density for an array of structs input.
%-------------------------------------------------------------------------
% Inputs:
%-------------------------------------------------------------------------
%      - input: array of structs containing all the necessary inputs for
%               each simulated orbital point.
%-------------------------------------------------------------------------
% Outputs:
%-------------------------------------------------------------------------
%       - rho: array containing the density of each orbital point
%-------------------------------------------------------------------------

rho = zeros(1,length(input));

for i = 1:length(input)
    rho(1,i) = nrlmsise00mod(input(i));
end
```

Annex 8: GravityGradientFromR.m

```
function t = GravityGradientFromR( q, inr, r, mu )

%------------------------------------------------------------------------
%   Computes a gravity gradient torque from r and mu.
%   q transforms from the frame in which r is defined to the body frame.
%
%   Since version 10.
%------------------------------------------------------------------------
%   Form:
%   t = GravityGradientFromR( q, inr, r, mu )
%------------------------------------------------------------------------
%
%------------------------------------------------------------------------
%   Inputs
%------------------------------------------------------------------------
%   q                   (4,1) Quaternion from the r frame to Body
%   inr                 (3,3) Inertia matrix
%   r                   (3,:) Orbit from the earth to the spacecraft
%                             in the spacecraft body frame (the same frame
%                             as the inertia matrix.)
%   mu                  (1,1) Gravitational Parameter
%
%------------------------------------------------------------------------
%   Outputs
%------------------------------------------------------------------------
%   t                   (3,:) Torque (units will be consistent with inertia)
%
%------------------------------------------------------------------------

%------------------------------------------------------------------------
%   Copyright (c) 2011 Princeton Satellite Systems, Inc.
%   All rights reserved.
%------------------------------------------------------------------------

% Demo
%------------------------------------------------------------------------
if( nargin == 0 )
  mu  = Constant('mu earth');
  inr = diag([100;20;300]);
  n   = 1000;
  a   = linspace(0,6*pi,n);
  r   = 7000*[cos(a);sin(a);zeros(1,n)];
  q   = [ones(1,n);zeros(3,n)];
  GravityGradientFromR( q, inr, r, mu );
  return;
end
```

```
n = size(r,2);
t = zeros(3,n);

for k = 1:n
    rK     = QForm( q(:,k), r(:,k) );
    t(:,k) = 3*mu*Cross(rK,inr*rK)/Mag(rK)^5;
end

% Default output
%-------------------------------------------------------------------------
if( nargout == 0 )
  Plot2D( 1:n, t, 'Step', {'T_x' 'T_y' 'T_z'}, 'Gravity Gradient' );
  clear t
end
```

Annex 9: CubeSatAero.m

```
function [force, torque] = CubeSatAero( p, d )
```

```
%% Aerodynamic model for a CubeSat.
% Contains a built-in demo for a 1U PocketQube in a 6500 km orbit.
% Also contains an option to retrieve the default data structure.
%
% The attitude model in CubeSatAttitude will be used if p.q is empty.
%-------------------------------------------------------------------------
%   Form:
%   [force, torque] = CubeSatAero( p, d )
%   d = CubeSatAero     % data structure
%   CubeSatAero         % demo for LEO orbit with LVLH alignment
%-------------------------------------------------------------------------
%
%-------------------------------------------------------------------------
%   Inputs
%-------------------------------------------------------------------------
%   p         (.)    Profile data structure
%                    .r    (3,1) ECI position
%                    .v    (3,1) ECI velocity
%                    .q    (4,1) ECI to body quaternion
%                    .vRel (3,1) ECI velocity relative to the earth (km/s)
%                    .rho  (1,1) Atmospheric density (kg/m^3)
%   d         (.)    Aero data structure
%                    .cD         (1,1) Drag coefficient
%                    .nFace      (3,:) Face outward unit normals
%                    .rFace      (3,:) Face vectors with respect to the
%                                      origin
%                    .cM         (3,1) Center of mass with respect to the
%                                      origin
%                    .area       (1,:) Area of each face (m^2)
%                    .att         (.)  Attitude data structure
%
%-------------------------------------------------------------------------
%   Outputs
%-------------------------------------------------------------------------
%   force    (3,1)   ECI force (N)
%   torque   (3,1)   Body fixed torque (Nm)
%
%-------------------------------------------------------------------------
% See also CubeSatFaces, CubeSatAttitude
```

```
%-------------------------------------------------------------------------
%   Copyright (c) 2009, 2014, 2016 Princeton Satellite Systems, Inc.
%-------------------------------------------------------------------------
%   Since version 8.
%   2014-04-21: Update demo to use RVFromKepler for the orbit.
%   2016.1: Fix x axis of demo plot; use attitude in p if available
%-------------------------------------------------------------------------

% Demo
%-------------------------------------------------------------------------
if( nargin < 1 )
  d = DefaultStruct;
  if nargout == 1
    force = d;
    return;
  end
  % Demo in LEO with LVLH pointing (the default)
  %-----------------------------------------------------------------------
  [r,v,t] = RVFromKepler([6500 0 0 0 0 0]);

  % Set the atmospheric density (kg/m3)
  %-----------------------------------------------------------------------
  p.rho  = 1.6e-9;
  p.q    = [];

  force  = zeros(3,100);
  for k = 1:100
    p.v    = v(:,k);
    p.r    = r(:,k);
    p.vRel = p.v;
    force(:,k) = CubeSatAero( p, d );
  end

  [tPlot, cTime] = TimeLabl( t );
  Plot2D( tPlot, force*1e6, cTime, {'F x (\mu N)' 'F y(\mu N)'...
```

```
      'F_z(\mu N)'}, 'Cubesat Aero Force');
  clear force
  return
end

% The vector dynamic pressure in the ECI frame
%-------------------------------------------------------------------------
vMag = Mag(p.vRel);
qAero = 0.5*p.rho*vMag*p.vRel*1e6;

if( isempty(p.q) )
    qECIToBody = CubeSatAttitude( d.att, p.r, p.v );
else
  qECIToBody = p.q;
end

qBody      = QForm( qECIToBody, qAero );
uBody      = QForm( qECIToBody, p.vRel/vMag);

force      = zeros(3,1);
torque     = zeros(3,1);

% For each face, compute in the body frame
% Only faces towards the velocity vector contribute.
nDot = Dot(uBody,d.nFace);
kFace = find(nDot>0);
for k = kFace
  fFace  = - nDot(k)*qBody*d.area(k)*d.cD;
  force  = force  + fFace;
  torque = torque + Cross( d.rFace(:,k) - d.cM, fFace );
end

% Force is in the ECI frame
%-------------------------------------------------------------------------
force = QTForm( qECIToBody, force );

function d = DefaultStruct
% Default data for a 1U PocketQube

d.cD              = 2.7;
d.cM              = [0;0;0];
model = [1 1 1];
[a,n,r]           = CubeSatFaces( model,1 );
d.area            = a;
d.nFace           = n;
d.rFace           = r;
d.att.type        = 'lvlh';
d.att.qLVLHToBody = [1;0;0;0];
```

Annex 10: CubeSatRadiationPressure.m

```
function [force, torque] = CubeSatRadiationPressure( p, d )
```

```
%% Radiation pressure model for a PocketQube around the Earth.
%
% Includes solar pressure, earth radiation pressure, and earth albedo.
% Has a built-in demo. Retrieve a default data structure
% by calling with no inputs, or append the additional fields to an
% existing structure. Set d.planet to 0 (false) to ignore planetary
% disturbances and calculate only solar pressure.
%--------------------------------------------------------------------------
%   Forms:
%   [force, torque] = CubeSatRadiationPressure( p, d )
%   d = CubeSatRadiationPressure       % default d
%   d = CubeSatRadiationPressure( d )  % add needed fields to d
%   CubeSatRadiationPressure           %
%--------------------------------------------------------------------------
%
%--------------------------------------------------------------------------
%   Inputs
%--------------------------------------------------------------------------
%   p         (.)    Data structure from CubeSatEnvironment
%                    .r    (3,1) ECI position
%                    .v    (3,1) ECI velocity
%                    .q    (4,1) ECI to body quaternion
%                    .uSun (3,1) ECI sun unit vector
%                    .solarFlux (1,1) Solar flux at position (W/m2)
%                    .nEcl      (1,1) Normalized source intensity from
%                                     Eclipse
%                    .radiation (1,1) Planet radiation
%                    .albedo    (1,1) Planet albedo fraction
%                    .radiusPlanet (1,1) Planet radius (km)
%   d         (.)    Surface data structure
%                    .nFace      (3,:) Face outward unit normals
%                    .rFace      (3,:) Face vectors with respect to the
%                                      origin
%                    .cM         (3,1) Center of mass with respect to the
%                                      origin
%                    .area       (1,:) Area of each face
%                    .sigma      (3,:) Radiation coefficients [absorbed;
%                                      specular; diffuse]
%                    .planet     (1,1) Flag for planetary contributions
%                    .att         (.)  Attitude data structure
%
%--------------------------------------------------------------------------
%   Outputs
%--------------------------------------------------------------------------
%   force    (3,1)   ECI force (N)
%   torque   (3,1)   Body fixed torque (Nm)
%
%--------------------------------------------------------------------------
```

```
%--------------------------------------------------------------------------
% See also SolarF, CubeSatAttitude
%--------------------------------------------------------------------------
%   Copyright (c) 2009, 2016 Princeton Satellite Systems, Inc.
%--------------------------------------------------------------------------
%   Since version 8 (2009).
%   2016-02-23: Update to use CubeSatEnvironment. Input p now provides the
%   solar flux, eclipse fraction, and planet radius.
%--------------------------------------------------------------------------

% Demo
%--------------------------------------------------------------------------
if( nargin < 1 )
  % The demo plots the disturbance calculated over one orbit. The
  % disturbances are calculated both with and without the planetary
  % components (albedo and radiation).
  %------------------------------------------------------------------------
  d = DefaultStruct;
  if nargout == 1
    force = d;
    return;
  end
  disp('CubeSatRadiationPressure Demo')

  % Demo in LEO with LVLH pointing
  %------------------------------------------------------------------------
  [r,v,t] = RVFromKepler([6500 0 0 0 0 0]);

  % Use a constant environment except eclipses
  %------------------------------------------------------------------------
  p = CubeSatEnvironment;
  nEcl    = Eclipse(r,p.uSun*p.rSun);
  force   = zeros(3,100);
  for k = 1:100
    p.nEcl = nEcl(k);
    p.r    = r(:,k);
    p.v    = v(:,k);
    [force(:,k),torque] = CubeSatRadiationPressure( p, d );
  end
  [tPlot, cTime] = TimeLabl( t );
  h = Plot2D( tPlot, force*1e6, cTime, {'F_x (\mu N)' 'F_y(\mu N)' ...
      'F_z(\mu N)'},'Cubesat Radiation Force');
  AddFillToPlots(tPlot,nEcl,h,[1 1 1; 0.9 0.9 0.9],0.5);
  clear force
  return
elseif (nargin==1)
  d = p;
  d.sigma = [1 1 1 1 1 1;zeros(2,6)];
  d.planet = true;
  force = d;
  return;
end
```

```
% Quaternion
%-------------------------------------------------------------------------
if isempty(p.q)
  qECIToBody = CubeSatAttitude( d.att, p.r, p.v );
else
  qECIToBody = p.q;
end

% Get the force in the body frame
%-------------------------------------------------------------------------
% Solar radiation pressure
%-------------------------------------------------------------------------
uSBody = QForm( qECIToBody, p.uSun );
cLight = 3e8; % speed of light in m/s2
pSolar = p.solarFlux/cLight; % solar pressure
solarF = p.nEcl*SolarF( pSolar, d.sigma, d.nFace, uSBody, d.area );

% Calculate planetary contributions only if requested
%-------------------------------------------------------------------------
if d.planet
  rBody = QForm( qECIToBody, p.r );

  % Source is at nadir
  %-----------------------------------------------------------------------
  uPR   = -Unit( rBody );

  % Scale for the distance from the planet
  %-----------------------------------------------------------------------
  pScale = p.radiusPlanet^2/(rBody'*rBody);

  % Infrared: diffuse is 1
  %-----------------------------------------------------------------------
  sigRad = [0;0;1];
  planetaryF = SolarF( p.radiation/cLight*pScale, sigRad, d.nFace,...
      uPR, d.area );
  albedoF = p.nEcl*SolarF(p.albedo*pSolar*pScale, d.sigma, d.nFace,...
      uPR, d.area );
else
  planetaryF = zeros(3,length(d.area));
  albedoF = zeros(3,length(d.area));
end

force = solarF + planetaryF + albedoF;

% Torque for each face
%-------------------------------------------------------------------------
torques = Cross( d.rFace - repmat(d.cM,1,length(d.area)), force );
torque = sum(torques,2);

% Force is in the ECI frame
%-------------------------------------------------------------------------
force = QTForm( qECIToBody, sum(force,2) );
```

```
%--------------------------------------------------------------------------
% Defaults: an example for 2U
%--------------------------------------------------------------------------
function d = DefaultStruct

d.cM              = [0;0;0];
[a,n,r]           = CubeSatFaces( '1U',1 );
d.area            = a;
d.nFace           = n;
d.rFace           = r;
d.sigma           = [1 1 1 1 1 1;zeros(2,6)];
d.att             = CubeSatAttitude;
d.att.type        = 'lvlh';
d.planet          = true;
```

Annex 11: fun_OPTREQUEST.m

```
%%  OPTIMAL REQUEST ALGORITHM

% This function implements the OPTIMAL REQUEST algorithm for attitude
% determination. It should be noted that in this algorithm we are using as
% indices n = n+1 and n1 = n. For example, what should be K_n+1/n+1 we call
% it K_n_n and what should be K_n+1/n we call it K_n_n1.
%-------------------------------------------------------------------------
%   FORM
%   [q_est, LoopState] = OPTREQUEST(Attitude, MagVec_e, SunVec_e,
%                                   MagRead_b, SunRead_b, w,
%                                   LoopState_prev, i, dt, EclipseBool)
%
% -------------------------------------------------------------------------
%   INPUT
%-------------------------------------------------------------------------
%       - Attitude: Structure in which are defined the physical and
%                   electrical characteristics of the sensors.
%       - MagVec_e: 3x1 double matrix, each column containing the Earth's
%                   magnetic field vector in ECI coordinates that come from
%                   the IGRF12 propagation algorithm.
%       - SunVec_e: 3x1 double matrix, each column containing the Sun
%                   Position vector in ECI coordinates that come from the
%                   SPA propagation algorithm.
%       - MagRead_b: 3x1 double matrix, each column containing the Earth's
%                    magnetic field vector in Body coordinates.
%       - SunRead b: 3x1 double matrix, each column containing the Sun
%                    Position vector in Body coordinates.
%       - w: 3x1 double matrix in which each column represent the Angular
%            velocity.
%       - sensor_weight: Weight of magnetorquers and sun_sensors,
%                        sensor_weight = [magnetometer, sun_sensors].
%       - LoopState_prev: Structure that saves every value computed.
%       - i: Number of the iteration being computed.
%       - dt: Time step between orbital points of the simulation.
%       - EclipseBool: Boolean number that indicates if the Satellite is in
%                      Eclipse (different from zero) or not (zero).
%-------------------------------------------------------------------------
%   OUTPUT
%-------------------------------------------------------------------------
%       - q_est: 4x1 matrix. Quaternion of the position of the satellite in
%                Body frame coordinate.
%       - LoopState: Estructure saving every value used to compute q_est.
%
% -------------------------------------------------------------------------
%
% REFERENCES
%
% This algorithm is explained in detail in: "D. Choukroun, I. Y.
% Bar-Itzhack, and Y. Oshman, □Optimal-REQUEST Algorithm for Attitude
% Determination,□ J. Guid. Control. Dyn., vol. 27, no. 3, pp. 418□425,
% 2008". All the variables are defined following the nomenclature of the
% paper.
```

```
% ------------------------------------------------------------------------
function [q_est, LoopStateK] = fun_OPTREQUEST(Gyro, SunSensor,...
    Magnetometer, MagVec_e, SunVec_e, MagRead_b, SunRead_b, w,...
    LoopState_prev, i, dt, EclipseBool, q)

    %% INPUT PARSING
    %------------------------------------------------------------------------
    if EclipseBool ~= 0
        sensor_weight = [1 0];
    else
        sensor_weight = [1/Magnetometer.var, 1/SunSensor.var];
    end

    gyroscope_var     = Gyro.var;
    magnetometer_var  = Magnetometer.var;
    sunSensor_var     = SunSensor.var;

    % 3x1 double matrix in which each column represent the vector [magnetic
    % field (3,1), sun vector (3,1)] in body frame coordinates, from sensor
    % measurement.
    %------------------------------------------------------------------------
    b = [MagRead_b/norm(MagRead_b), SunRead_b/norm(SunRead_b)];
    b(isnan(b)) = 0;

    % 3x1 double matrix in which each column represent the vector [magnetic
    % field (3,1), sun vector (3,1)] in ECI coordinates, from propagators.
%------------------------------------------------------------------------
r = [MagVec_e/norm(MagVec_e), SunVec_e/norm(SunVec_e)];


%% COMPUTE dK
%------------------------------------------------------------------------
% Compute B
B = 0;
for jj=1:length(sensor_weight)
    B = B + sensor_weight(jj) * b(:,jj) * r(:,jj)';
end
% Compute sigma
sigma = trace(B);

% Compute S
S = B + B';

% Compute z
z = 0;
for jj=1:length(sensor_weight)
    z = z + sensor_weight(jj) * cross(b(:,jj), r(:,jj));
end

% Compute dK_0 (matrix 4x4)
dK n = [ S - sigma * eye(3), z;
```

```
            z'                , sigma ];
```

```
    %% DEFINE V (matrix 3x2)
    %-------------------------------------------------------------------------
    db = [ sqrt(magnetometer_var), sqrt(sunSensor_var);
           sqrt(magnetometer_var), sqrt(sunSensor_var);
           sqrt(magnetometer_var), sqrt(sunSensor_var) ];

    % Compute Bb
    Bb = 0;
    for jj=1:length(sensor_weight)
        Bb = Bb + sensor_weight(jj) * db(:,jj) * r(:,jj)';
    end

    % Compute kappab
    kappab = trace(Bb);

    % Compute Sb
    Sb = Bb' + Bb;

    % Compute zb
    zb = 0;
    for jj=1:length(sensor_weight)
        zb = zb + sensor_weight(jj) * cross(db(:,jj), r(:,jj));
    end
% Compute V (matrix 4x4)
V_n_n = [ Sb-kappab*eye(3), zb;
          zb'             , kappab ];
R_n = cov(V_n_n);
```

```
%% DEFINE W
%-------------------------------------------------------------------------
% Compute Be (matrix 3x3)
Be = [ 0                   -sqrt(gyroscope_var)  sqrt(gyroscope_var);
       sqrt(gyroscope_var)  0                    -sqrt(gyroscope_var);
      -sqrt(gyroscope_var)  sqrt(gyroscope_var)  0                   ];

Be = Be * B;
% Compute kappae
kappae = trace(Be);

% Compute Se
Se = Be + Be';

% Compute ze
matrixBe = Be' - Be;
ze       = [0; 0; 0];
ze(1)    = matrixBe(3, 2);
ze(2)    = matrixBe(1, 3);
ze(3)    = matrixBe(2, 1);
```

```
% Compute W_n (matrix 4x4)
W_n = [ Se - kappae * eye(3), ze;
        ze'                 , kappae ];
W_n = W_n * dt;

% Compute Q_n
Q_n = cov(W_n);

% Compute m
dm_n = sum(sensor_weight);
```

```
%% TIME UPDATE
%------------------------------------------------------------------------
% Rotation matrix due to w (matrix 4x4)
omega = 0.5 .* [ 0     w(3) -w(2)  w(1);
                -w(3)  0     w(1)  w(2);
                 w(2) -w(1)  0     w(3);
                -w(1) -w(2) -w(3)  0    ];

% Transition matrix, eq. (9)
w_norm = norm(w);
c = cos(0.5 * w_norm * dt);
s = sin(0.5 * w_norm * dt);

for j=1:3
        n(j) = w(j)/w_norm;
        n(isnan(n)) = 0;
    end

    phi_n = [ c,       n(3)*s, -n(2)*s, n(1)*s;
             -n(3)*s,  c,       n(1)*s, n(2)*s;
              n(2)*s, -n(1)*s,  c,      n(3)*s;
             -n(1)*s, -n(2)*s, -n(3)*s, c      ];
```

```
    %% INITIAL CONDITIONS
    %------------------------------------------------------------------------
    if i == 1

        % For K_0_0 = dK_0, eq. (65)
        K_n_n   = dK_n;
        normaK  = norm(K_n_n);
        K_n_n   = K_n_n/normaK;

        % For P_0_0 = R_0, eq. (66)
        P_n_n = R_n;

        % For m_0 = dm_0, eq. (67)
        m_n = 1;
```

```
%% AFTER INITIAL ESTIMATION
%--------------------------------------------------------------------
else

    %% RETRIEVE PREVIOUS STATE
    %--------------------------------------------------------------------
    K_n1_n1 = LoopState_prev.K;
    P_n1_n1 = LoopState_prev.P;
    m_n1    = LoopState_prev.m;
    Q_n1    = LoopState_prev.Q;
    phi_n1  = LoopState_prev.F;


    %% TIME UPDATE
    %--------------------------------------------------------------------
    K_n_n1 = phi_n1*K_n1_n1*phi_n1';
    P_n_n1 = phi_n1*P_n1_n1*phi_n1'+ Q_n1;


    %% MEASUREMENT UPDATE
    %--------------------------------------------------------------------
    rho_n = m_n1^2 * trace(P_n_n1)/(m_n1^2*trace(P_n_n1)+dm_n^2*trace(R_n));

    if(SunRead_b==[0;0;0])
        rho_n = 0;
    else

         rho_n = 1;
     end

     % m_n
     m_n     = (1 - rho_n) * m_n1 + rho_n * dm_n;

     % Compute K
     K_n_n   = (1 - rho_n) * K_n_n1 + rho_n * dK_n;
     normaK  = norm(K_n_n);
     K_n_n   = K_n_n/normaK;

     % Compute P
     P_n_n   = ((1 - rho_n)^2 * P_n_n1 + (rho_n)^2 * R_n);


end

%% LOOPSTATE UPDATE
%--------------------------------------------------------------------
% K
LoopStateK.K = K_n_n;
% m_k1
LoopStateK.m = m_n;
% phi
LoopStateK.F = phi_n;
%P
```

```
    LoopStateK.P = P_n_n;
    %Q
    LoopStateK.Q = Q_n;


    %% QUATERNION ESTIMATION
    %--------------------------------------------------------------------------
    % In this section the Quaternion is estimated as the eigenvector
    % corresponding to the column in which is the maximum eigenvalue.

    % Compute lambda_max
    % Using Matlab method --> Ideal
    a = (K_n_n * 10^12);
    b = a - fix(a);
    a = a - b;
    K_n_n = a / 10^12;

    [VV,D] = eig(K_n_n);

    lambda_max = max(max(D));
    lambda = D==lambda_max;
    [row, col] = find(lambda);

    if(length(col) > 1)
        col = col(1);

        disp('IMPOSSIBLE STATE: Two auto vectors have the maximum auto value')
    end

    if EclipseBool ~= 0

        q = fun_QuaternionNormalization(q);

        P = [- q(2), - q(3), - q(4);
               q(1), - q(4),   q(3);
               q(4),   q(1), - q(2);
             - q(3),   q(2),   q(1)];

        q_p = 0.5 * fun_QuaternionNormalization(P) * w;

        q_est = q + q_p;

    else
        q_est = VV(:, col);
        q_est = [q_est(4); -q_est(1:3)];
    end
end
```

Annex 12: DetumblingFunction.m

```
function [moment] = DetumblingFunction(d,p,dT)

%% Detumbing Function for a PocketQube orbit and attitude dynamical model.
%
% Calculates the torque that the magnetometers have to do to detumble the
% satellite.
%
%--------------------------------------------------------------------------
% Authors:
% - Yago Recarey Liste
% - Youssef El Kaisi Rahmoun
% - Jose Faneca Ruedas
%--------------------------------------------------------------------------
%   Inputs
%--------------------------------------------------------------------------

%   d           (.)     Data structure
%                       .jD0      (1,1) Julian date of epoch
%                       .mass     (1,1) Spacecraft mass (kg)
%                       .inertia  (3,3) Inertia matrix (kg-m2)
%                       .dipole   (3,1) Residual dipole (ATM^2)
%                       .power     (.)  Power data, see SolarCellPower
%                       .aeroModel    * Handle, see CubeSatAero
%                       .opticalModel * Handle, see CubeSatRadiationPressure
%                       .surfData  (.)  optional;
%                                       .cD    (3,1) Drag coefficient
%                                       .cM    (3,1) Center of mass (m)
%                                       .area  (1,n) Area (m2)
%                                       .nFace (3,n) Face normals
%                                       .rFace (3,n) Face locations (m)
%                                       .att    (.)  Attitude model
%                                       .sigma (3,n) Optical coefficients
%                                       .planet (1)  Planet effects flag
%                       .atm       (.) optional;
%                       .kWheels         (n), empty if no wheels,
```

```
%                       .inertiaRWA     (1,1), optional,polar inertia(kg-m2)
%                       .tRWA           (3,1), optional,wheel torque(Nm)
%   p           (.)     Data structure
%                       .a                      (2,2) Plant matrix
%                       .b                      (2,1) Input matrix
%                       .c                      (1,2) Output matrix
%                       .d                      (1,1) Feedthrough matrix
%                       .desired_orbit_vector   (3,1) Axis in orbit frame
%                       .eci_vector             (3,1) Axis in ECI frame
%                       .bFieldBody             (3,1) Field in Body Frame
%                       .bFieldBody_before      (3,1) Field in Body Frame
%   dT          (1,1)    Step
%
%-------------------------------------------------------------------------
%   Outputs
%-------------------------------------------------------------------------
%   moment      (3,1) Dipole moment generate by the magnetorquers
%-------------------------------------------------------------------------


% Constant for the detumbling
%-------------------------------------------------------------------------
kd=1e6;

% Maximum Dipole moment generated by magnetorquers
m_max = 0.0069;

%DETUMBLING
%-------------------------------------------------------------------------
Bdot              = (p.bFieldBody - p.bFieldBody_before)./dT;
p.bFieldBody_before  = p.bFieldBody;


% Control system momentum management
%-------------------------------------------------------------------------
 moment     = -kd*diag(d.inertia).*Bdot; % Amp-turns m^2

if  max(abs(moment)) > m_max
    disp('pre');
d.dipole = moment*m_max/max(abs(moment));

end
```

Annex 13: PD3Axis.m

```
function [moment] = PD3Axis( q_ECI_body, d, bf, anglevel, alfa, beta)

%% 3 axis PD controller for rigid body.
% The main objective of this controller is to allign the orbit frame with
% the body frame and bring the angular velocity to almost cero.
%-------------------------------------------------------------------------
% Authors:
% - Yago Recarey Liste
% - Youssef El Kaisi Rahmoun
% - Jose Faneca Ruedas
%-------------------------------------------------------------------------
% Inputs
%-------------------------------------------------------------------------
%   q_ECI_body (4,1) ECI to body quaternion
%   d          (1,1) Data structure
%                    .a                        (2,2) Plant matrix
%                    .b                        (2,1) Input matrix
%                    .c                        (1,2) Output matrix
%                    .d                        (1,1) Feedthrough matrix
%                    .inertia                  (3,3) Inertia matrix
%                    .desired_orbit_vector     (3,1) Axis in orbit frame
%                    .eci_vector               (3,1) Axis in ECI frame
%-------------------------------------------------------------------------

%   Outputs
%-------------------------------------------------------------------------
%
%   moment     (3,1) Dipole moment generate by the magnetorquers
%
%-------------------------------------------------------------------------

% Maximum dipole moment
%-------------------------------------------------------------------------
m_max = 0.0069;

% We find the quaternion that aligns a unit vector with the desired vector
%-------------------------------------------------------------------------

q_desired = U2Q( d.eci_vector, d.desired_orbit_vector);

% Compute the quaternion error relative to the desired quaternion
%-------------------------------------------------------------------------
q_error = QMult(QPose(q_ECI_body), q_desired);
```

```
% Maintain sign convention
%-------------------------------------------------------------------------
if (q_error(1,1) < 0.0)
  q_error = -q_error;
end

% Compute the dipole magnetic moment generated by the magnetorquer with
% the following control law
%-------------------------------------------------------------------------
moment =...
    (alfa*(cross(bf,q_error(2:4)))-beta*(cross(bf,anglevel)))/(norm(bf));

% We consider the saturation point of the magnetorquers
%-------------------------------------------------------------------------
if  max(abs(moment)) > m_max
      disp('pre');
d.dipole = moment*m_max/max(abs(moment));
end
```

Annex 14: Attitude Simulator

```
%% Attitude Determination and Control System for a PocketQube
% Institution: Universitat Politècnica de Catalunya (UPC - BarcelonaTech)
% Laboratory: NanoSatellite and Payload Laboratory (NanoSat Lab)
% ------------------------------------------------------------------------
% Authors:
% - Yago Recarey Liste
% - Youssef El Kaisi Rahmoun
% - Jose Faneca Ruedas
% ------------------------------------------------------------------------
% The Attitude simulator is able to control the rotation of the satellite
% and to point at a specific location using the nadir algorithm.
% In this simulator we consider that we know the determination of the
% satellite as the calibration of the sensors is not already done.
% Consequently, the sensors are considered ideal.
% Once the calibration is done, the determination algorithm should be
% implemented.
% The determination algorithm is done by Optimal Request Algorithm.
% ------------------------------------------------------------------------
% This code is made with Princeton Control Toolbox.
%-------------------------------------------------------------------------
% Last revision: 27/07/2023 by Yago Recarey Liste
%-------------------------------------------------------------------------

close all;
clear all;
```

```
%% Orbital Propagator
% We load all the neccessary information about the orbit followed by the
% PocketQube.
%-------------------------------------------------------------------------

load('Orbit_Theory.mat');
```

```
%% Simulation parameters
days  = Orbit.Npoints/10/3600/24;  % [days]
tEnd  = days*86400;                % time [s]
dT    = Orbit.dt;                  % Step [s]
nSim  = Orbit.Npoints;             % Number of simulations
```

```
%% Spacecraft Configuration
% We initialize the data structure of the PocketQube
%-------------------------------------------------------------------------

d = RHSCubeSat;

% Model data
d.surfData.area = d.area;
d.surfData.nFace = d.normals;
d.surfData.rFace = d.r;
```

```
%% Initial state vector
```

```
r    = Orbit.Propagation_ECIPos(:,1)/1000;  % Position vector [km]
v    = Orbit.Propagation_ECIVel(:,1)/1000;  % Velocity vector [km/s]
q    = [1;0;0;0];                           % Initial Quaternion
w    = [30;30;30].*pi/180;                  % Angular rate [rad/s]
c    = [0;0;0];                             % Reaction wheel rates
b    = 2*3600;                              % Battery state of charge
                                            %(J = Wh*3600)

% State is [position;velocity;quaternion;angular velocity;wheels;...
%           battery charge]
%-------------------------------------------------------------------------
x    = [r;v;q;w;c;b];
```

```
%% Start Julian date
% We compute the Julian Date from the date.
%-------------------------------------------------------------------------

d.jD0  = Date2JD([2023 3 31 0 0 0]);
```

```
%% Design the PD Controller
% The function PIDMIMO produces a state space n degree of freedom PID
% controller. This function allows us to obtain the proportional and
% derivative constants. We will set to 0 the integral constant as we
% want a PD controller.
%-------------------------------------------------------------------------
[p.a, p.b, p.c, p.d, constants] = PIDMIMO( 1, 1, 0.01, 200, 0.1, dT );

p.desired_orbit_vector = [0;0;1]; % Allign with the center of the Earth
p.c(1,1)=0;
```

```
%% Initialization of the plotting array to save time
%-------------------------------------------------------------------------
qECIToBody   = x(7:10);

% We compute the local geomagnetic field vector of the Earth for
% a specific location
%-------------------------------------------------------------------------
bFieldECI = BDipole( x(1:3), d.jD0 );

% We transform the geomagnetic vector in the direction of the quaternion.
%-------------------------------------------------------------------------
bField       = QForm( qECIToBody, bFieldECI );

% We unitize the position vector
%-------------------------------------------------------------------------
p.eci_vector = Unit(x(1:3));

% We compute the angle error between our position and the desired position.
% This is, the difference between the z axis and the nadir.
%-------------------------------------------------------------------------
```

```
  angleError    = acos(Dot(-Unit(x(1:3)),...
      QTForm(qECIToBody,p.desired_orbit_vector)))*180/pi;

  disp('MTQ control')

  if abs(w_read_avg(1)) > 2*pi/180 || abs(w_read_avg(2)) > 2*pi/180 || ...
          abs(w_read_avg(3)) > 2*pi/180

    % DETUMBLING ALGORITHM
    %--------------------------------------------------------------------
    [moment] = DetumblingFunction(d,p,dT);
    disp('detumbling')
    d.dipole  = moment;
  else

    % NADIR POINTING ALGORITHM
    %--------------------------------------------------------------------
    [moment]  = PD3Axis( qECIToBody, p, bField, w_read_avg,...
        constants.kP, constants.kR);
    d.dipole = moment;
  end

  % We compute the torque generate by the magnetorquers
  %--------------------------------------------------------------------
  torque=Cross(moment,bField);

  % We compute the current drive by the magnetorquer
  %--------------------------------------------------------------------
  MaximumCurrent=32e-3;
  MQTCurrent(:,k) = fun_MQTCurrent(MaximumCurrent,d.dipole);

  p.bFieldBody_before = bField;

  % A time step with 4th order Runge-Kutta
  %--------------------------------------------------------------------
  x_ant=x;
  x             = RK4( @RHSCubeSat, x_ant, dT, t, d );
  if isnan(x); disp('nanafter'); nSim = k+1; end

  % Get the external disturbance forces and the power
  %--------------------------------------------------------------------
  [xDot, dist, power] = RHSCubeSat( x, t, d );

  % Update plotting and time
  %--------------------------------------------------------------------
  xPlot(:,k+1) = [x;dist.tAerodyn;dist.tGG;angleError;...
      MQTCurrent(:,k);dist.tOptical];
  t             = t + dT;

end
TimeDisplay( 'close' );
```

The end.