UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Escola Superior d'Enginyeries Industrial,
Aeroespacial i Audiovisual de Terrassa

Project of 1 DoF Attitude Control System of 1U Cubesat Based On Reaction Wheel

Thesis

Student:	Yi Qiang Ji Zhang — 季毅强
Director:	Dr. David González Diez
Co-director:	Dr. Manuel Lamich Arocas

Bachelor's degree in Aerospace Technologies Engineering

Technical University of Catalonia — BarcelonaTech

June 22, 2021

Abstract

The following project is undertaken within **PLATHON** (*PLATform of Optical communications in Nanosatellites*) project in **DISEN** and **TIEG** Research Group lead by Dr. David González and Dr. Javier Gago in the ESEIAAT (Escola Superior d'Enginyeries Industrial, Audiovisual i Aeronàutica de Terrassa) School of the Technical School of Catalonia - BarcelonaTech.

PLATHON's main mission is to simulate a Networks System of CubeSats that collect information and communicate with other different orbital group satellites using IoT sensors to retrieve the information towards a ground station minimizing delay and maximizing global coverage. Final year degree and master students from several departments are the main contributors to the project and most of the system components are designed and built by students.

The project's overview encompasses an analytical estimation of the worst-case disturbance torques calculations for the mission. Moreover, a detailed control algorithm is developed for fine pointing and coarse pointing modes in the orbit as well as a 3D real-time monitoring interface. Finally, the design of the Reaction Wheel (RW) is also made following the technical requirements set by the project specifications.

The final stage of the project focuses on the testing and simulation of the implemented control algorithms. To perform the simulation, the CubeSat is introduced in an Air bearing in the centre of a magnetic simulator, which will generate a magnetic field similar to the conditions that the satellite will be subjected in a Low Earth Orbit (LEO). The Onboard Computer (OBC) of the nanosatellite communicates via Bluetooth with the ground computer waiting for commands. The ground station has full access to the satellite's attitude and full control over the distinct modes of the satellite (i.e. Detumbling mode, Pointing mode, Normal mode, among others). Once the magnetic field and other sources of disturbances are set, the reaction wheels and magnetorquers will activate to control the CubeSat's attitude. Analogously, this attitude data measured by the Inertial Measurement Unit (IMU) of the Attitude Determination and Control Subsystem (ADCS) is sent back to the ground station's computer and further displayed with a computer-generated 3D model in real-time. The CubeSat is intended to be powered with both solar panels or a LiPo battery. Running a software-in-the-loop and a hardware-in-the-loop test has shown that the system requires some modifications to achieve more precise results.

Resumen

El siguiente proyecto se realiza dentro de **PLATHON** (*Plataforma de comunicaciones ópticas en nanosatélites*) del proyecto **DISEN** y del Grupo de Investigación **TIEG** liderado por el Dr. David González y el Dr. Javier Gago en la ESEIAAT (Escola Superior d'Enginyeries Industrial, Audiovisual i Aeronàutica de Terrassa) de la Escuela Politécnica de Cataluña - BarcelonaTech.

La misión principal de PLATHON es simular un Sistema de redes de CubeSats que recoja información y se comunique con otros satélites de diferentes grupos orbitales utilizando sensores IoT para recuperar la información hacia una estación terrestre minimizando el retraso y maximizando la cobertura global. Estudiantes de último curso de grado y máster de varios departamentos son los principales colaboradores del proyecto y la mayoría de los componentes del sistema están diseñados y construidos por los estudiantes.

La siguiente tesis comprende una estimación analítica de los cálculos de los pares de perturbación en el peor de los casos para la misión. Además, se desarrolla un algoritmo de control detallado para los modos de apuntamiento fino y grueso en la órbita, así como una interfaz de monitorización 3D en tiempo real. Por último, el diseño de la Rueda de Reacción (RW) también se realiza siguiendo los requisitos técnicos establecidos por las especificaciones del proyecto.

La etapa final del proyecto se centra en la realización de pruebas y la simulaciones de los algoritmos de control implementados. Para realizar la simulación, se introduce el CubeSat en un cojinete de aire en el centro de un simulador magnético, que generará un campo magnético similar a las condiciones a las que estará sometido el satélite en una órbita baja terrestre (LEO). El ordenador de a bordo (OBC) del nanosatélite se comunica vía Bluetooth con el ordenador de tierra a la espera de órdenes (en este caso, el ordenador central del laboratorio). La estación de tierra tiene acceso total a la actitud del satélite y control total sobre los distintos modos del satélite (es decir, modo de Detumbling, modo de apuntamiento, modo normal, entre otros). Una vez fijado el campo magnético y otras fuentes de perturbación, las ruedas de reacción y los magnetorquers se activarán para controlar la actitud del CubeSat. Análogamente, estos datos de actitud medidos por la Unidad de Medición Inercial (IMU) del Subsistema de Determinación y Control de Actitud (ADCS) se envían de vuelta al ordenador de la estación de tierra y se visualizan además con un modelo 3D generado por ordenador en tiempo real. El CubeSat está pensado para ser alimentado tanto con paneles solares como con una batería LiPo. La realización de una prueba de software-in-the-loop y de hardware-in-the-loop ha demostrado que el sistema requiere algunas modificaciones para lograr resultados más precisos.

摘要

以下项目是在 **DISEN**和**TIEG**研究组领导的**PLATHON**（*PLATform of Optical communications in Nanosatellites*）项目中进行的。项目中进行的，该项目由David González博士和Javier Gago博士领导的加泰罗尼亚技术学校的ESEIAAT（Escola Superior d'Enginyeries Industrial, Audiovisual i Aeronàutica de Terrassa）学院的研究小组负责。

PLATHON的主要任务是模拟一个立方体卫星的网络系统，该系统使用IoT传感器收集信息并与其他不同的轨道组卫星进行通信，将信息检索到地面站，最大限度地减少延迟并最大化全球覆盖。来自几个系的毕业班学生和硕士生是这个项目的主要贡献者，大部分的系统部件都是由学生设计和建造的。

该项目的概述包括对任务中最坏情况下的干扰扭矩计算的分析估计。此外，还为轨道上的精细指向和粗略指向模式开发了一个详细的控制算法，以及一个三维实时监控界面。最后，反应轮（RW）的设计也是按照项目规范所规定的技术要求进行的。

项目的最后阶段主要是对已实现的控制算法进行测试和仿真。为了进行模拟，CubeSat被引入到一个磁性模拟器中心的空气轴承中，这将产生一个类似于卫星在低地球轨道（LEO）上所受条件的磁场。纳米卫星的机载计算机（OBC）通过蓝牙与地面计算机进行通信，等待指令。地面站可以完全了解卫星的姿态，并完全控制卫星的不同模式（即脱离模式、指向模式、正常模式等）。一旦磁场和其他干扰源被设定，反应轮和磁轮就会启动，以控制CubeSat的姿态。类似地，由姿态确定和控制子系统（IMU）的惯性测量单元（ADCS）测量的这些姿态数据被送回地面站的计算机，并进一步用计算机生成的三维模型实时显示。CubeSat打算用太阳能电池板或LiPo电池供电。运行软件在环和硬件在环的测试表明，该系统需要进行一些修改，以获得更精确的结果。

Acknowledgments

The author wishes to express his deepest gratitude to his supervisor Prof. Dr. David González and Prof. Dr. Manel Lamich for their guidance, advice, criticism, encouragements and insight throughout the research. I could not have hoped to accomplish what I have in the time I did without our many hours of meetings and discussions; your guidance and support has been essential to the success of this work. Special thanks to DISEN and TIEG Research group for making this project possible.

I would like to thank the faculty of Aerospace Engineering at the Technical University of Catalonia, which has been part of my life for 4 years, in which I learned countless concepts allowed me to achieve astonishing goals.

A hearty thank you also goes out to all professors, assistants and people that gave me insights for all the useful advices and general willingness to help during these 4 years.

A big thanks to all colleagues with which I had the possibility to confront, to help, and also challenge each other.

Thanks to the staff and students of the UPC Space Program's Horus mission (Student's Association Research Group). Being part in the Structural department, ADCS department, Electronics department and Mission Trajectory Analysis enabled me to have a broad overview of the whole process of a mission. This enabled me to incorporate the knowledge and experience earned there and applied it to the actual project.

Thank you to the members of my defense committee and the members of my reading committee. Your comments, criticisms and questions were invaluable in writing this thesis.

Finally, I like to dedicate this work to my family: my father 季国平 (GuoPing Ji), my mother 章崇崇 (Chong Chong Zhang) and my brother 季承昊 (ChengHao Ji) for supporting me with this endeavour and always cheering me up to improve myself everyday. I will always be indebted to them and feel blessed and privileged to have them as my parents.

Declaration on Honour

I declare that,

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

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

all references have been clearly cited,

I'm authorised to make use of the research group TIEG and DISEN related information I'm providing in this document.

I understand that an infringement of this declaration leaves me subject to the foreseen disciplinary actions by The *Technical University of Catalonia - BarcelonaTECH*.

Yi Qiang Ji Zhang		22 June 2021
Name of the Student	Signature	Date

Thesis title: **Project of 1 DoF Attitude Control System of 1U Cubesat Based On Reaction Wheel**

Declaración de Honor

Declaro que,

el trabajo en esta Tesis de Grado es completamente mi propio trabajo,

ninguna parte de esta Tesis de Grado se toma del trabajo de otras personas sin darles crédito,

todas las referencias han sido claramente citadas,

Estoy autorizado a utilizar la información relacionada con los grupos de investigación TIEG y DISEN que estoy proporcionando en este documento.

Entiendo que una infracción de esta declaración me deja sujeto a las acciones disciplinarias previstas por la *Universitat Politècnica de Catalunya - BarcelonaTECH.*

Yi Qiang Ji Zhang		22 Junio 2021
Nombre del estudiante	Firma	Fecha

Titulo de la Tesis: **Project of 1 DoF Attitude Control System of 1U Cubesat Based On Reaction Wheel**

Contents

CONTENTS

List of Figures

List of Tables

List of Codes

Nomenclature

Acronyms

1U   1-Unit CubeSat

2U   2-Unit CubeSat

3U   3-Unit CubeSat

AD0   Address pin

ADCS   Attitude Determination and Control Subsystem

ADC   Analog to Digital Converter

ADC   Attitude Determine and Control

AOCS   Attitude and Orbit Control Systems

C&DH   Command and Data Handling

CAC   CubeSat Acceptance Checklist

CCD   Charged Coupled Devices

CCW   Counterclockwise

CDH   Command & Data Handling

CFS   Core Flight System

COM   Center of Mass

COM   Centre of Mass

COTS   Off-The-Shelf Components

CPU   Central Processing Unit

CW   Clockwise

DAC   Digital to Analog Converter

DISEN   Distributed Sensor Network

DMP   Digital Motion Processor

DOF   Degree of Freedom

ECEF Earth-Centered Earth Fixed (ECEF)

ECI   Earth-Centered Inertial

FCC   Federal Communication Comission

FHSS Frequency-Hopping Spread-Spectrum

GNC   Guidance, Navigation & Control

GND   Ground

HEO   High Earth Orbit

HiTL   Hardware-in-the-Loop

I/O   Input / Output

I^2C Inter-Integrated Circuit

IMU   Inertial Measurement Unit

IoT   Internet of Things

ISM   Industrial, Scientific and Medical

ISR   Interrupt Service Routine

LEO   Low Earth Orbit

LV   Launch Vehicle

MCU   MicroController Unit

MCU   Microcontroller Unit

MEO   Medium Earth Orbit

MISO Master In Slave Out

MOSFET Metal–Oxide Silicon Transistor

MOSI Master Out Slave In

N/A   Not Applicable

NASA National Aeronautics and Space Administration

NCS   Chip Select

NOAA National Oceanic and Atmospheric Administration

OBC   Onboard Computer

OSK   OpenSatKit

$P-POD$ Poly Picosatellite Orbital Deployer

PCB   Printed Circuit Board

PC   Personal Computer

PLATHON PLATform of Optical communications in Nanosatellites

PPOD Poly-Picosatellite Orbital Deployer

PSoC Programmable System on Chip

RBF   Remove Before Flight

RCS   Reaction Control System

RF   Radio Frequency

RWA   Reaction Wheel Assembly (RW + motor)

RW   Reaction Wheel

RXD   Receive data

SCL   Serial Clock

SDA   Serial Data

SiTL   Software-in-the-Loop

SPI   Serial Peripheral Interface

TIEG   Terrassa Industrial Electronics Group

TTC   Tracking, Telemetry & Command

TXD   Transmit data

UART Universal Asynchronous Receiver-Transmitter

UART Universal Asynchronous Receiver/Transmitter

UHF   Ultra High Frequency

Vin   Input voltage

Physics Constants

μ_{Earth}	Earth's gravitational constant	$3.9860 \cdot 10^1 4 \text{ m}^3/\text{s}^2$
c	Speed of light in a vacuum inertial system	299792458 m/s
G	Gravitational Constant	$6.67408 \cdot 10^{-3} \text{ Nm}^3/\text{kgs}^2$
M_{E}	Earth's mass	$5.9742 \cdot 10^{24} \text{ kg}$
R_{Earth}	Earth's mean radius	$6371.000 \cdot 10^3 \text{ m}$

Variables and Parameters

α Angle between the incident ray and $\hat{n}$

$\mathrm{d}m$ Differential of mass

$\mathrm{d}t$ Change in time

$\dot{\Omega}$ Angular acceleration

$\dot{r}$ Velocity of the spacecraft

$\hat{m}_{\mathrm{north}}$ Normal vector representing the position of the north geomagnetic pole

$\hat{n}$ Unit vector normal to the incident surface $\mathrm{d}A_{\mathrm{inc}}$

$\hat{s}$ Unit vector perpendicular to $\hat{n}$ co-planar to the incident ray

μ Gravitational parameter

Ω Angular velocity

ω Angular velocity

ω_{motor} Angular velocity of the motor

ω_{RW} Reaction Wheels angular velocity

ϕ Pitch angle

ϕ Pitch angular velocity

ψ Yaw angle

ψ Yaw angular velocity

ρ Density

σ_n Normal Exchange coefficient

σ_t Momentum Exchange coefficient

τ_D Drag torque

$\tau_{\mathrm{ext,dist}}$ External disturbances torque

τ_{sat} Satellite's torque

θ Roll angle

θ Roll angular velocity

θ_{dev} Maximum deviation angle of the Z axis and therefore is maximum at $\theta_{\mathrm{dev}} = 45°$ in body frame from its local vertical

$\vec{\tau}_{\mathrm{gg}}$ Gravity gradient torque

$\vec{\tau}_{\mathrm{m}}$ Magnetic torque

$\vec{a}$ Acceleration

$\vec{B}_{\mathrm{Earth}}$ Earth's magnetic field

$\vec{F}$	Force
$\vec{p}$	Impulse
$\vec{r}$	Position of the satellite's COM with respect to Earth's COM
$\vec{r}^B$	Position of the satellite in ECEF reference frame
A_{inc}	Incident satellite surface area
A_{inc}	Incident surface area
C_d	Drag coefficient
c_{rd}	Coefficient of specular diffusion
c_{rs}	Coefficient of diffuse reflection
D	Drag
e	Eccentricity
F_{SRP}	Solar Radiation force
h	Altitude above the surface of Earth
h	Height
h_{disk}	Disk height
H_{G}	Angular momentum computed from its center of mass
h_{ring}	Ring height
H_{RW}	Angular momentum generated by the Reaction Wheel
I_{disk}	Reaction Wheel's disk inertia
I_{ring}	Reaction Wheel's ring inertia
I_{RW}	Reaction Wheel's Inertia
I_{RW}	Reaction Wheels inertia
$I_{\text{sat,max}}$	Satellite's maximum moment of inertia matrix
$I_{\text{sat,min}}$	Satellite's minimum moment of inertia matrix
I_{sat}	Satellite's moment of inertia matrix
I_{SRP}	Energy per unit of time through a cross sectional unit area
K_d	Derivative gain constant
K_i	Integral gain constant
K_p	Proportional gain constant
m	Mass

M_{Earth} Satellite's local magnetic induction

$m_{\text{rw,disk}}$ Reaction Wheel's disk mass

$m_{\text{rw,ring}}$ Reaction Wheel's ring mass

$m_{\text{rw,total}}$ Reaction Wheel's total mass (disk and ring)

$m_{\text{RW,total}}$ Reaction Wheel's total mass

$n_{\text{d}A}$ Normal vector of surface dA

O Origin of axis variable

q Quaternion

R Rotation matrix

$R^3_{\text{Earth}}H_0$ Strength of geomagnetic moment

$r_{\text{d}A}$ Location of the centroid

r_{cg} Drag torque

r_{disk} Disk radius

r_{hollow} Reaction Wheel hollow radius

r_{ring} Ring radius

T_{RW} Reaction Wheels distribution matrix

$u(t)$ PID control variable

V Velocity

v^T Spacecraft tangential velocity

x x variable

y y variable

z z variable

Chapter 1

Introduction

The present Bachelor Thesis covers the study of the attitude determination and control system (ADCS) of a future Cubesat, which is the main goal of TIEG, the Electronics Aerospace Research Group of the Polytechnical University of Catalonia.

Cubesats are tiny satellites made up of one or more $10 \times 10 \times 10$ cm cubes. Although reaching Earth orbit may seem like an unattainable achievement, numerous universities and related organizations and institutions have already designed, constructed and launched a CubeSat into orbit.

The CubeSat that is being developed consists of a 1-Unit Cubesat printed using additive manufacturing method with 2 electromagnetic coils (magnetorquers) embedded as actuators that will interact with Earth's geomagnetic field reaction wheel. The symbiotic relation between the two actuators will be responsible for controlling the satellite's attitude and orientation in space. This Cubesat also carries a set of communication modules to interact with the ground station. The main matter is the study of rigid solid mechanics, which will provide the mathematical equations that control the satellite's motion and the interaction with Earth's magnetic field and other disturbances. The CubeSat is intended to be powered using either solar panels or a LiPo battery.

Ultimately, the control algorithm and several simulations are performed for the different pointing modes. The thesis embraces classical Newtonian mechanics, programming several control algorithms, sizing and calibrating sensors and actuators, PCB and CAD designs and assembling all the work done within the rest of the department members. The result of the work is a fully operative automatic controlled CubeSat with the ability to send and receive telemetry data.

1.1   Aim of the project

The following thesis aims to study and design both the hardware and the control software of the Attitude Determination and Control Subsystem of a 1DOF 1U Cubesat using reaction wheels. This project is part of **PLATHON** (*PLATform of Optical communications in Nanosatellites*) Research Group the goal of which is to establish and simulate communications in a constellation of satellites.

A **1 degree of freedom (1DOF)** control of a Cubesat means there is only one system variable that is unbound (free). Degrees of freedom are used instead of variables to clarify the freedom of a system instead of a specific number of variables.

For instance, suppose a 2D grid plane. To get the exact location of any particle at any location of the plane there always must be 2 coordinates. For instance, using Cartesian coordinates the position of the particle is specified by (x, y), or analogously, using polar coordinates the position of the particle is also determined using two variables (r, β), where r is the radius and β is the angle. Thus, there are x, y, r, β multiple variables that can define the position but only using two of them the system's position is fully defined.

The final goal is to send a telecommand with the desired rotation to the CubeSat and the CubeSat's response shall turn into the desired angle and the correct direction. To do so, two main pointing modes were developed, a coarse pointing mode which will trigger the motor driver to make an impulse and then, when the CubeSat is approaching the desired set-point, fine pointing mode comes in with a PD controller (see Section 7).

1.2 Justification

1.2.1 PLATHON Project

The successful development of missions with nanosatellite clusters requires tools that integrate orbital propagation, command and telemetry transmission between the nanosatellite and the ground station and the operational status of the satellites. For missions where nanosatellites are communication nodes, it is also necessary to integrate the communication network model.

The applications that constitute the mission object of the satellite can be validated by simulating the behaviour of all the hardware that surrounds it and emulating the flight conditions (position, inclination, velocity, etc.). This form of software validation is called Software-in-the-Loop (SiTL).

A more advanced step to ensure mission success is to include testing of the vehicle hardware and its correct use by the applications developing the mission. This is covered when the actual hardware can be incorporated into the application development and validation environment. This is known as HiTL (Hardware-in-the-Loop).

The **PLATHON** (*PLATform of Optical communications in Nanosatellites*) Research Group system aims to meet these needs by developing a HiTL (Hardware-in-the-Loop) platform integrated with a network and orbital simulator that allows the verification of the communication network of a nanosatellite cluster considering the operational constraints of the nodes that form it. Specifically, the aim is to address the current needs for the design of low orbit missions that could establish a communications network between satellites in the cluster itself and links with ground stations and with other satellites in other orbits.

All development will be based on the OpenSatKit software developed in open-source by NASA for the simulation of satellite missions. The software will be adapted to the inclusion of SiTL for simulations in the first instance, and subsequently to emulations with HiTL. For this purpose, several real CubeSat

prototypes (equipped with power, communications and ADCS subsystems) will be placed on air-bearing platforms. In this way, the operation of the ADCS system is emulated in conditions as close as possible to an actual flight. These satellite models will communicate with each other via a real UHF link and an optical communication link.

CubeSats appeared as a teaching tool in universities during the '90s but immediately shifted from university labs to commercial applications and business. At present, they have become a mature standard in the aerospace industry. Recently the first CubeSat based constellations intended to offer global communication coverage have appeared. However, we consider that there are some lacks in two areas related to such constellations based missions:

- Simulation tools. Current communication networks simulators used by the scientific community (NS2, SNS3 and Opensand) do not have the features required to design communications among nodes in placed LEO, between LEO and GEO and all above mentioned and ground stations as well. Neither the power availability of nodes to establish proper communications are considered.

- Reliability: CubeSat missions still have a large fail rate (55% in CubeSat from Universities and 33% from Companies). Most of these failures (40%) are attributed to a malfunction of the Electric Power Systems (EPS).
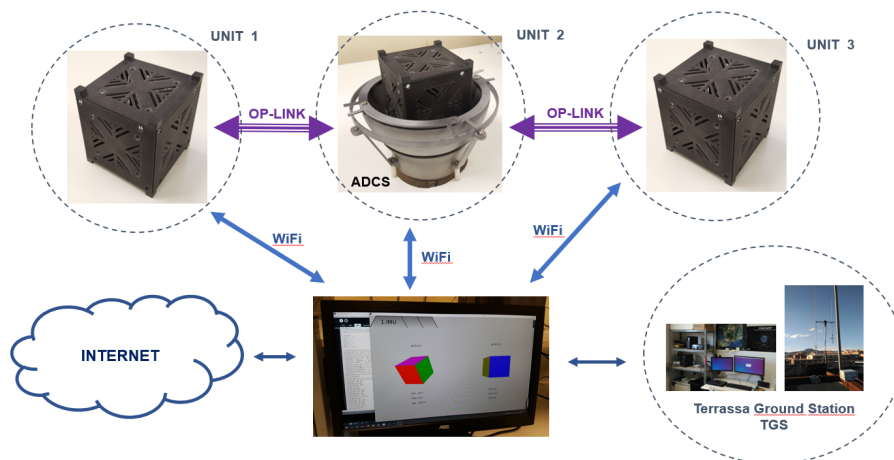
Figure 1.1 PLATHON real architecture. Source: PLATHON [1].

PLATHON project proposes to develop an integrated simulation platform intended as a test bench for validation of engineering models before flight and network communication architectures. This platform considers the following aspects:

1. Integration of an orbital propagator and the restrictions imposed by the visibility among network nodes placed in different orbits (LEO, GEO) and the power availability on each node to establish a reliable optical communication.

2. Design and manufacturing of an emulator (HiL) to test engineering models of CubeSat. This emulator is a CubeSat with EPS, ADCS and COMS, which will be connected to the integrated simulator above mentioned.
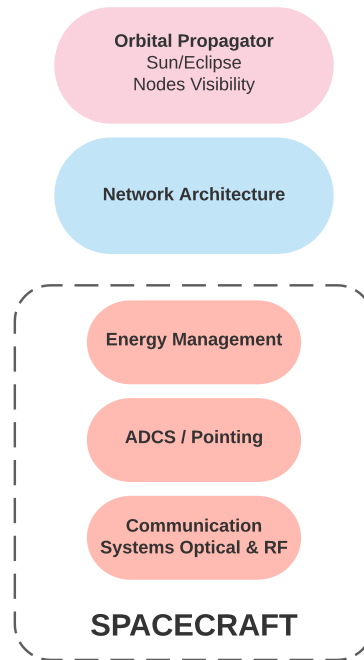
Figure 1.2 PLATHON diagram. Source: PLATHON [1].

This new tests facility will be a helpful assessment tool in the design of more reliable constellation missions.
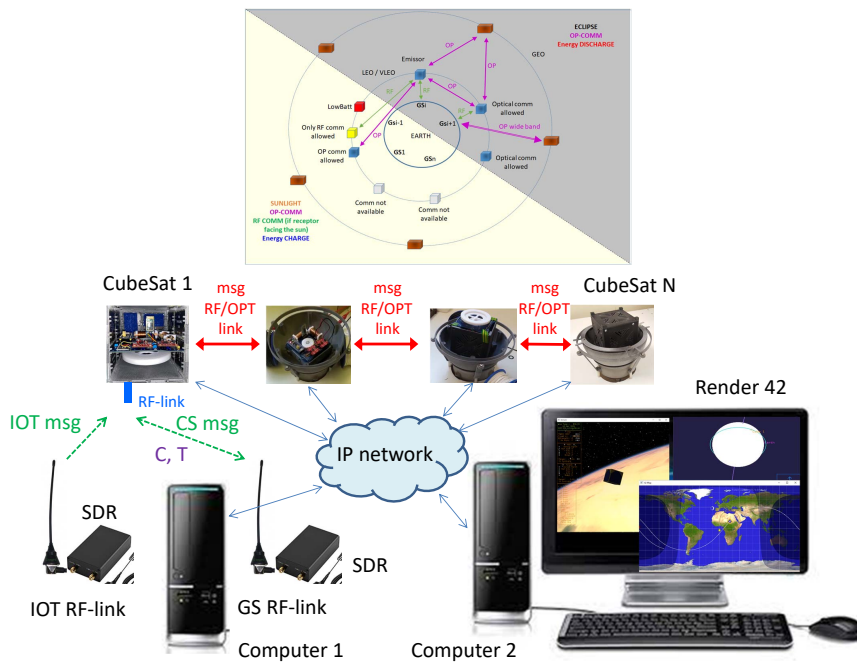
Figure 1.3 PLATHON hardware diagram. Source: PLATHON [1].

Hitherto, the common communication with satellites was using a Ground Station with a specific radio-frequency band to communicate with the satellite. Nevertheless, there are some limitations as the satellite is only visible to the Ground Station in a certain period $(4-6$ min) in general). The major problem that arises was that a huge amount of information needed to be transmitted in an acutely reduced amount of time. Thus, PLATHON project is to use communication between satellites to broaden the effective communication area. At the same time, the operative state of the satellite can be increased as part of the constellation of satellites will always be in the sunlight region.
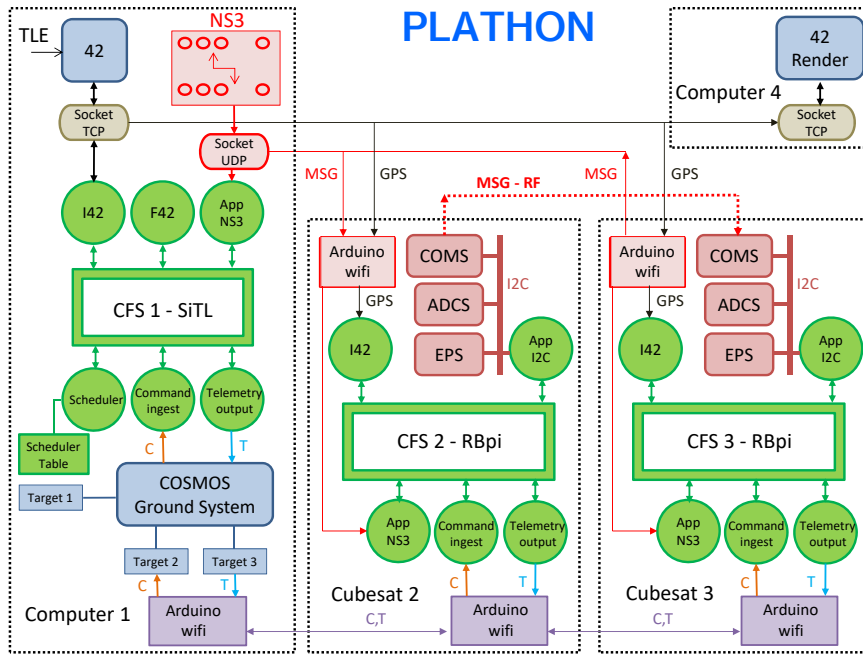
Figure 1.4 PLATHON Open Sat Kit diagram. Source: PLATHON [1].

Step by Step Operation of the PLATHON system

We can synthesise a PLATHON emulation with the following steps:

1. Creating the orbits of the satellites in 42 from TLE data or manually.

2. Designing the mission and translating it into communication links to be carried out at each instant or point of the orbit to bring the data from the original point (e.g. the IoT in our example) to the end (the corresponding GS).

3. Apply NS3 to establish the optimal routes at each instant or point of the orbit.

4. Design the Command and Telemetry messages to be sent between GS and satellite each time they have visibility. Only one GS is selected as a tracking station.

5. There is one satellite simulated as SiTL in Computer 1, and N hardware satellites as HiTL.

6. The communication of 42 and COSMOS with the CFS of the SiTL satellite in Computer 1 is via sockets or software communication ports, as all the software is on the same machine.

7. If 42 manages the satellite attitude control, the attitude and position data is passed from 42 to CFS through the TCP socket to the I42 application.

8. If it is the CFS that performs the attitude control, this data is provided by the F42 application.

9. The NS3 provides it, synchronised with the 42, with the information on the attitude setpoint to be followed at any given moment. Given that we have previously planned the mission, we know at each instant how we have to rotate the antenna or the optical link to focus on another satellite that will be the next node of the communication following the optimal route. This information is sent through the UDP socket to the NS3 App of the CFS.

10. The NS3, once it has estimated that it has already made the turn, tells it which message to send via the antenna or the optical link. It is assumed that the same previous operation has been performed on the receiving satellite, and therefore, they are oriented, and when sending the message, it must be received correctly on the receiving satellite.

11. Each time the tracking station is passed, COSMOS activates the Command Ingest application, to send command messages, and Telemetry Output, to receive the status of the satellite's sensors. This is managed through the scheduler table.

12. The communication of the 42 and COSMOS with the CFS of the HiTL satellite of each satellite is by means of sockets or communication ports through the IP Network with Wi-Fi access points.

13. The 42 sends the position data (GPS) via Wi-Fi to the satellite's Wi-Fi through the IP network. And the Wi-Fi injects it through the I2C bus to the GPS application that transforms it into the CFS format for the satellite's CFS bus software.

14. The NS3 provides it, synchronised with the 42, with the information on the attitude setpoint to be followed at each instant and the message (MSG) to be sent, via Wi-Fi and IP network. The satellite's Wi-Fi will inject the attitude setpoint data through the I2C bus to the ADCS application, and the message (MSG) through the SPI or UART bus to the NS3 application. These applications transform the data format and inject it into the software bus.

15. The ADCS board detects and changes the attitude of the satellite according to the data received, orienting it to the satellite, IoT or GS that corresponds to it at that moment according to the design of the mission.

16. The corresponding data is sent by the COMS board, either in RF or by optical link and is received by the receiving satellite, which will also have carried out the attitude setpoint operations to be oriented to the transmitter. Or they are received by the GS. Or they are transmitted from the IoT, whose hardware is also an SDR.

17. Communication with COSMOS to send the information to or from the tracking station is also done through CFS applications, the Command Ingest or Telemetry Output of the satellite that sends the data by UART or SPI bus to the Wi-Fi or COMS boards.

18. The Command Ingest or Telemetry Output data is sent to Wi-Fi when there is no SDR hardware connected to Computer 1. In this case, the data is sent via IP Network to the Wi-Fi board of Computer 1.

19. If the SDR is connected via USB to Computer 1, the Command Ingest or Telemetry Output data is sent via the COMS RF antenna to the SDR RF antenna.

20. The Command Ingest or Telemetry Output data arriving either by Wi-Fi or by SDR are injected into COSMOS through the corresponding Targets.

21. Finally, the position data of the 42 (GPS) and satellite attitude (ADCS) are sent via IP network to Computer 2 to be displayed on the Render 42 in real-time.

1.3   Scope of the project

The scope of the project encompasses the design and integration of the full assembly inside the 1U Cubesat. All hardware used must be Open Source.

The reaction wheel model must consider all different constraints as the satellite will be subjected to many perturbations and the goal is to obtain the most realistic approach to its attitude control while performing its mission in orbit.

1.4   Project Requirements

- The reaction wheel must not exceed the size of a 1U CubeSat $10 \times 10 \times 10$ cm.
- Every electronic component must fit into the ADCS PCB which carries the microcontroller, motor drivers, a motor and two motor drivers, a Bluetooth module and two magnetorquers.
- The microcontroller must have enough pins to work with several peripherals at the same time and numerous communication interfaces.
- The reaction wheels design must meet the weight constraints set by 1U CubeSat California University requirements [2].
- The code must be a Commercial-Off-The-Shelf (COTS) cod, meaning it shall be used without any customization.

Chapter 2

State of the art

2.1 What is a Cubesat?

Cubesats are small-sized satellites based on a standardized unit of mass and volume. The most basic Cubesat unit has $10\times10\times10$ cm, conforming to specific interfaces for allowing a standardized containerized launch and had a maximum mass of 1 kg (the mass was later on increased to 1.33 kg) [3].

The purpose of the Cubesats is to provide a standard for the design of nanosatellites to reduce cost and development time, increase accessibility to space, and sustain frequent launches. Additionally, thanks to their size and weight, basic Cubesats units could be combined for supplying larger objectives for major missions while adhering to the same constraints and requirements. Therefore, a 3-Unit Cubesat consists of 3 different standard 1-Unit Cubesat stacked together [4].

(a) Basic 1U Cubesat structure. Source: Cubesatworld [5].
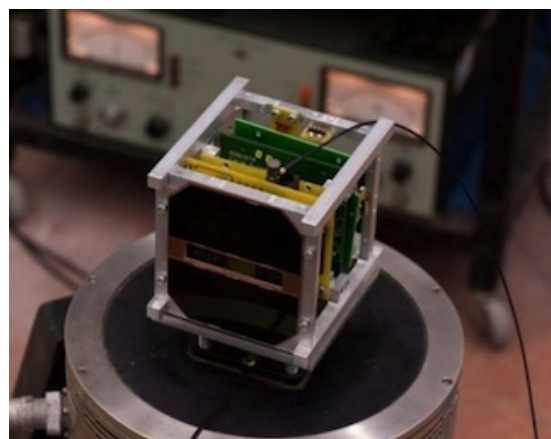
(b) CELESTA Cubesat. Source: CERN [6].

Figure 2.1 Cubesat examples. Source: Cubesatworld [5] and [6].

Cal Poly and Stanford University developed the Poly-Picosatellite Orbital Deployer (P-POD) launch

dispenser standard to provide a low-cost solution to develop and safely launch CubeSats. Since Cubesats are vastly light-weighted satellites, the overall mission costs are reduced exponentially as the amount of fuel needed is significantly minor. Besides, Cubesats have a handful of useful applications such as remote sensing or communications, but as engineers are beginning to get used to the technology, CubeSats are beginning to venture farther afield.

The CubeSat is planned to be modular, with each module or unit measuring $100 \times 100 \times 100$ mm, weighing approximately 1 kg for a 1U CubeSat, and having a functional capacity of 1 L. Its modularity enables relatively flexible scaling, allowing it to accommodate the required components for a specific application in several connected units with little effort or additional cost [2].

Meteorological research, atmospheric analysis, high-energy particle research, disaster monitoring, spacecraft damage research, and spacecraft attitude control systems are among the current experiments conducted in low-earth orbit. In point of fact, because these smaller satellites have the same essential components as their bigger counterparts (communication systems, processing units, and solar panels), the only limitations imposed on t are related to experiments that have lower power requirements.

At the same time, in 1998, at a joint meeting of students between the United States and Japan, Dr, Jordi Puig proposed the idea of launching a spaceship the size of soda water (Coca-Cola) into space. This original concept would subsequently evolve into other nanosatellite projects and influence the CubeSat concept. At the time, the plan was to take a Coke can, add some electronics in it, mount it in a high-powered amateur rocket, launch it, and eject it on a parachute. Professor Shinici Nakazuka and his students at the University of Tokyo began to develop CanSat [7].
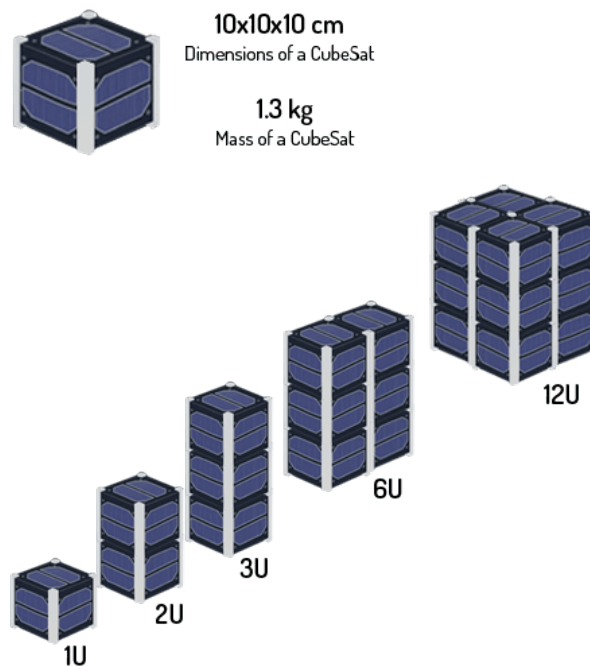
Figure 2.2 Cubesat sizes. Source: Alen Space [8].
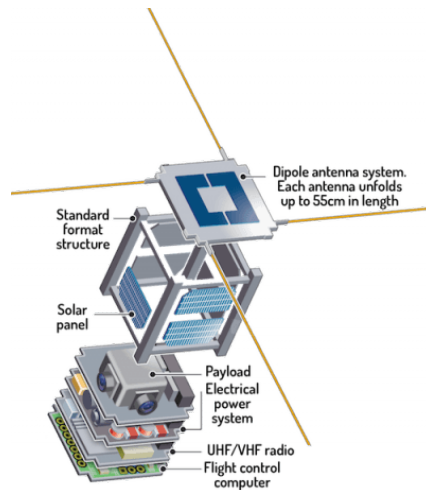
Figure 2.3 Cubesat components. Source: Alen Space [8].

Nanosatellites are typically launched into low circular or elliptical orbits (altitudes ranging from 400 to 650 km) and travels at around 8 km/s. At this altitude and height, it takes them around 90 minutes to orbit the Earth, with 14 to 16 orbits completed every day. These are great configurations for nanosatellites. They are better shielded from solar and cosmic radiation by circling closer to the Earth [8].

Satellites move in circular or elliptical orbits around the Earth because of the balance of gravitational and escape pull at launch. Because air friction is very small, they can stay in orbit for a long operating time. When a nanosatellite's operational life comes to an end, it re-enters the atmosphere and disintegrates.

2.1.1 Satellite Constellations

Usually, nanosatellites work in constellations in which they give support, redundancy, and granularity to the services they deliver. Each satellite in a constellation is replaced every 4-5 years, ensuring that the operator always has an optimized, low-risk service that gets continual technical advances. As a result, nanosatellite constellations are systems in which the ideas of obsolescence and useful life are no longer relevant [8].
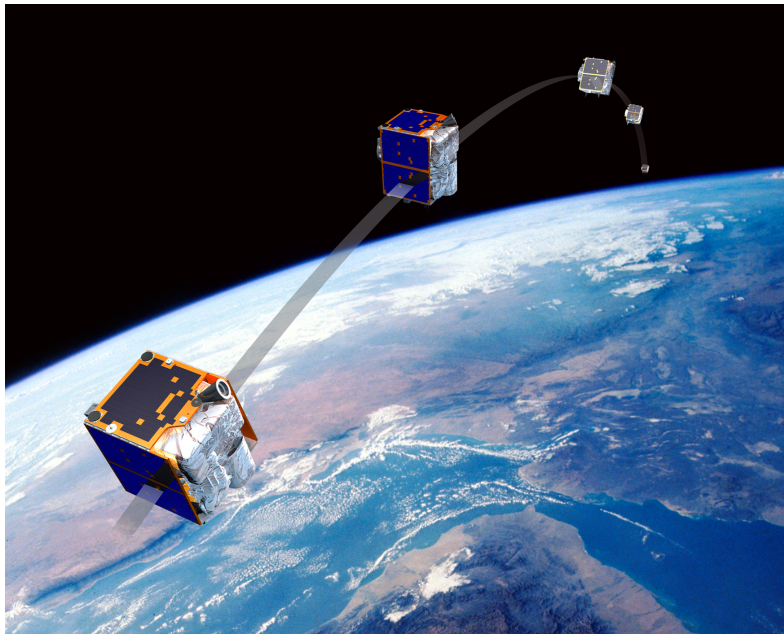
Figure 2.4 Concept of a satellite constellation. Source: Space News [9].

2.1.2 CubeSat Applications

The key driving factors behind the worldwide adoption and growth of CubeSats can be approximately abbreviated as follows [7]:

1. **Affordability**: The intention is to create an instructional tool for a research group and university students. With these platforms, students can develop, design, launch, operate satellites to further research. As simplicity and profitability is key, cost-effectiveness factors are achieved by creating a simplified design, using recommended affordable COTS (Off-The-Shelf Components) and accepted specifications and requirements that streamlined various stages of the development cycle such as deployment, structural design, and some verification requirements.

2. **Standardization**: As the technology becomes more popular design configurations arrived over time. Several engineering solutions and industry standards, such as the PPOD mechanism of common development of electronic devices, were table alternatives and industry standards under the limitations of the design specification restrictions. COTS products for Cubesats enable mass manufacture, and mass production components of whole parts and subsystems, as well as lowering development costs for developers.

3. **High-Return value**: Over time, CubeSats become high-return investments for space applications. This is what allows the CubeSat to perform such a wide range of functions in such a wide range of mission circumstances, thanks to a diverse set of developers and customers.

4. **Technology**: New technologies are implemented day after day in a range of missions such as advancements in material research, manufacture, energy, software algorithms, mission architectures (both segments of space and terrestrial segments) and communications.

2.2   Satellite Subsystems

Alike any other satellite, CubeSats incorporates numerous subsystems such as Structures, Attitude Determination and Control Subsystem, Power Subsystem, among others. For instance, the On-Board Computer (OBC) belongs to the communication subsystem. NASA provides a multi-facet platform Core Flight System (CFS) that can serve as a command and control platform using 42 Simulator.

The principal subsystems are listed below:

Table 2.1 Principal subsystems. Source: [10]

Subsystem	Principal Function	Other denominations
Propulsion	Provides thrust to adjust orbit and attitude, and to manage angular momentum	Reaction Control System (RCS)
Attitude Determination & Control System (ADCS)	Provides determination and control of attitude and orbit position, plus pointing of spacecraft and appendages	Guidance, Navigation & Control (GNC) Attitude and Orbit Control Systems (AOCS)
Communication	Communicates with ground and other spacecraft Spacecraft tracking	Tracking, Telemetry & Command (TTC)
Command & Data Handling (CDH)	Processes and distributes commands Processes, stores and formats data	Spacecraft Computer System Spacecraft Processor
Thermal	Maintains equipment and structure within environmental allowed temperature ranges	Environmental Control System
Structures and Mechanisms	Provides support structure, booster adapter, and moving parts	Structure Subsystem

Regarding dry-mass allocation [10],

- Payload = 28% of total satellite mass.

- Power = 23% of total satellite mass.

- Structure = 18% of total satellite mass.

- Propulsion = 12% of total satellite mass.

2.2.1   Attitude Determination and Control Subsystem

Attitude determination is the process of combining available sensor inputs with spacecraft dynamics information to offer an accurate and unique answer for the attitude state as a function of time, either

on-board for immediate use or after the fact (i.e. post-processing). Most attitude algorithms that were previously conducted as post-processing may now be implemented as on-board computations, due to the powerful microprocessors available for spaceflight. As a result, while there are still compelling engineering reasons for certain procedures to be carried out only by ground-based attitude systems [11].

To sum up, the ADCS initials yields for [12]:

- **Attitude**

    - The orientation of a spacecraft in space.
    - The angular orientation of the spacecraft body frame with respect to an external frame.
    - Describes the rotational motion of the body of the spacecraft about its center of mass.

- **Determination**

    - The process of computing the orientation of the spacecraft relative to either an inertial reference or some object of interest, involving:
        * Several types of sensors on each spacecraft
        * Sophisticated data processing procedures

- **Control**

    - Attitude stabilization: the process of maintaining an existing orientation.
    - Attitude maneuver control: the process of controlling the reorientation of the spacecraft from one attitude to another.

2.2.2 Attitude determination

Reviewing Wertz's documentation [12], there are different instruments that aids in the process of attitude determination.

2.2.2.1 Sun Sensors

Sun sensors are visible-light or infrared detectors that measure one or two angles between their mounting base and incident sunlight. These devices have a thin entry slit at the top surface which is sensitive to light. When the Sun's light rays enter the thin cabin on the base of the chamber, it illuminates a sequence of photo-cell detectors. Eventually, the digital output can be mapped onto a unique entry angle, allowing the plane in which the Sun lies to be determined. Sun sensor systems are designed to give two fundamental output signal types, analogue and digital, even though hardware designs vary greatly. The output of an analogue system is a continuous function of the angle of incidence, whereas the output of a digital system is discrete [13]. Sun sensors can be quite accurate ($< 0.01 deg$), but it is not always possible to take advantage of that feature [11] since most low earth orbits include eclipse periods.

Figure 1.—Analog sun sensor.

Figure 2.—Digital sun sensor.
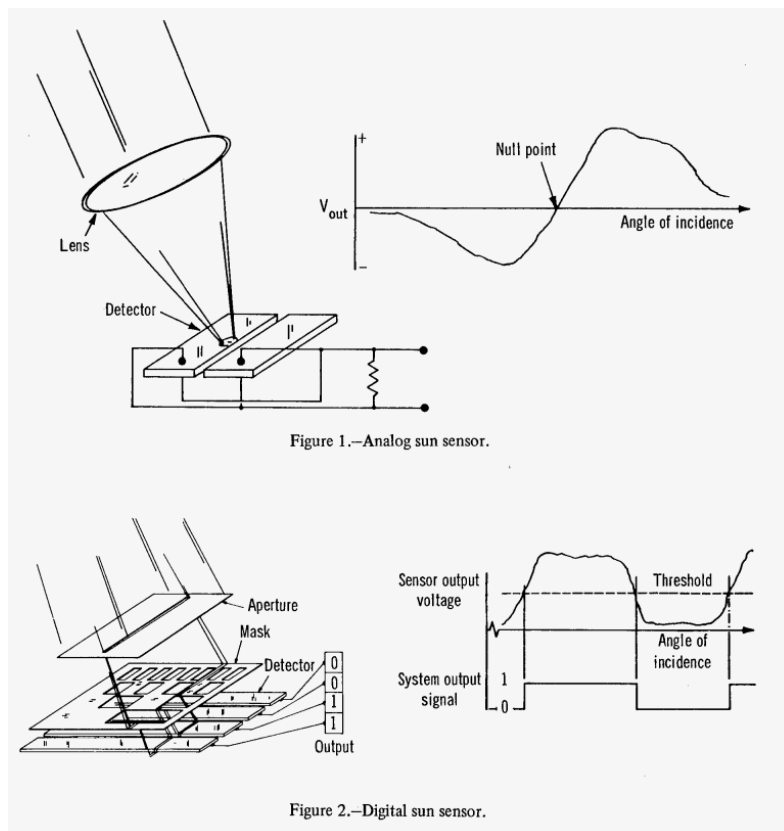
Figure 2.5 Analog and digital sun sensors. Source: NASA [13].

2.2.2.2 Magnetometers

Magnetometers are simple, reliable, lightweight sensors that detect the amount and direction of the Earth's magnetic field. The magnetometer output assists in determining the spacecraft's attitude relative to the local magnetic field. A flux-gate magnetometer measures the strength of the magnetic field in one direction. Thereby, adding a 3-axis magnetometer, which consists of three orthogonal magnetometers will permit measurements of the three components of the magnetic field in the spacecraft body frame. Subsequently, the attitude is determined by comparing the measured field to an inertial magnetic model held in the onboard processor [14] [11].

2.2.2.3 Earth Horizon Sensors

Earth Horizon sensors are infrared devices that measure the difference in temperature between deep space and the Earth's atmosphere (approximately 40 km above the surface in the detected range). Pippers (simple narrow field-of-view fixed-head) are used to detect the Earth phase. These devices are static sensors that can provide only roll and pitch information. Their operating principle consists in detecting Infra-Red (IR) radiation in a given range, for instance 14 to 16 μm (the emission band of the CO_2 CO_2 molecule) [15] [11].

A mirror (or prism) focuses a narrow pencil of light onto a sensing element. Afterwards, the mirror

spins as a result of spinning the spacecraft. As the mirror or prism rotates its field of view sweeps out a cone. Then, electronics in the sensor detect when the infrared signal from the Earth is first received or finally lost during each sweep of the scan cone. Finally, the time between the arrival and loss of signal determines the Earth width.
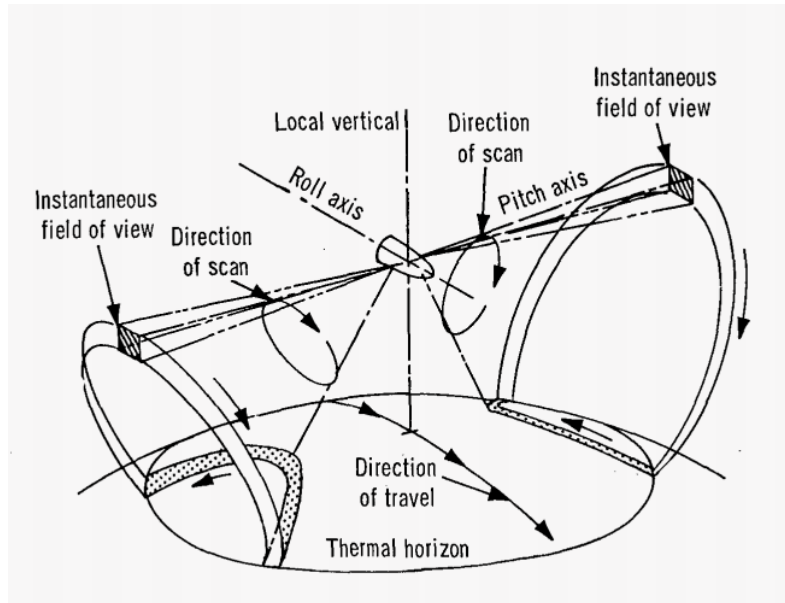
Figure 2.6 Earth horizon sensor. Source: NASA [15].

2.2.2.4 Star sensors/trackers

Star sensors are the most accurate reference sensors for measuring attitude. The light from a star strikes a light-sensitive surface. The point of impact on the surface is determined using Charged Coupled Devices (CCDs), similar to the optical element in a video camera. Then, using a star catalogue, internal processing calculates a three-axis attitude. Many systems can calculate an extremely precise attitude within seconds of turning on [11].
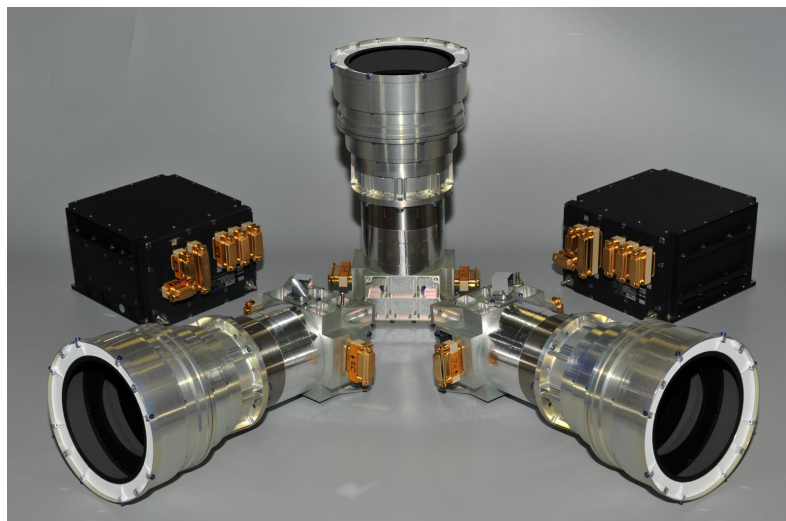
Figure 2.7 Star sensor. Source: ESA [16].

2.2.2.5 Gyroscopes

Gyroscopes are any instrument that uses a rapidly spinning mass to sense and responds to changes in the inertial orientation of its spin axis. These inertial sensors measure the speed or angle of rotation from an initial reference but lack knowledge of an external, absolute reference [11].

2.2.2.6 Inertial Measurement Unit

Inertial Measurement Units are devices that measure a spacecraft's accelerations within a stable frame of reference provided by key ingredients: gyroscopes and accelerometers.

The gyroscope's role is to detect instantaneous motions that might disrupt the spacecraft's stability and respond with compensating drive signals to spin the gimbals properly. The accelerometers' duty is to monitor acceleration forces operating on the spacecraft – and so changes in spacecraft direction and position – within a gyro-stabilized frame of reference. Because gyroscopes and accelerometers are mechanical instruments, the gyros may drift a tad over lengthy missions [17].
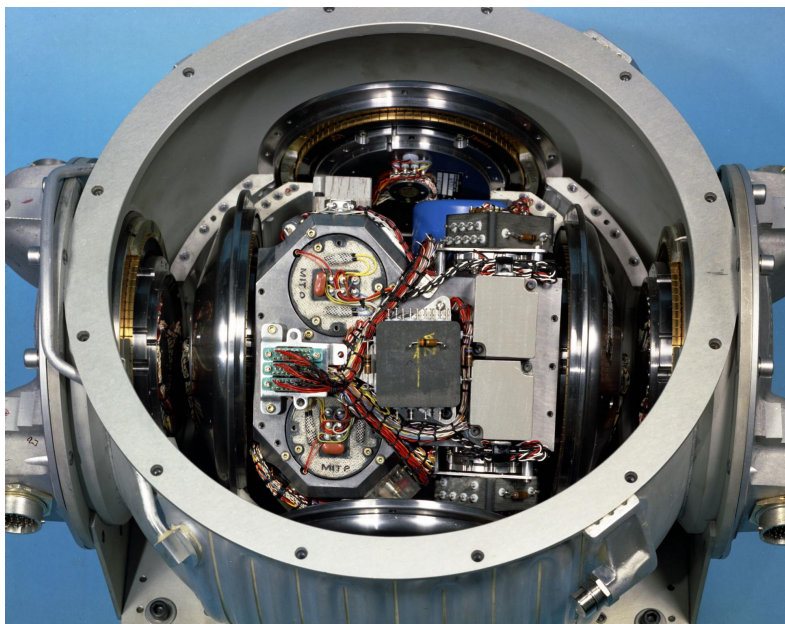
Figure 2.8 Apollo Inertial Measurement Unit. Source: We Hack the Moon [17].

2.2.3 Attitude Control

Once the attitude is determined by the aforementioned instruments and databases, the next step is to properly adjust the orientation of the spacecraft to stabilize or point towards the desired direction. Here is where attitude control comes into play. The following set of instruments helps the Cubesat to perform orbit manoeuvres and each of them has different torques as the goal may be a fine or coarse pointing.

2.2.3.1   Magnetorquers

Magnetorquers are often used as actuation actuators in spacecraft. These torques are magnetic coils (electromagnets) that produce magnetic dipole moments of a specified magnitude. The magnetic field generated by the spacecraft interacts with the local field from a Planet, producing an external torque on the vehicle. Then electromagnets may be used to provide a controllable external torque. Their strength can be controlled by employing current. When three orthogonal magnetorquers are attached to a spacecraft, they may generate a magnetic dipole in any direction and magnitude up to the magnetorquers' strength. Magnetic torquers can adjust for residual magnetic fields or attitude drift caused by modest disturbance torques on a spacecraft. Besides, they can also be used to desaturate momentum-exchange systems. However, they require much more run time than other actuators since their torque is quite small and requires longer run-times [11].

2.2.3.2   Reaction Wheels

Reaction wheels are essentially devices that provide attitude control and stability on a spacecraft. They allow the spacecraft to rotate quickly towards the desired location, in other words, they are torque motors with high inertia rotors. By adding or removing energy from the flywheel, torque is applied to a single axis of the spacecraft, causing it to react by rotating. These wheels can spin at different speeds using electric motors. If the wheel turns clockwise direction, according to Newton's Third Law, the CubeSat will rotate in the opposite direction (counterclockwise). By maintaining flywheel rotation, known as momentum, a single axis of the spacecraft is stabilized. Several reaction/momentum wheels can be used to provide full three-axis attitude control and stability [11].

One major problem appears when slowing down the wheels, reaction wheels cannot spin indefinitely and if they slow down, the spacecraft will turn in the opposite direction.

Let's define the differences between reaction wheels and momentum wheels to avoid future confusion with the terminology. Both reaction and momentum wheels are flywheels that spin and creates torque by changing their momentum. However, there is a major dissimilarity between the two actuators:

- Reaction wheels are spun to create the torque and hence, force the spacecraft to rotate.
- Momentum wheels, on the other hand, are constantly rotating at very high speed to provide a nearly constant angular momentum which creates a stabilization of the spacecraft, thus, generating an angular momentum by creating a resistance in the change of the spacecraft's attitude.

2.2.3.3   Thrusters

Because of its dual utility in altering orbital parameters, thrusters (i.e. rocket engines) are arguably the most commonly flown attitude actuator. Almost every spacecraft that has to conduct orbital manoeuvres will utilize thrusters to do so, and in many cases, part of the thrusters utilized will be for attitude control. Thrusters exert force on the spaceship by ejecting a high-velocity material known as propellant from its exit nozzles [11].

Thrusters produce torque in proportion to their moment torque, which is the distance by which the direction of the force is offset from the vehicle's centre of mass. So, while a thruster may generate a lot of force, the torque it can produce is limited by the actual size of the vehicle and how the thrusters are placed on it. Thrusters have the advantage of being able to provide large instantaneous manoeuvres at any time in orbit. Notwithstanding, the other issue with thrusters is that the amount of stored gas is not unlimited.

Chapter 3

Mathematical description of the physical system

The following section intends to summarize the mathematical model expressions of the basic astrodynamics equations and the development of it.

3.1   Orbit Classification

The preceding section's characterization of orbital elements allows for the establishment of an orbital taxonomy. As a matter of fact, orbits are classified based on their altitude (relative to the semi-major axis a), eccentricity e, and inclination i. Low Earth orbits (LEO) are those with a maximum distance from the Earth's centre of mass of less than 2000 km; medium Earth orbits (MEO) are those between 2000 and $20,000$ km; and high Earth orbits (HEO) are those with a maximum distance more than $20,000$ km [7].

Closed and open orbits are differentiated by eccentricity (see Table 3.1): circular ($e = 0$) and elliptical ($e > 1$) orbits are closed trajectories, whereas parabolic ($e = 1$) and hyperbolic ($e > 1$) orbits are open paths that can escape the gravitational attraction of the celestial body. Finally, orbits can be classified as equatorial ($i = 0$ degrees) or polar ($i = 90$ degrees) based on their inclination.

Table 3.1 Conic sections of an orbit. Source: Own.

Eccentricity	Conic section	Open/closed trajectory
$e = 0$	Circle	Closed
$0 < e < 1$	Ellipse	Closed
$e = 1$	Parabola	Open
$e > 1$	Hyperbola	Closed

Some of the most important orbits are produced by combining some of the properties mentioned be-

forehand. For instance, geosynchronous orbits have a duration equal to one sidereal day. Geostationary orbits are zero-inclination geosynchronous circular orbits. Taking the point of view of an observer on Earth, a geostationary satellite seems fixed in the sky, which is why these orbits are utilized for communications missions, since they are always in the same position and communication can be performed without problems. CubeSats are unlikely to be launched into geostationary orbits, but the majority of them have been launched into a different type of synchronous orbit known as the Sun-synchronous orbit, which is nearly a polar orbit.

Notice how in a Sun-synchronous orbit, the orbital plane rotates to maintain a consistent orientation concerning the Sun throughout the year ($\dot{\Omega} = 360$ degrees per year. The spaceship will travel over the Earth at the same local solar time at all times of the year (points 1, 2, and 3). This orbit is extremely useful for remote sensing missions since Earth's surface are hugely susceptible to variations in lighting conditions.
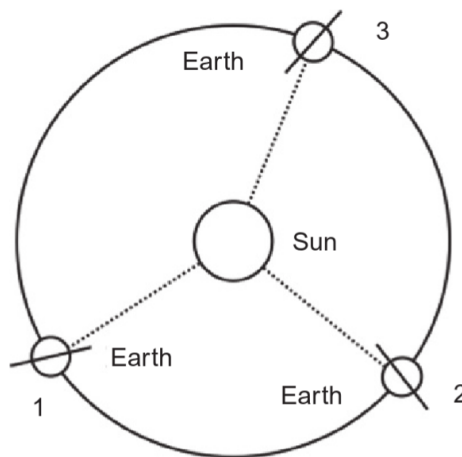
Figure 3.1 Orientation of the orbital plane in a Sun-Synchronous orbit.
Source: Cubesat Handbook [7].

For low orbits, orbits with inclinations around 90 degrees are the most accessible for CubeSats. As a result, a satellite in a low Sun-synchronous orbit traverses the whole Earth's surface multiple times per day, providing several possibilities for interaction with ground stations. This is critical, especially if the spacecraft can only be operated by a single station, as is the situation with many academic missions. The Earth is always in the same local solar time zone and, as mentioned before, this is critical for distant sensing missions since the Earth's surface is incomparable lighting conditions at each passing.

3.2 Attitude description

First of all, before beginning to work on the actual equations a reference frame must be defined. This is crucial since the actual parameters depend on the reference frames. In both reference systems and attitude representation methods and rotation matrices are to be well defined to present accurately the satellite's dynamic equations.

3.2.1 Reference frames

Flight mechanics use a series of specific reference systems with the aim of projecting the positions, rotations linear and angular velocities, linear and angular accelerations, forces and torques in them. These systems can be represented in a generic way as $F(O, x, y, z)$ with its origin in (O) and 3 perpendicular axis (x, y, z) [18] [19] [20].

3.2.1.1 Earth-Centered Inertial (ECI) Reference Frame

The ECI $F(O_{\mathrm{eci}}, x_{\mathrm{eci}}, y_{\mathrm{eci}}, z_{\mathrm{eci}})$ is a fixed in the spatial reference frame, this means that there is not any kind of acceleration so Newton's second is applicable [1].

- The origin of this reference frame is located at the centre of the Earth (the geo-centre of Earth)
- The z-axis points towards the North Pole, specifically, towards Epoch J2000 which is near the Pole Star.
- The x-axis points towards the Vernal Equinox at J2000 (where equatorial and ecliptic planes intersect).
- The y-axis is orthogonal to both of the prior axis following the right-hand rule.

Note the Equatorial plane and Ecliptic plane are inclined at 23º 27' to each other.
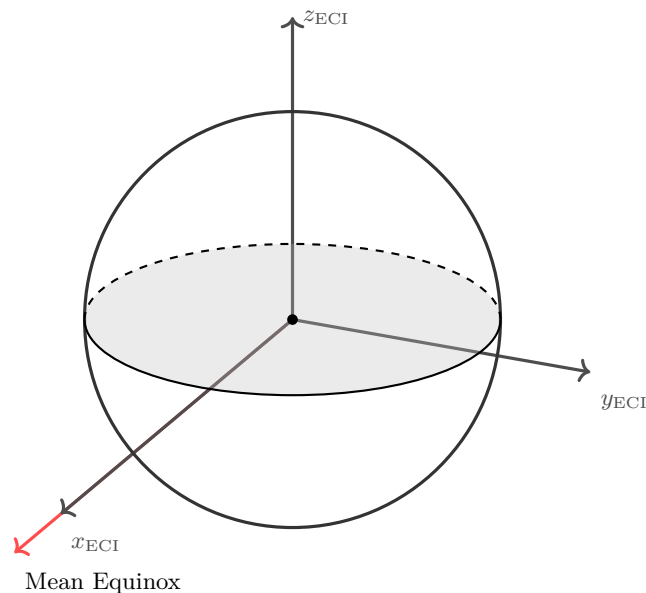
Figure 3.2 ECI reference frame with x_{ECEF} pointing towards the mean
Equinox. Source: Own.

---

[1]Newton's laws apply only in "inertial" non-rotating systems that are not rotating or non-accelerating concerning the distant galaxies or accelerating. All inertial reference frames can be seen as being in a state of constant velocity or rectilinear (straight-line) motion concerning one another. Consequently, an accelerometer moving with any inertial frame of reference should always detect zero acceleration.

3.2.1.2   Earth-Centered Earth Fixed (ECEF) Reference Frame

The ECEF reference frame $(F(O_{\text{ecef}}, x_{\text{ecef}}, y_{\text{ecef}}, z_{\text{ecef}}))$ is linked to Earth itself in a sense that, in a generic instant, the inertial system rotates with the same angular velocity as the rotation of the Earth around its axis with an angular velocity of $\omega = 7.27 \cdot 10^{-5}$ rad/s [21].

- Just as the ECI, the ECEF's origin is also located at the center of the Earth.
- The z-axis points towards the North Pole, specifically, towards Epoch J2000 which is near the Pole Star.
- The x-axis is contained in the intersection between the Greenwich Meridian and the Equator plane.
- The y-axis is orthogonal to both of the prior axis following the right hand rule.
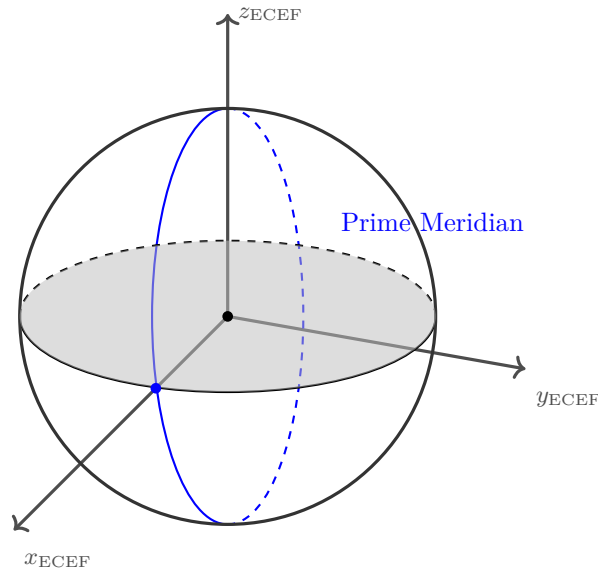
Figure 3.3 ECEF reference frame with the x-axis pointing to the fixed point on the Prime Meridian. Source: Own.

3.2.1.3   Orbit Reference Frame

This reference frame is also known as the satellite's reference frame. This coordinate system is very useful for satellites that have to perform terrestrial imaging manoeuvres.

- The origin of coordinates corresponds to the Centre of Mass (COM) of the satellite itself.
- The z-axis points towards the centre of Earth.
- The x-axis follows the satellite's velocity vector in the same direction.
- The y-axis is orthogonal to both of the prior axis following the right-hand rule.

At any given time, the Satellite Reference Frame and Satellite Body Frame will be at the same location, i.e., the origin of the Satellite Body Frame will always coincide with that of the Satellite Reference Frame,

however, the Satellite Reference Frame may not be aligned with Satellite Body Frame. The ADCS of a satellite works to align the Satellite Body Frame with the Satellite Reference Frame. To simplify our test cases and to understand ADCS easily, we will be considering a satellite whose reference frame is the Orbit Reference Frame.
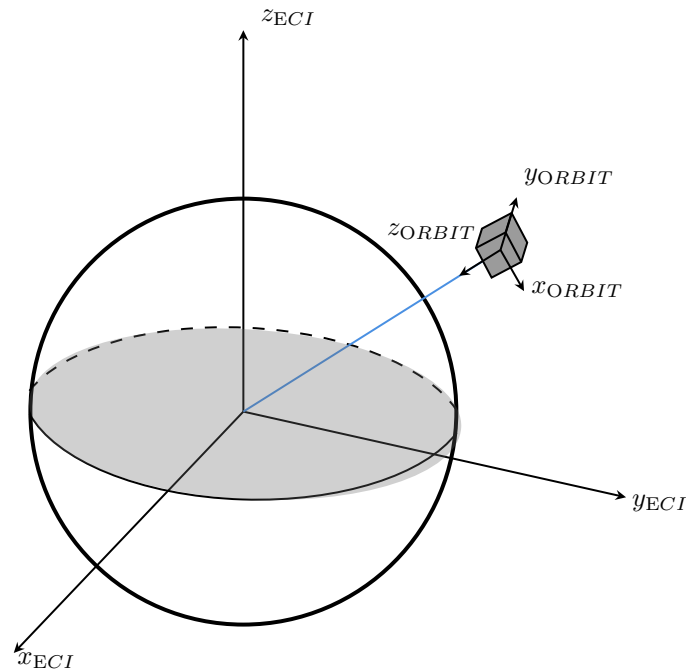
Figure 3.4 Orbit Reference Frame. Source: Own.

3.2.1.4 Body Reference Frame

Unlike the Orbit reference frame, the Body reference frame is fixed to the satellite. The orientation in space is defined with respect to the Orbit frame using *Euler angles* thus representing the satellite's attitude.

- The origin of coordinates correspond to the Centre of Mass (COM) of the satellite itself.
- The $x\,y\,z$ axis points along the principal directions of the spacecraft, normal to the satellites surfaces forming an orthogonal Cartesian coordinate system.

Though, depending on the mission, these axis can be set according to each mission's specifications. Each mission shall bear in mind to adapt the dynamics equations in consonant of the selected reference frame. Eventually, by doing this will only affect the representation of the motion but the entire dynamics of the system remain the same no matter which reference frame it is used.

This reference frame will be used for acquiring data for the CubeSat with the incorporated IMU. Ans the orientation angles will be referenced to this particular frame.
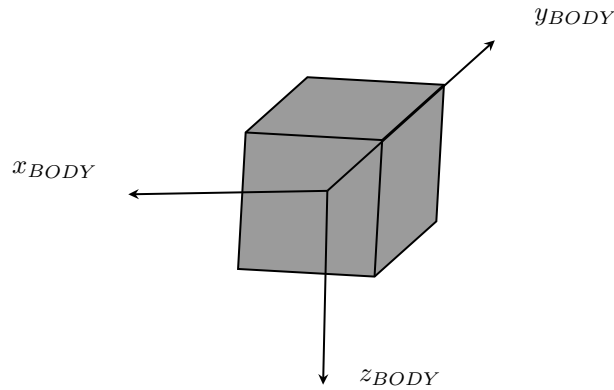
Figure 3.5 Body Reference Frame. Source: Own.

3.2.2 Euler Angles

So far, in the latter section was explained how the orientation of the Body frame with respect to the satellite's Orbit frame describes its attitude. This is expressed in Euler angles. According to *Euler's rotation theorem*, any rotation may be described using three angles. If the rotations are written in terms of rotation matrices D, C, and B, then a general rotation A can be written as

$$A = R_B^O = R_{x,\psi} R_{x',\theta} R_{z',\phi} = BCD \tag{3.2.1}$$

These three rotations matrices provides three different angles called *Euler angles*. The convention used to define the Euler angles is illustrated below. Let A be

$$A = R_B^O = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} \tag{3.2.2}$$

The rotation given by Euler angles (ϕ, θ, ψ) is as follows, where

1. The first rotation is by an angle ψ about the z-axis using D.

2. The second rotation is by an angle $\theta \in [0, \pi]$ about the former x-axis (now x') using C.

3. The third rotation is by an angle ϕ about the former z-axis (now z') using B.

Here, the notation (ψ, θ, ϕ) is used. In the x -convention, the component rotations are then given by the following rotations:

$$R_{z',\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi & \cos\phi \end{bmatrix} \quad R_{x',\theta} = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \quad R_{x,\psi} = \begin{bmatrix} \cos\psi & -\sin\psi & 0 \\ \sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$
$$\tag{3.2.3}$$

Finally, by developing the expressions (3.2.3) the rotation matrix is described as a rotation across all the axis. The final matrix can be expressed as

$$R_B^O = R_z(\psi)R_y(\theta)R_x(\phi) =$$

$$= \begin{bmatrix} \cos\psi\cos\theta & -\sin\psi\cos\phi + \cos\psi\sin\theta s\phi & \sin\psi\sin\phi + \cos\psi\cos\phi\sin\theta \\ \sin\psi\cos\theta & \cos\psi\cos\phi + \sin\psi\sin\theta\cos\phi & -\cos\psi\cos\phi + \cos\psi\cos\phi\cos\theta \\ -\sin\theta & \cos\theta\sin\phi & \cos\theta\cos\phi \end{bmatrix} \quad (3.2.4)$$

(a) Euler for I. Source: Stack-exchange [22].

(b) Euler for I, for 1. Source: Stack-exchange [22].

(c) Euler for I, for 1, for 2. Source: Stack-exchange [22].

(d) Euler for I, for 1 for 2, for B. Source: Stack-exchange [22].

(e) Euler for I for B. Source: Stack-exchange [22].
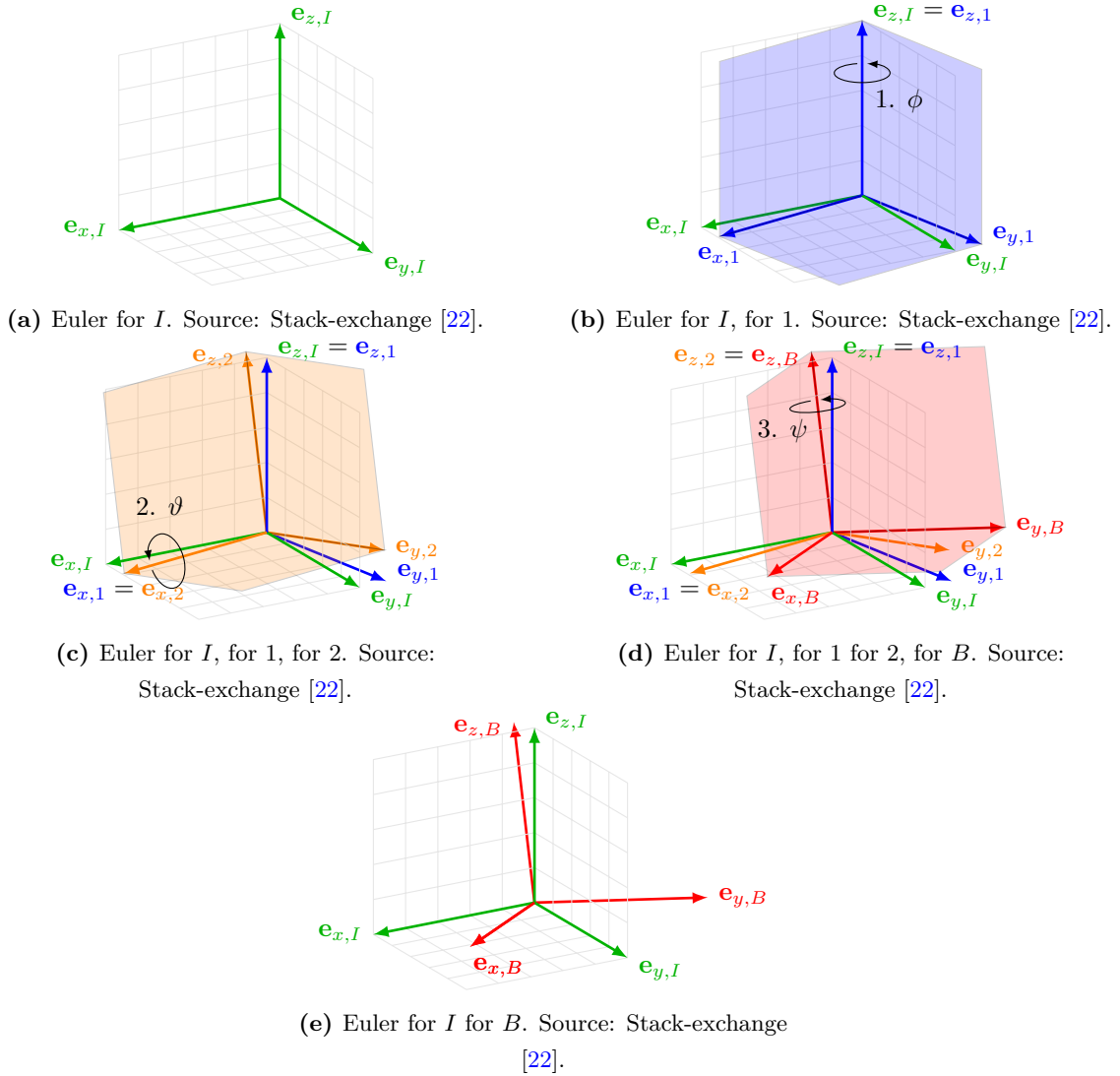
Figure 3.6 Euler angles rotations. Source: Stack-exchange [22].

Eventually, the angular velocities ω can be determined by the Euler angles derivatives [20].

Since the position is uniquely defined by Euler's angles, angular velocity is expressible in terms of these angles and their derivatives. The angular velocity vector $\Omega_B^O = [\Omega_1\Omega_2\Omega_3]^T$, which components are the rates is determined by the Euler angles and the derivative of ϕ, θ, ψ. The angle rates are respectively are known as *precession rate* $\dot\phi$, *nutation rate* $\dot\theta$, and *spin* $\dot\psi$ [20].

To do so, the strategy is is to find the angular velocity components along the body axes x_1, x_2, x_3 of $\dot{\phi}$ $\dot{\theta}$ $\dot{\psi}$ in turn. $\dot{\theta}$ is along the line ON, and hence, in the x_1, x_2 plane. Next, $\dot{\phi}$ is about the Z axis.

Thus, the Euler angle angular velocities (components along the body's principal axes) are:

$$\begin{aligned}
\vec{\theta} &= (\dot{\theta}\cos\psi, -\dot{\theta}\sin\psi, 0) \\
\vec{\phi} &= (\dot{\phi}\sin\theta\sin\psi, \dot{\phi}\sin\theta\cos\psi, \dot{\phi}\cos\theta) \\
\vec{\psi} &= (0, 0, \dot{\psi})
\end{aligned} \tag{3.2.5}$$

and the angular velocity components velocity components along those in-body axes x_1, x_2, x_3 are:

$$\Omega_B^O = \begin{cases} \Omega_1 = \dot{\phi}\sin\theta\sin\psi + \dot{\theta}\cos\psi \\[2mm] \Omega_2 = \dot{\phi}\sin\theta\cos\psi - \dot{\theta}\sin\psi \\[2mm] \Omega_3 = \dot{\phi}\cos\theta + \dot{\psi} \end{cases} \tag{3.2.6}$$

The above expressions (3.2.6) can be solved in terms of precession, nutation and spin rates:

$$\begin{aligned}
\dot{\phi} &= \tfrac{1}{\sin\theta}\left(\Omega_1\sin\psi + \Omega_2\cos\psi\right) \\
\dot{\theta} &= \Omega_1\cos\psi + \Omega_2\sin\psi \\
\dot{\psi} &= -\tfrac{1}{\tan\theta}\left(\Omega_1\sin\psi + \Omega_2\cos\psi\right) + \Omega_3
\end{aligned} \tag{3.2.7}$$

It is crucial to avoid nutation angle through 90 ° so as to avoid the singularity that exists in the precession angle [23].

Euler angles are astoundingly useful to get an idea of a satellite´s attitude. Notwithstanding, a particular scenario can cause Euler angles to provide erroneous attitude due to singularities for some angles, e.g. $\cos\pi/2$ when calculating the resulting rotation matrix. As attitude exists in $\mathbb{SO}^3$, no three-parameter representation, such as Euler angles or the Gibbs Vector is singularity free. For this reason *Euler parameters*, also called *quaternions* are used in the numerical computations in this project. However, for ease of interpretation, Euler angles are often used to represent the results.

3.2.2.1   Euler Gimbal Lock

Euler gimbal lock occurs when the orientation of the sensor cannot be uniquely described using Euler Angles. The exact orientation at which the gimbal lock occurs is determined by the rotational sequence used. When the pitch angle α is 90 degrees, the sequencing of actions on the IMU sensor results in a gimbal lock. This results in a loss of a degree of freedom since two axes are coupled and there is no third axis to turn on.

To avoid Euler Gimbal Lock singularities, the correct way to approach this problem is using *quaternions* (see Section 3.2.3) by taking the Euler parameters (quaternions) and performing a transformation, the rotation matrix expressed in terms of quaternions is obtained as

$$R_B^O = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2\left(q_1q_2 + q_0q_3\right) & 2\left(q_1q_3 - q_0q_2\right) \\ 2\left(q_1q_2 - q_0q_3\right) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2\left(q_2q_3 + q_0q_1\right) \\ 2\left(q_1q_3 + q_0q_2\right) & 2\left(q_2q_3 - q_0q_1\right) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{3.2.8}$$
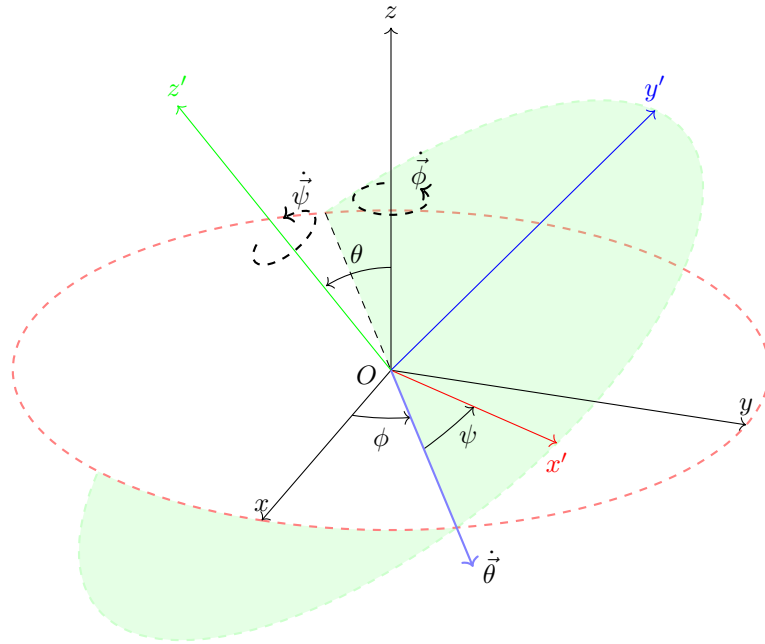
Figure 3.7 Euler Angles. Source: Own.

3.2.3 Quaternions

A quaternion q (a.k.a Euler parameters) is a singularity free attitude representation hyper complex number. To avoid confusion, the term quaternion will be used from now on. Quaternions are used in computation to overcome the singularity problem in the attitude representation and is defined as a complex number composed of a scalar real part q_0 and an imaginary three component vector $q_{1:3}$, with components spanning $\mathbb{R}^3$ defined by [24] [25]:

$$
q = \begin{bmatrix} q_0 \\ q_{1:3} \end{bmatrix} = \begin{cases} q_0 = \cos \phi/2 \\ q_1 = e_1 \sin \phi/2 \\ q_2 = e_2 \sin \phi/2 \\ q_3 = e_3 \sin \phi/2 \end{cases}
\tag{3.2.9}
$$

they represent a rotation about a unit vector through and angle ϕ, where

$$
e_1^2 + e_2^2 + e_3^2 = 1
\tag{3.2.10}
$$

and q satisfies the holonomic constraint $q^T q = 1$ which also means

$$
q_0^2 + q_1^2 + q_2^2 + q_3^2 = 1
\tag{3.2.11}
$$

Note that this constraint geometrically describes a four-dimensional unit sphere.

Quaternions can also be written (q_0, q_1, q_2, q_3). It is an element of $\mathbb{R}^4$ and mathematically not defined in

ordinary linear algebra. The hyper imaginary numbers i, j and k must satisfy the following conditions

$$
\begin{aligned}
i^2 = j^2 = k^2 &= ijk = -1 \\
ij &= k = -ji \\
jk &= i = -kj \\
ki &= j = -ik
\end{aligned}
$$

(3.2.12)

Subsequently, the transformation from Euler angles to quaternions can be calculated as

$$
q = \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} (\cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2)) \\ (\sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2)) \\ (\cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2)) \\ (\cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2)) \end{bmatrix}
$$

(3.2.13)

The inverse relation, to transform from quaternions to Euler angles is expressed as [2]

$$
\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \operatorname{atan} 2\left(2\left(q_0 q_1 + q_2 q_3\right), 1 - 2\left(q_1^2 + q_2^2\right)\right) \\ \arcsin\left(2\left(q_0 q_2 - q_1 q_3\right)\right) \\ \operatorname{atan} 2\left(2\left(q_0 q_3 + q_1 q_2\right), 1 - 2\left(q_2^2 + q_3^2\right)\right) \end{bmatrix}
$$

(3.2.14)

3.2.3.1 Time derivation of a quaternion

The calculation of the time derivative of a quaternion q in terms of the quaternion itself and the corresponding angular rate is defined by Kuipers [24] as

$$
\dot{q} = \frac{1}{2}\vec{\omega}' \otimes q = \frac{1}{2}\Omega'(\vec{\omega})q
$$

(3.2.15)

$$
\dot{q} = \frac{1}{2}q^* \otimes \vec{\omega}' = \frac{1}{2}\Xi(q)\vec{\omega}',
$$

(3.2.16)

where

$$
\Omega'(\vec{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}, \Xi(q) = \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \text{ and } \vec{\omega}' = \begin{bmatrix} \vec{\omega} \\ 0 \end{bmatrix} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ 0 \end{bmatrix}
$$

(3.2.17)

Based on prior results, the equation for calculating angular rates in terms of a rotation quaternion and its derivative is as follows:

$$
\vec{\omega}' = 2\dot{q} \otimes q^*
$$

(3.2.18)

Similarly, Kuipers c[24] proposes the second derivative of a quaternion with respect to time.

$$
\ddot{q} = \frac{1}{2}\left(\dot{\vec{\omega}}' \otimes q + \vec{\omega}' \otimes \dot{q}\right)
$$

(3.2.19)

---

[2]Four-quadrant inverse tangent which is also known as arc tangent of two numbers. $\operatorname{atan} 2(y, x)$ returns the arctangent of the two numbers x and y. It is similar to calculating the arctangent of y/x, except that the signs of both arguments are used to determine the quadrant of the result. The result is an angle expressed in radians rad. Notice how arctan and arcsin functions implemented in computer languages only produces values from $-\pi/2$ to $\pi/2$, for all the possible rotation between those values one does not obtain all possible orientations. This is why it is needed to replace arctan to $\operatorname{atan} 2$ [26].

3.3   Disturbance analysis

Any spacecraft orbiting Earth is subject to some perturbations and disturbance forces and torques. These perturbations come from various sources and their magnitude and direction depend on different physical properties of the satellite such as its weight, shape, material, etc. Whenever a force is not acting through the COM of the satellite it will result in a net torque.

Four major sources of disturbance torques shall be taken into account when designing the ADCS hardware and software:

- Aerodynamic drag (atmospheric density)
- Geomagnetic field disturbances
- Gravity gradient
- Solar radiation pressure

For LEO orbits, the ones that have a larger impact are the aerodynamic drag and the geomagnetic field disturbances in terms of torque disturbances [12]. Nonetheless, the following sections will present an overview of all these four sources of disturbances as well as analytical estimation models of them as a function of the position and orbital velocity. In all the sources, the worst-case scenario will be taken into account.

3.3.1   Aerodynamic drag

The aerodynamic drag is one of the major contributors to the disturbance torque. This is due to the number of air particles present at the altitude at which the CubeSat will be operating.

NASA's Space Vehicle Design Criteria SP-8058 [27] is used to formulate the aerodynamic drag. To compute the torque, the aerodynamic pressure acting on the surface of the CubeSat is to be integrated over the exposed forward-facing ("wetted") area of the spacecraft.

To assay how the atmosphere interacts with the spacecraft, the interaction between a body and an atmosphere through which it is moving is to be analyzed in the subsequent section. At low orbit altitudes, the interaction of the vehicle and the atmosphere can be characterized by the *free-molecular flow regime* of gas dynamics, i.e., when the molecular mean free path is much greater than a characteristic spacecraft dimension.

The incident flow is considered to be undisturbed by the presence of the spacecraft. The net aerodynamic torque can be calculated by summing up all the contributions of each spacecraft element surface. The gross value of the aerodynamic force may be obtained using the following expression:

$$D = \frac{1}{2}\rho V^2 C_d A_{\text{inc}} \tag{3.3.1}$$

ρ [kg/m^3]: is the atmospheric density at given altitude.

V [m/s]: is the spacecraft velocity also depending on the orbit altitude.

C_d [adim]: is the drag coefficient. A reference value between $2 - 2.5$ is taken for CubeSat.

A_{inc} [m^2]: is the incident satellite surface area.

The atmospheric drag is predominant for Earth orbits up to 600 km, which are the most common for CubeSat missions. In general, for spacecraft in Earth orbit, the radiation force on a given surface becomes more significant than the aerodynamic force at orbital altitudes above 1000 km. As seen previously, nanosatellites missions are launched in low circular or elliptical orbits LEO (altitudes of between 400 and 650 km) and travel at around 8 km/s. [8] [7]. Between 600 and 1000 km, torques resulting from solar radiation and aerodynamic forces are likely to be of the same order of magnitude. Nevertheless, aerodynamic torques have a direct impact on the attitude control of a spacecraft orbital lifetime.

The major drag induced by the friction with the upper atmosphere's particles will be produced in the lowest layer of the orbit's altitude. Hence, the main hypothesis used in this part is to consider the nanosatellite's orbit with a constant altitude of $h = 400$ km above the surface of Earth. Furthermore, the ISA model cannot be applied at altitudes above 100 km, however, Bowman suggests a more accurate empirical thermospheric density model of the atmosphere for altitudes ranging between 175 to 1000 km [28].

According to [27], using a simplified particle/surface interaction models, based on free-molecular flow theory. The aerodynamic force on an elemental area dA is:

$$\mathrm{d}D = \rho \left[(2 - \sigma_{\mathrm{n}} - \sigma_t) \left(v^{\mathrm{T}} n_{\mathrm{d}A} \right) n_{\mathrm{d}A} + \sigma_{\mathrm{t}} v \right] v^{\mathrm{T}} n_{\mathrm{d}A} \mathrm{d}A \tag{3.3.2}$$

where

σ_n [adim]: is the normal exchange coefficient.

σ_t [adim]: is the tangential momentum exchange coefficient.

v^{T} [m/s^2]: is the spacecraft tangential velocity.

ρ [kg/m^3]: is the atmospheric density.

$n_{\mathrm{d}A}$: is the normal vector of a surface dA (outwards the CubeSat).

As regards the momentum exchange coefficients, a good empirical values is around $\sigma_n = \sigma_t = 0.8$. The linear relationship between the dD and the density ρ in the equation above calls for an accurate atmospheric model density.

The elemental torque $\mathrm{d}\tau_D$ generated from a force dD by an elemental area dA is given by equation (3.3.3), where $r_{\mathrm{d}}A$ is the location of the centroid of the elemental area and r_{cg} is the location of the centre of mass:

$$\mathrm{d}\tau_D = (r_{\mathrm{d}A} - r_{\mathrm{cg}}) \times \mathrm{d}D \tag{3.3.3}$$

Total torque can be computed by integrating the above expression over the whole wetted surface that is affected by the aerodynamic drag using

$$D = \int_{A_{\text{wetted}}} \mathrm{d}D \tag{3.3.4}$$

Particularly, PLATHON's 1U CubeSat dimensions are equal on all sides, as it is a square CubeSat, thus, the six faces of the satellite can be considered as an elemental area with the net force acting through the centroid of each surface. Hence, the net aerodynamic torque is the result of summing all elemental forces on all surfaces where $v^T \cdot n_{dA} > 0$. However, if the Cubesat is not square, the wetted area is not the same for all orientations. Combining this explanation to equations (3.3.2) and (3.3.4),

$$D_j = \rho \left[(2 - \sigma_n - \sigma_t) \left(v^T n_j \right) n_j + \sigma_t v \right] v^T \max(0, v^T n_j) n_j A_j \quad j = \pm x, \pm y, \pm z \tag{3.3.5}$$

$$\tau_{D,j} = \sum_{j=\pm x, \pm y, \pm z} \left(r_{A_j} - r_{cg} \right) \times D_j \tag{3.3.6}$$

In the above expression, n_j indicates the normal vector of the surface in j direction, A_j is the surface facing n_j direction and $r_{A,j}$ represents the centroid of surface j.

Now, the maximum wetted area of a 1U CubeSat is 0.01 m^2. The density of the spacecraft around 400 km is $\rho =$ and the satellite's orbital velocity ranges from 7.6738 km/s and 7.5629 km/s at an orbital altitude of 600 km (see expression (3.3.7))

$$v = \sqrt{\frac{GM_E}{R_E + h}} = \sqrt{\frac{6.67408 \cdot 10^{-11} \cdot 5.9742 \cdot 10^{24}}{6371 \cdot 10^3 + 400 \cdot 10^3}} = 7673.7770 \text{ m/s} \tag{3.3.7}$$

Jacchia-Bowman Atmosphere Model [28] provides density values from the last decade (see Figures 3.8), which shows how difficult is to model Earth's atmospheric density. There is are no tabular values of density. Instead, the barometric equation and diffusion equation are integrated numerically using the Newton-Coates method to produce the density profile up to the input position. (check Appendix K for the MATLAB code):
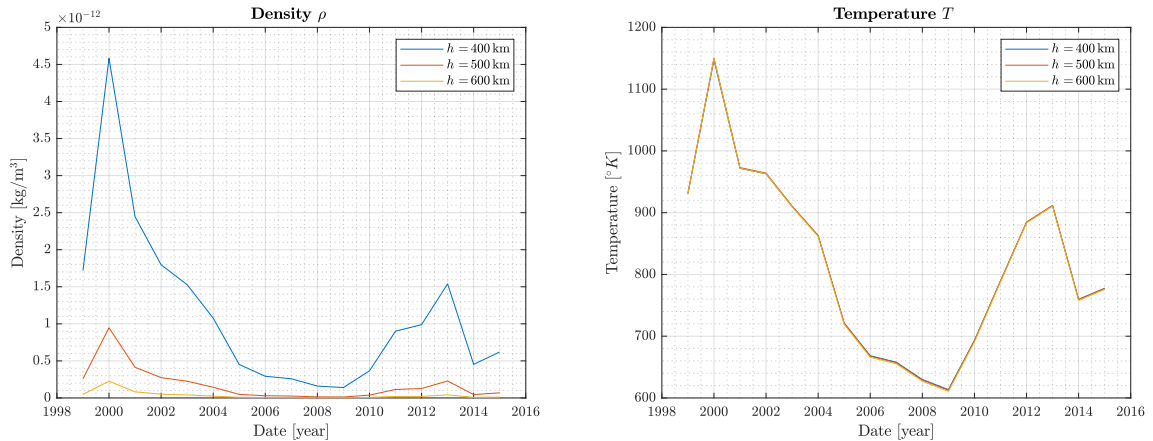
(a) Density from 1999 to 2015. Source: Own K.2.　　**(b)** Temperature from 1999 to 2015. Source: Own K.2.

Figure 3.8 Density and temperature fluctuation from 1999 to 2015 using
JB2008. Source: Own K.2.

Besides, the density fluctuates according to solar activity (solar cycles) and JB2008 takes into account those events. Higher values of density means that there are more particles in the atmosphere colliding with the satellite. Thus, according to Figure 3.8, a value of

$$\rho_{h_{400}} \approx 4.58268 \cdot 10^{-12} \text{ kg/m}^3 \tag{3.3.8}$$

will be used for the calculations, which is the maximum density at an altitude of $h = 400$ km. The maximum wetted area for a 1U CubeSat is $A_{\max} = 0.01$ m^2 with the largest offset on the centroid is approximately 0.03 m.

With these assumptions, one can estimate the aerodynamic drag using expression (3.3.5) and (3.3.6):

$$D_{\max} = 4.58268 \cdot 10^{-12} \left[(2 - 0.8 \cdot 0.8) \cdot 7673.7770 + 0.8 \cdot 7673.7770\right] \cdot 7673.7770 \cdot 0.01$$

$$= 1.0978 \cdot 10^{-7} \text{ N} \quad (3.3.9)$$

$$\tau_D = (0.05 - 0) \cdot 1.0978 \cdot 10^{-7} = \boxed{3.9939 \cdot 10^{-7} \text{ Nm}} \quad (3.3.10)$$
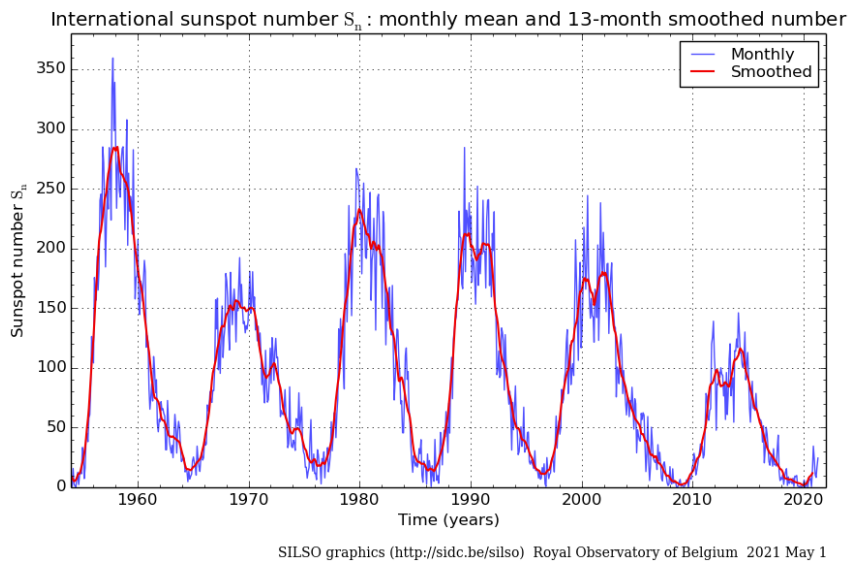
Figure 3.9 Solar activity over the past 50 years. Source: SILSO [29].

Since the value of atmospheric density is affected by altitude and solar activity, the following plot 3.10 simulates atmospheric torque as a function of height h and C_D drag coefficient in terms of solar activity. The value of the atmospheric density is presented in this model for various levels of solar activity (low, moderate, and high).

In this case, equations (3.3.1) (3.3.4) may be used to obtain the following expression:

$$\tau_D = \frac{1}{2} C_D \rho A v^2 \left(r_{\text{d}AF} - r_{\text{cg}}\right) \frac{3}{\sqrt{2}} \quad (3.3.11)$$

Since the value of C_D has yet to be determined, Wertz and Larson [30] offers a table with typical drag coefficients for various missions flown in past years, where it is seen that C_D typically fluctuates between 1 and 4 (see Appendix K for the code and JB2008 algorithm).

(a) Low Solar activity: F10.7 = 65 s.f.u.. Source: Own.

(b) Moderate Solar activity: F10.7 = 150 s.f.u.. Source: Own.

(c) High Solar activity: F10.7 = 250 s.f.u.. Source: Own.

(d) High Solar activity: F10.7 = 300 s.f.u.. Source: Own.
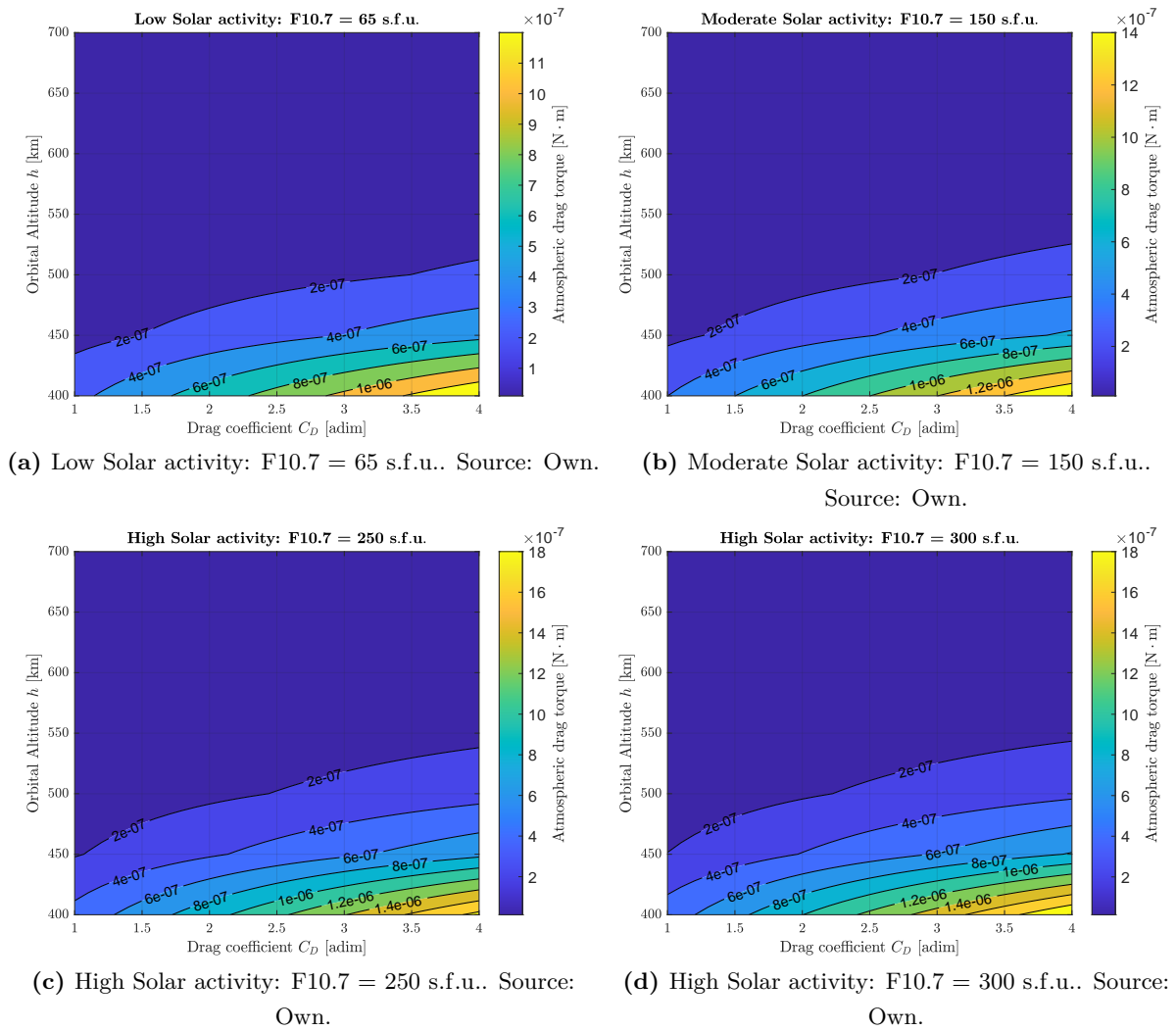
Figure 3.10 Evolution of the atmospheric torque in terms of h, C_D and the solar activity using JB2008. Source: Own.

Closer examination of the simulations in Figure 3.10 shows clearly how the aerodynamic torque increases as drag coefficient grows. Besides when solar activity F10.7 index increases it also influences the induced torque.

3.3.2   Gravity gradient torque

Any non-symmetrical object with finite dimensions in orbit is subject to a force known as gravitational torque because of the variance in the Earth's gravitational pull over the object. The inverse square gravitational force field causes this gravity-gradient torque; in a homogeneous gravitational field, there would be no gravitational torque [12].

Equation (3.3.12) may be used to calculate the torque caused by a gravity gradient. The question to be asked here is how the gravitational force affects the satellite itself since the lower part of the satellite is closer to Earth while the upper side of the satellite is further from Earth.

Newton's law shows how gravitational force decreases with the square root of the distance $1/r^2$. Although at first, this height difference may be inappreciable, a constant torque acting on a body in space where friction is considerably smaller can end up rotating the whole CubeSat upside down if not taken into consideration.

NASA [31] and Wertz [12] approach shows that gravity gradient torque can be defined as

$$\vec{\tau}_{\text{gg}} = \left(\frac{3Gm_{\oplus}}{\|\vec{r}\|^3}\right)[\hat{r} \times (I_{\text{sat}} \cdot \hat{r})] = \left(\frac{3\mu_{\text{Earth}}}{\|r\|_2^5}\right)[\vec{r} \times (I_{\text{sat}}\vec{r})] \tag{3.3.12}$$

where

μ_{Earth} $[\text{m}^3/\text{s}^2] = 3.9860 \cdot 10^{14}$ m^3/s^2: is Earth's gravitational constant.

$\vec{r}$ [m]: is the position of the satellite's COM with respect to Earth's COM.

I_{sat} [kgm^2]: is the satellite's moment of inertia matrix.

All torques that tend to upset a spacecraft's attitude must be addressed in the design of spacecraft attitude control systems. Gravity gradient is one of these torques and it derives from variations in the gravitational force over the distributed mass of the spacecraft.

Considering the worst case scenario, expression (3.3.12) can be simplified as:

$$\tau_{\text{gg,max}} = \frac{3\mu_{\text{Earth}}}{2R_{\text{Orbit}}^3}\left(I_{\text{sat,max}} - I_{\text{sat,min}}\right) \cdot \sin(2\theta_{\text{dev}}) \tag{3.3.13}$$

in which $I_{\text{sat,max}}$ and $I_{\text{sat,max}}$ are the maximum and minimum moment of the inertia; θ_{dev} is the maximum deviation of the Z axis and therefore is maximum at $\theta_{\text{dev}} = 45°$ in body frame from its local vertical (refer to section 3.2.1) and R_{Orbit} represents the orbital altitude, in which $R_{\text{Orbit}} = R_{\text{Earth}} + h$.

The inertia of the satellite can be deduced using expression (3.4.5), since it is not a massive cube the following subtraction must be made (considering solely the frame with the total components' mass):

$$I_{\text{sat,min}} = m_{\text{sat}}\frac{d_{\text{out}}^2 - d_{\text{int}}^2}{6} = 1.3 \cdot \frac{0.01^2 - 0.008^2}{6} = 6 \cdot 10^{-6} \text{ kg} \cdot \text{m}^2 \tag{3.3.14}$$

For the maximum inertia, let's suppose the satellite is has some mass shifted outside the center of mass (where m_{shift} can be some electric component such as the battery):

$$I_{\text{sat,max}} = m_{\text{sat}}\frac{d_{\text{out}}^2 - d_{\text{int}}^2}{6} + m_{\text{shift}} \cdot d_{\text{shift}}^2 = 1.3 \cdot \frac{0.01^2 - 0.008^2}{6} + 0.3 \cdot 0.02^2 = 1.278 \cdot 10^{-4} \text{ kg} \cdot \text{m}^2 \tag{3.3.15}$$

where m_{sat} is the satellite's mass and d represents the side length of the CubeSat. Since it is a 1U CubeSat, the Inertia is the same for all principal axis. Considering the worst scenario case, the CubeSat's maximum weight shall be around 1.3 kg

Accounting for the following parameters:

$$I_{\text{sat,max}} = 2.957 \text{ kgm}^2$$
$$I_{\text{sat,min}} = 2.899 \text{ kgm}^2$$
$$R_{\text{Earth}} = 6371.000 \text{ km}$$
$$h = 400 \text{ km}$$
$$R_{\text{Orbit}} = 6771.0000 \text{ km}$$
$$\theta_{\text{dev}} = 45^{°}$$

$$(3.3.16)$$

which results in a maximum gravitational torque of

$$\tau_{\text{gg,max}} = \frac{3 \cdot 3.9860 \cdot 10^{14}}{2 \cdot (6771 \cdot 10^3)^3} \left(1.274 \cdot 10^{-4} - 6 \cdot 10^{-}6\right) \cdot \sin(2 \cdot 45°) = \boxed{2.3594 \cdot 10^{-10} \text{ Nm}} \qquad (3.3.17)$$

Since the CubeSat is symmetric, it's gravity gradient contribution shall shall not affect too much.

3.3.3 Geomagnetic Field

The magnetic torque results from the interaction between the magnetic properties of the spacecraft and the ambient magnetic field (the Earth's magnetic field, in the case that applies to this project), is the subject of section [32].

Sources of magnetic disturbance torques to be considered should include but should not be limited to:

- Spacecraft magnetic moments: permanent magnetism in the spacecraft.
- Eddy currents: spacecraft generated current loop.
- Hysteresis: Magnetism induced by external fields or currents induced by external fields, i.e., when magnetic induction lags behind the magnetizing force.

For missions conducted within the Earth's magnetosphere (which is approximately 10 times Earth's radius from the geocenter), the magnetic field can be described by:

1. A **steady-state** nominal analytical model that defines magnitude and direction as a function of location with corrections distortions.

2. **Distortions** in the field caused by influences from outside the magnetosphere (effects of solar plasma) not accounted for by the nominal model.

3. **Time dependent disturbances** that alter and are superimposed on the model.

For missions that go beyond the Earth's magnetosphere, a value of 40 nT in an arbitrary direction is an appropriate upper-bound characterisation of the interplanetary magnetic field at about 1 AU for missions performed beyond the Earth's magnetosphere. With increasing distance from the Sun, the influence of Earth's magnetic field weakens.

The mathematical model of Earth's geomagnetic field is given by Wertz [12]:

$$B_{\text{Earth}} = \frac{R_{\text{Earth}}^3 H_0}{\vec{r}^3} \left[3\left(\hat{m}_{\text{north}} \cdot \hat{r}\right) \hat{r} - \hat{m}_{\text{north}}\right] \qquad (3.3.18)$$

where

$R_{\text{Earth}}^3 H_0 \ [\text{Tm}^3]$ represents the strength of the geomagnetic moment.

$\vec{r}^3 \ [\text{m}]$: is the position of the satellite in ECEF reference frame.

$\hat{m}_{\text{north}}$: is the normal vector representing the position of the north geomagnetic pole (which is updated by IAGA (International Association of Geomagnetism and Aeronomy) every 5 years [33]).

R_{Earth} [m]: denotes the mean Radius of Earth.

$$H_0 = \sqrt{\left(g_1^0\right)^2 + \left(g_2^1\right)^2 + \left(h_1^1\right)^2} \tag{3.3.19}$$

where g_n^m and h_n^m are called Gauss coefficients which are functions of time and are conventionally given in units of nano-Tesla [nT]. and the most recent values found in [33] are:

$$\begin{aligned}
g_1^0 &= -29442 \cdot 10^{-9} \text{ T} \\
g_1^1 &= -1501 \cdot 10^{-9} \text{ T} \\
h_1^1 &= 4797.1 \cdot 10^{-9} \text{ T}
\end{aligned} \tag{3.3.20}$$

Subsequently, the average geomagnetic moment is calculated using the preceding values in equation

$$R_{\text{Earth}}^3 H_0 = 7.7238 \cdot 10^{15} \text{ T} \cdot \text{m}^3 \tag{3.3.21}$$

The interaction between the Earth's magnetic field and any other magnetic field created inside the satellite produces the magnetic disturbance torque. Eddy currents, hysteresis, and the magnetic dipole of the satellite's structure are the three fundamental processes that lead to the formation of magnetic torques. However, the torque generated by Eddy currents is insignificant in magnitude when compared to the torque generated by the CubeSat's internal structure [32].

The instantaneous magnetic disturbance torque $\vec{\tau}_{\text{m}}$ is the vector cross-product of the spacecraft effective dipole moment $\vec{M}$ and the local magnetic induction or flux density $\vec{B}$:

$$\vec{\tau}_{\text{m}} = \vec{M}_{\text{sat}} \times \vec{B}_{\text{Earth}} \tag{3.3.22}$$

where

M_{Earth} [Am^2]: satellite's local magnetic induction.

$\vec{B}_{\text{Earth}}$ [Wb/m^2] or [T]: Earth's magnetic field.

The maximum magnetic torque occurs in a polar orbit (when the inclination is 90 °). Therefore, expression (3.3.18) can be simplified as,

$$B_{\text{Earth,max}} = \frac{2 \cdot R_{\text{Earth}}^3 H_0}{\vec{R}_{\text{Orbit}}^3} \tag{3.3.23}$$

For values of

$$\begin{aligned}
R_{\text{Earth}}^3 H_0 &= 7.7238 \cdot 10^{15} \text{ T} \cdot \text{m}^3 \\
R_{\text{Earth}} &= 6371.0000 \text{ km} \\
h &= 400 \text{ km} \\
R_{\text{Orbit}} &= 6771.0000 \text{ km}
\end{aligned} \tag{3.3.24}$$

and (3.3.22) can be reduced to

$$\tau_{\text{m,max}} = M_{\text{sat}} \cdot B_{\text{Earth}} \tag{3.3.25}$$

This way, replacing the above values into equation (3.3.23) As for the moment there is no data available for PLATHON's CubeSat, the magnetic moment shall be as:

$$\vec{M}_{\text{sat}} = \left[\frac{1}{\sqrt{3}} \, \frac{1}{\sqrt{3}} \, \frac{1}{\sqrt{3}}\right]^T \text{ Am}^2 \quad \implies \quad M_{\text{sat}} = 1 \text{ Am}^2 \tag{3.3.26}$$

Thus, the maximum torque due to the geomagnetic field is

$$\tau_{\text{m,max}} = M_{\text{sat}} \cdot \frac{2 \cdot R_{\text{Earth}}^3 H_0}{\vec{R}_{\text{Orbit}}^3} = 1 \cdot \frac{2 \cdot 7.7238 \cdot 10^{15}}{(6771 \cdot 10^3)^3} = \boxed{4.9763 \cdot 10^{-5} \text{ Nm}} \tag{3.3.27}$$

3.3.4   Solar radiation pressure

Radiation incident on a spacecraft's surface generates forces that may cause a torque about the spacecraft's mass centre. Surface characteristics are dominant among the factors causing both the radiation and aerodynamic torques that act on a spacecraft [34].

The direct solar illumination is the primary source of radiation force. Earth-reflected sunlight and infrared emission from the Earth and its atmosphere are additional sources for spacecraft in Earth orbit. Asymmetrical emission of electromagnetic energy (typically heat or radio signals) from onboard the spacecraft should also be considered as a radiation source.

Major factors in the determination of radiation torques are

- The intensity, spectrum, and direction of the incident or emitted radiation.
- The shape of the surface and the location of the Sun face with respect to the mass centre of the spacecraft.
- The optical properties of the surface upon which the radiation is incident or from which it is emitted.

Since solar irradiation varies as the inverse square of the distance between the spacecraft and the Sun, for a spacecraft in near-Earth orbit, this contribution to the radiation force is essentially independent of altitude. Because most other disturbance torques tend to diminish with increasing altitude, radiation torque is most likely to be a significant factor in the design of spacecraft with a large surface area that operate at orbital altitudes above about 1000 km.

NASA's Space Vehicle Design Criteria on Radiation torques provides a good approximation of the force from solar radiation pressure acting on an illuminated elemental area $\mathrm{d}A$. The assumption is made that all the incident radiation is partly absorbed, partly reflected diffusely, partly reflected specularly and negligible scattering occurs.

$$\mathrm{d}F_{\text{SRP}} = \frac{I_{\text{SRP}}}{c} \left\{ - \left[(1 + c_{rs}) \cos \alpha + \frac{2}{3} c_{rd} \right] \hat{n} + (1 - c_{rs}) \sin \alpha \; \hat{s} \right\} \cos \alpha \; \mathrm{d}A_{\text{inc}} \tag{3.3.28}$$

where

I_{SRP} [W/m^2] = 1396 W/m^2: is the energy per unit of time through a cross sectional unit area [34].

c [m/s] = 299792458 m/s: is the speed of light in vacuum.

c_{rd}: is the coefficient of specular diffusion.

c_{rs}: is the coefficient of diffuse reflection.

$\hat{n}$: is the unit vector normal to the incident surface $\mathrm{d}A_{\text{inc}}$.

$\hat{s}$: is the unit vector perpendicular to $\hat{n}$ co-planar to the incident ray.

α [rad]: is the angle between the incident ray and $\hat{n}$.

A_{inc} [m^2]: is the incident satellite surface area.

with

$$0 \leq c_a + c_{rd} + c_{rs} \leq 1^3 \tag{3.3.29}$$

According to [35], for a solar panel the coefficient of specular diffusion and diffuse reflection are $c_{rd} = 0.042$ and $c_{rs} = 0.168$, and it is assumed for the whole spacecraft.

Analogously, the elemental torque $\mathrm{d}T_{\text{SRP}}$ generated from a force $\mathrm{d}F_{\text{SRP}}$ on elemental area $\mathrm{d}A$ is given by

$$\mathrm{d}\tau_{SRP} = (r_{\mathrm{d}A} - r_{\text{cg}}) \times \mathrm{d}F_{SRP} \tag{3.3.30}$$

The total torque acting on the spacecraft can be computed by integrating Equation (3.3.30) over the entire spacecraft.

$$\tau_{\text{SRP}} = \int_{A_{\text{inc}}} d\tau_{\text{SRP}} \tag{3.3.31}$$

It is important to mention that the disturbance force generated by the albedo effect (the radiation reflected by Earth) is not taken into account since its contribution is way more reduced.

Thus, the worst case scenario can be computed as

$$F_{\text{SRP,j}} = \frac{I_{\text{SRP}}}{c} \left\{ -\left[(1 + c_{rs}) \cos \alpha + \frac{2}{3} c_{rd} \right] \hat{n}_j + (1 - c_{rs}) \sin \alpha \ \hat{s} \right\} \cos \alpha \ \mathrm{d}A_{\text{inc,j}} \quad j = \pm x, \pm y, \pm z \tag{3.3.32}$$

$$\tau_{\text{SRP,j}} = \sum_{j = \pm x, \pm y, \pm z} \left(r_{A_j} - r_{\text{cg}} \right) \times F_{\text{SPR,j}} \tag{3.3.33}$$

The incident area of a 1U CubeSat is 0.01 m^2. The six faces can be accounted as one elemental area and hence, the integral in can be replaced by the summation over the centroid of each face. With the following assumptions,

$$\begin{aligned} I_{sr} &= 1396 \ \text{W/m}^2 \\ c &= 299792458 \ \text{m/s} \\ c_{rd} &= 0.042 \\ c_{rs} &= 0.168 \\ \alpha &= 0^{\circ} \\ A_{inc} &= 0.01 \ \text{m}^2 \end{aligned} \tag{3.3.34}$$

the solar radiation torque yields to a maximum radiation torque of

$$F_{\text{SRP,max}} = \frac{1396}{299792458} \left\{ -\left[(1 + 0.168) \cos 0 + \frac{2}{3} 0.168 \right] \hat{n}_j + (1 - 0.168) \sin 0 \ \hat{s} \right\} \cos 0 \cdot 0.01 = 4.3958 \cdot 10^{-8} \tag{3.3.35}$$

$$T_{\text{SRP,max}} = (0.05 - 0) \cdot = \boxed{2.1979 \cdot 10^{-9} \ \text{Nm}^2} \tag{3.3.36}$$

$^3 c_a$ is the absorption coefficient

(a) Absorption. Source: Own.

(b) Reflection. Source: Own.

(c) Diffusion. Source: Own.
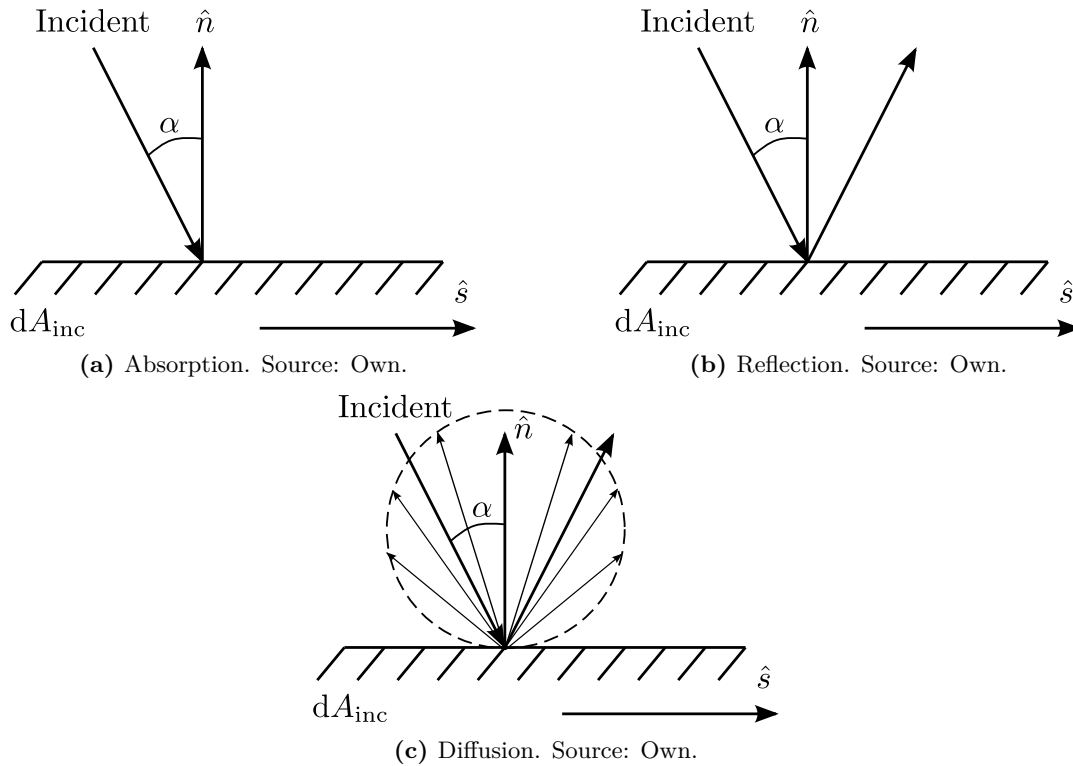
Figure 3.11 Absorption, Reflection and Diffusion over a surface. Source Own.

3.3.5 Total Disturbance

The total disturbance is the sum of all disturbances in their worst-case scenario. This comprises the worst-case scenario for all disturbance torques. As claimed by Dr. Brown [36], the satellite would be able to produce twice as much torque as the sum of all the external torques analyzed hitherto.

Table 3.2 summarizes the calculations obtained in previous sections. These results shall be taken into account when choosing the right motor and the design of the reaction wheel.

Table 3.2 Summary of external disturbance torques and their contribution. Source: Own.

	Disturbance Torque	**Magnitude** [Nm]
τ_D	Aerodynamic	$3.9939 \cdot 10^{-7}$
τ_{gg}	Gravity Gradient	$2.3594 \cdot 10^{-10}$
τ_{m}	Geomagnetic Field	$4.9763 \cdot 10^{-5}$
τ_{SRP}	Solar Radiation Pressure	$2.1979 \cdot 10^{-9}$
τ	TOTAL	$5.0165 \cdot 10^{-5}$

This contribution of external torque shall be added to the control algorithm. However, the algorithm implemented in section 7 is not considered this contribution since the CubeSat will be tested in an Air bearing. Notwithstanding, for a future mission, these disturbance torques are highly important and

should not be disregarded.

3.4 Rotational Dynamics

3.4.1 Moment of Inertia

In a rigid body rotating with an angular velocity ω, the angular momentum H_{G} computed from its center of mass [20] yields:

$$H_{\mathrm{G}} = \int \rho \times \dot{r} \, \mathrm{d}m = \int \rho \times (\omega \times \rho) \, \mathrm{d}m \tag{3.4.1}$$

where

ρ: is the density of the spacecraft.

$\mathrm{d}m$: is a differential of mass.

$\dot{r}$: is the velocity of the spacecraft.

The angular momentum H_{G} can also be expressed with respect to the moment of inertia as

$$\vec{H} = [I]\vec{\omega} \quad \rightarrow \quad \begin{pmatrix} H_x \\ H_y \\ H_z \end{pmatrix} = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ I_{yx} & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{3.4.2}$$

Eventually, the components of the Inertia tensor about the center of mass are:

$$\begin{array}{lll} I_x = \int \left(y^2 + z^2 \right) \, \mathrm{d}m & I_{xy} = -\int xy \mathrm{d}m & I_{xz} = -\int xz \mathrm{d}m \\ I_{yx} = -\int yx \mathrm{d}m & I_y = \int \left(x^2 + z^2 \right) \mathrm{d}m & I_{yz} = -\int yz \mathrm{d}m \\ I_{zx} = -\int zx \mathrm{d}m & I_{zy} = -\int zy \mathrm{d}m & I_z = \int \left(x^2 + y^2 \right) \mathrm{d}m \end{array} \tag{3.4.3}$$

If ij plane is a plane of symmetry of the body, then $I_{ij} = I_{ji} = 0$. The inertia of a mass expresses:

- How the mass of a rigid body is distributed
- How the body responds to applied torques

$$\frac{1}{12}m \begin{bmatrix} \left(a^2 + l^2 \right) & 0 & 0 \\ 0 & \left(b^2 + l^2 \right) & 0 \\ 0 & 0 & \left(a^2 + b^2 \right) \end{bmatrix} \tag{3.4.4}$$

However, in the case of a uniform density cube, the above expression can be simplified as:
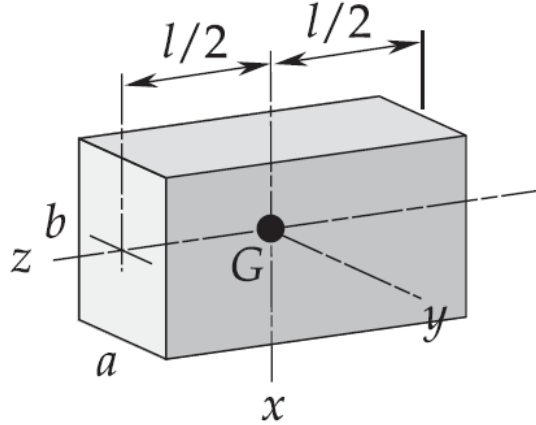
Figure 3.12 Moments of inertia for ta rectangular parallelepiped. Source [20].

with d side and mass m_{sat}, the inertial matrix for a cube results as:

$$I = m_{\text{sat}} \frac{d^2}{6} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4.5}$$

Notice that the diagonal terms are much higher than non-diagonal ones, then the homogeneous uniform density assumption would not be correct.

3.4.2 Dynamics for Satellite Model

The subsequent section derives the dynamics and kinematics of the satellite. The description of the dynamics will employ Newton-Euler formulation where the angular momentum changes as a function of applied torque.

$$H = r \times p \tag{3.4.6}$$

where H is the angular momentum, r is the position and $\vec{p} = m\vec{v}$ is the impulse.

Thereby, developing the derivative of the angular momentum one can see that

$$\frac{\mathrm{d}}{\mathrm{d}t}H = \left(\frac{\mathrm{d}}{\mathrm{d}t}r \times p\right) + \left(r \times \frac{\mathrm{d}}{\mathrm{d}t}p\right) = (v \times mv) + (r \times ma) = r \times F = \tau \tag{3.4.7}$$

where $v \times v = 0$ and it is proved that

$$\frac{\mathrm{d}}{\mathrm{d}t}H = r \times F = \tau \tag{3.4.8}$$

Recall that the angular momentum H can be expressed with respect to the moment of inertia tensor and the angular velocity vector ω as

$$\vec{H} = [I]\vec{\omega} \tag{3.4.9}$$

Now, if the CubeSat is assumed to be a rigid solid that only has reaction wheels as control actuators, the total angular momentum of the satellite in body reference H_{sat}^b yields,

$$H_{\text{sat}}^b = I\omega_{\text{sat}}^b + H_{\text{RW}} \tag{3.4.10}$$

where H_{RW} is the angular momentum generated by the reaction wheels, and ω_{sat}^b is the vehicle's angular velocity in body reference frame.

The angular momentum generated by the reaction wheel is defined as

$$H_{\text{RW}} = T_{\text{RW}} I_{\text{RW}} \omega_{\text{RW}} \tag{3.4.11}$$

where T_{RW} is the reaction wheels distribution matrix, I_{RW} expresses the inertia tensor of the reaction wheel and ω_{RW} is the angular velocity of the reaction wheel. By using Euler moment equations [20] [37].

$$\frac{\mathrm{d}}{\mathrm{d}t} H^b = -\omega^b(t) \times H^b(t) + \tau_{\text{ext,dist}}(t) \tag{3.4.12}$$

where $\tau_{\text{ext,dist}}$ encompasses all the external disturbance torques calculated in the prior section.

Eventually, joining expressions (3.4.2) (3.4.11) and (3.4.12),

$$\dot{H}^b = \left[\frac{dH}{dt}\right]^b = I\frac{\mathrm{d}\omega^b}{\mathrm{d}t} + H_{\text{RW}} \tag{3.4.13}$$

$$I\frac{\mathrm{d}\omega^b}{\mathrm{d}t} = -\omega^b(t) \times \left(I\omega^b + H_w\right) + \tau_{\text{ext,dist}}(t) - H_{\text{RW}} \tag{3.4.14}$$

Since momentum exchange of the reaction wheel and the satellite is the same but in opposite direction τ_{sat}, the above expression yields

$$\dot{H}_{\text{RW}} = -\tau_{\text{sat}} \tag{3.4.15}$$

Finally, the derivative of the angular velocity can be derived

$$\frac{\mathrm{d}\omega^b}{\mathrm{d}t} = I^{-1}\left[-\omega^b(t) \times \left(I\omega^b + H_{\text{RW}}\right) + \tau_{\text{ext,dist}}(t) + \tau_{\text{sat}}\right] \tag{3.4.16}$$

Furthermore, the calculation of the time derivative of a quaternion q in terms of the quaternion itself and the corresponding angular rate is defined by Kuipers [24] as

$$\dot{q} = \frac{1}{2}\vec{\omega}' \otimes q = \frac{1}{2}\Omega'(\vec{\omega})q \tag{3.4.17}$$

$$\dot{q} = \frac{1}{2}q^* \otimes \vec{\omega}' = \frac{1}{2}\Xi(q)\vec{\omega}', \tag{3.4.18}$$

where

$$\Omega'(\vec{\omega}) = \begin{bmatrix} 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \\ -\omega_x & -\omega_y & -\omega_z & 0 \end{bmatrix}, \Xi(q) = \begin{bmatrix} q_4 & -q_3 & q_2 & q_1 \\ q_3 & q_4 & -q_1 & q_2 \\ -q_2 & q_1 & q_4 & q_3 \\ -q_1 & -q_2 & -q_3 & q_4 \end{bmatrix} \text{ and } \vec{\omega}' = \begin{bmatrix} \vec{\omega} \\ 0 \end{bmatrix} = \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \\ 0 \end{bmatrix} \tag{3.4.19}$$

Based on prior results, the equation for calculating angular rates in terms of a rotation quaternion and its derivative is as follows:

$$\vec{\omega}' = 2\dot{q} \otimes q^* \tag{3.4.20}$$

Similarly, Kuipers proposes the second derivative of a quaternion with respect to time [24].

$$\ddot{q} = \frac{1}{2}\left(\dot{\vec{\omega}}' \otimes q + \vec{\omega}' \otimes \dot{q}\right) \tag{3.4.21}$$

Chapter 4

Reaction wheel design

The core of the thesis resides in the design of a reaction wheel that is able to be fit into the 1U Cubesat. A major source of limitation is the size and mass specifications. A secondary aim of the design was to broaden the current knowledge of inertia wheels integration in CubeSats and reduce its interference with other components, as well as preserving a minimal weight and energy consumption.

Recapitulating, reaction wheels operate by accelerating an inertia wheel in one direction and thereby forcing the satellite to rotate in the other direction by conservation of angular momentum. Three reaction wheels can be mounted in an orthogonal system providing three-axis control of a spacecraft. However, according to PLATHON research Group's needs, the first approximation was to build a 1 DOF CubeSat, thus using only 1 reaction wheel.

4.1 Theoretical basis

Oland's [38] approach seem to be well-founded. The equations given are an accurate and comprehensive methodology for designing a Reaction wheel. In sizing wheels, two performance quantities must always be considered: angular momentum capacity, and torque authority.

The intended reaction wheel was designed to minimize the inertia when it begins to spin. In a few words, the design comprises a ring along the periphery of the wheel where most of the mass is placed, which reduces the total weight while maintaining a high level of momentum. Hence, as the size and mass are the most limiting factors, the reaction wheel can be decomposed into

$$m_{\text{RW,total}} = m_{\text{RW,disk}} + m_{\text{RW,ring}} \qquad (4.1.1)$$

The mass of the reaction wheel can be found by using the expressions for mass of a cylinder and a ring as

$$m_{\text{RW,total}} = \rho \pi r_{\text{RW,disk}}^2 h_{\text{RW,disk}} + \rho \pi \left(r_{\text{RW,ring}}^2 - r_{RW,disk}^2 h_{\text{RW,ring}} \right) \qquad (4.1.2)$$

where ρ is the density of the material and r and h are the radius and height, respectively.

Since the radius and height are constrained by the dimensions of the CubeSat, the only parameter that

seems to be left with freedom of choice is the material. However, just as the other parameters, the material selection is also constrained by the maximum weight of the CubeSat.

The proposed design values for the reaction wheels are illustrated in the Table below (check Appendix H for the drawing):

Table 4.1 Reaction Wheel design parameters

Part	Parameter	Value [mm]
Disk radius	r_{disk}	60
Ring radius	r_{ring}	80
Disk height	h_{disk}	2.5
Ring height	h_{Ring}	7.5

These values were chosen according to the restrictions made by PLATHON's project requirements. To start with, the ring's maximum diameter shall not surpass 80 mm since the size of the PCB is a 9.10×9.30 cm plate. In order to maximize inertia, the ring shall concentrate all the inertia in the reaction wheel.
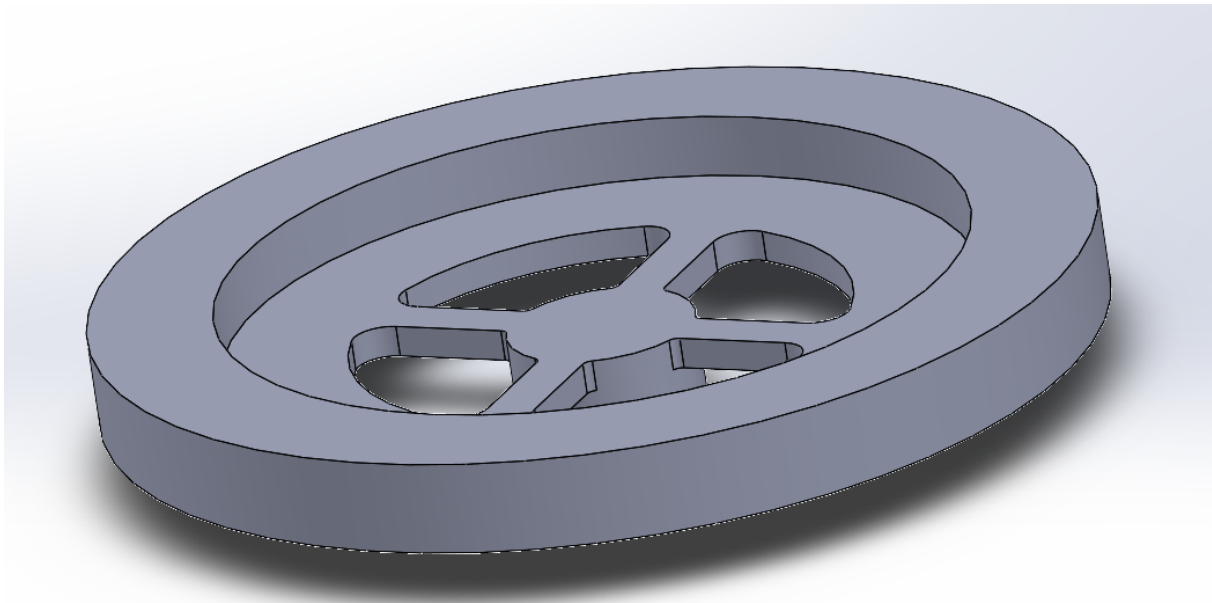
Figure 4.1 Reaction wheel CAD design. Source: Own.

Some particularities of this design are holes in the centre of the disk. These four hollow spaces were left for operational purposes. When the ring is attached to the motor, and the motor is fixed in the PCB, if the motor happens to be damaged during the testing phase. Then, it is very simple to replace the motor for a new one unscrewing the fixation parts from the PCB (see Figures 6.6 and 6.8).

The weights of the other components can be added up to be 1 kg approximately (considering the worst-case scenario).

First, the hollow part of the reaction wheel shall be calculated. This can be approximated as the difference between an outer circle and the inner circle.

$$r_{\text{hollow}} = r_{\text{outer,hollow}} - r_{\text{inner,hollow}} = 20 - 7.5 = 12.5 \text{ mm} \tag{4.1.3}$$

There are many materials that are suitable for the reaction wheel. The table below shows the main properties and differences between each (see Table 4.2).

	ABS [39]	PLA [39]	Aluminium [40]	Iron [41]
Density [g/cm^3]	1.04	1.24	2.70	7.87
Ultimate Strength [GPa]	0.04	0.065	68.3	212
Young's Modulus [GPa]	1.1-2.9	3.5	70.3	204
Maximum Service Temperature [°]	98	52	127	267
Melting point [°]	105	88-128	450	1532
Coefficient of Thermal Expansion [K^{-1}]	$73.8 \cdot 10^{-6}$	$41 \cdot 10^{-6}$	$16 \cdot 10^{-6}$	$12 \cdot 10^{-6}$
Thermal Conductivity [W/mK]	0.15	0.13	205	70
Other characteristics	Impact and heat resistant	High durability	Light and good conductor	High density and good conductor

Table 4.2 Different materials properties. Source: Various [39] [40] [41] [42] [43] [44] [45].

The material selected must have high density values so the flywheel's momentum can beat the whole satellites' momentum. The only material available in the laboratory was either to use PLA or ABS with additive manufacturing. PLA was chosen as the material for the reaction wheel since it has higher density ρ of 1.24 g/cm^3. In the future, for a 3DOF control, the CubeSat won't have enough space to fit the designed reaction wheels. Hence, other materials such as aluminium or iron should be considered to reduce volume.

The mass of the hollow is then,

$$m_{\text{hollow}} = \rho \pi r_{\text{hollow}}^2 \cdot 0.0025 = 1240 \cdot \pi \cdot 0.0125 = 1.522 \cdot 10^{-3} \text{ kg} \tag{4.1.4}$$

Thus, the mass of the disk then can be calculated as:

$$m_{\text{RW,disk}} = \rho \pi r_{\text{RW,disk}}^2 h_{\text{RW,disk}} - m_{\text{hollow}} = 1240 \cdot \pi \cdot 0.06 \cdot 0.0025 - 1.522 \cdot 10^{-3} = \boxed{0.0335 \text{ kg}} \tag{4.1.5}$$

Now, the ring's mass yields,

$$m_{\text{RW,ring}} = \rho \pi \left(r_{\text{RW,ring}}^2 - r_{RW,disk}^2 h_{\text{RW,ring}} \right) = 1240 \cdot \pi \left(0.08^2 - 0.06^2 \right) \cdot 0.0075 = \boxed{0.0818 \text{ kg}} \tag{4.1.6}$$

Finally, recovering expression (4.1.1), **maximum weight of a single reaction wheel** is found,

$$m_{\text{RW,total}} = m_{\text{RW,disk}} + m_{\text{RW,ring}} = \boxed{0.1153 \text{ kg}} \tag{4.1.7}$$

The inertial momentum of the disk is given by the following equation:

$$I_{\text{RW,disk}} = \frac{m_{\text{RW,disk}} \, r_{\text{disk}}^2}{2} = \frac{0.0335 \cdot 0.06^2}{2} = \boxed{6.03 \cdot 10^{-5} \text{ kg} \cdot \text{m}^2} \tag{4.1.8}$$

The inertial momentum of the ring is given by the following equation:

$$I_{\text{RW,ring}} = \frac{m_{\text{RW,ring}} \, r_{\text{ring}}^2}{2} = \frac{0.0818 \cdot (0.08^2 + 0.06^2)}{2} = \boxed{4.09 \cdot 10^{-4} \text{ kg} \cdot \text{m}^2} \tag{4.1.9}$$

Finally, with the known design parameters, the inertial momentum of the reaction wheel can be deduced:

$$I_{RW} = I_{\text{disk}} + I_{\text{ring}} = \boxed{4.693 \cdot 10^{-4} \text{ kg} \cdot \text{m}^2} \tag{4.1.10}$$

Since the motor used is the RF-300EA-1D390 which achieves an angular velocity of 3250 rpm, the maximum angular momentum this reaction wheel can achieve is deduced by the following expression (see Chapter 3.4.2):

$$H_{\text{RW}} = \omega_{\text{motor}} \cdot I_{\text{RW}} = 3520 \cdot \frac{2\pi}{60} \cdot 4.693 \cdot 10^{-4} = \boxed{0.173 \text{ N} \cdot \text{m}} \tag{4.1.11}$$

Then, the maximum torque generated by the reaction wheel is found as (3.4.15) (without considering external disturbance torques):

$$H_{\text{RW}} = \omega_{\text{motor}} \cdot I_{\text{RW}} \tag{4.1.12}$$

which means that the torque (i.e. the time derivative of the angular momentum) is

$$\tau_{RW} = \frac{\mathrm{d}I}{\mathrm{d}t}\omega + I_{\text{RW}}\frac{\mathrm{d}\omega_{\text{RW}}}{\mathrm{d}t} \tag{4.1.13}$$

Because the moment of inertia of a cylinder is $\frac{1}{2}mr^2$, it follows that

$$\frac{\mathrm{d}H_{RW}}{\mathrm{d}t} = m_{\text{RW}} \cdot r_{\text{ring}} \cdot v_{\text{RW}} + I_{\text{RW}}\frac{\mathrm{d}\omega_{\text{RW}}}{\mathrm{d}t} \tag{4.1.14}$$

which reduces to

$$\tau_{\text{RW}} = m_{\text{RW}} \cdot r_{\text{ring}} \cdot v_{\text{RW}} + I_{\text{RW}} \cdot \alpha = 0.1153 \cdot 0.08 \cdot 29.49 + 0 = \boxed{0.2720 \text{ N} \cdot \text{m}} \tag{4.1.15}$$

considering the Cubesat is in stationary phase (no angular acceleration $\alpha = 0$) and $v_{\text{RW}} = \omega_{\text{RW}}r_{\text{ring}} = 29.49$ m/s

The total angular rotation θ_{rot} produced by a reaction wheel within a given time

$$t_{\text{rotation}} = \frac{2 \cdot I_{\text{max}}\theta_{\text{rotation}}}{h_{\text{max}}} \tag{4.1.16}$$

This is,

Table 4.3 Angular rotation produced the reaction wheel. Source: Own.

Rotation angle θ_{rotation} [°]	Time of rotation t_{rotation} [s]
90	0.31
180	0.62
270	0.93
360	1.24

Later in Section 8, we'll see the results differ from these times since the motor has a transient phase before reaching the maximum rotation speed of 3520 rpm.

Another remark shall be noticed in the design of the Reaction Wheel design. Take a look of Figure 4.2. Notice there is a lateral hole. This lateral hole was added for a highly important issue. Whenever a piece is manufactured using additive manufacturing techniques, the final result not always is perfect and probably there might be some imperfections, i.e., leftover support material or leftover material inside drill holes. These imperfections are undesired since they will make the reaction wheel's axis to be shifted from the principal z-axis of the CubeSat's principal axis, hence, ending up with a motion lead by precession phenomena.
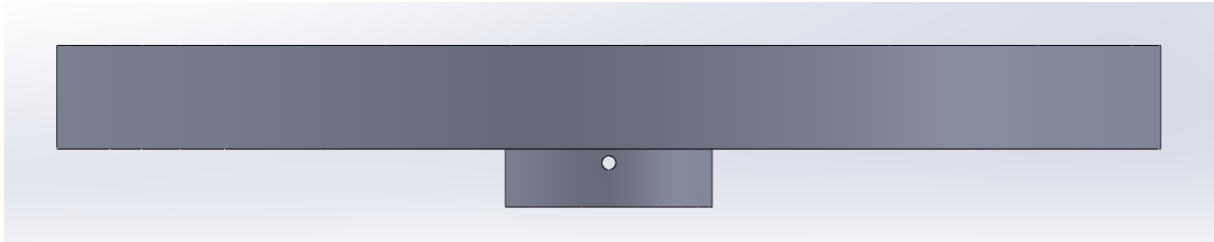
Figure 4.2 Reaction Wheel lateral hole. Source: Own.

The final printed pieces are shown in Figure 4.3,
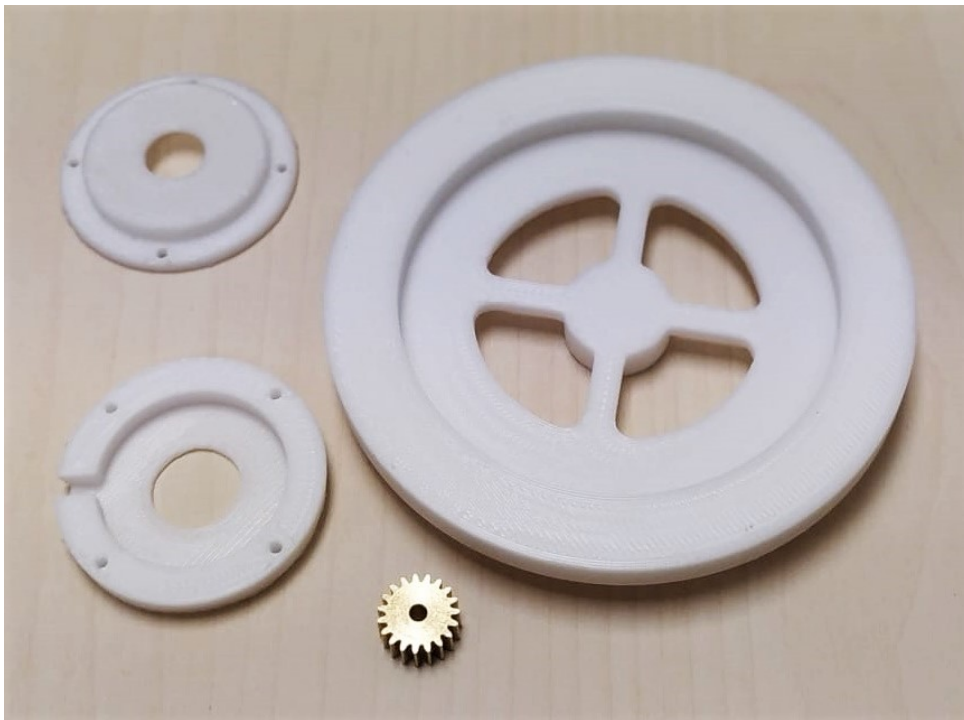
Figure 4.3 3D printed pieces (Reaction wheel and supports). Source: Own.

Chapter 5

Attitude Determination and Control Subsystem

This chapter will revise all the different components that the ADCS board will have as well as showing the assembly process of each component.

5.1 Electrical Components

5.1.1 Microcontroller

Before diving into the different hardware available, the first thing one must choose is a **MicroController Unit**. A _MicroController Unit_ (or MCU) is a compact integrated circuit designed to govern a specific operation in an embedded system. A typical microcontroller includes a processor, memory and input/output (I/O) peripherals on a single chip [46].

In simple terms, a microcontroller is similar to a computer. The difference is that a computer can perform several tasks at the same time, whereas a microcontroller, on the other hand, is typically dedicated to a single task. However, taking a deeper look at the basic components that make both the PC and a microcontroller it can be seen they share many similarities.

The fundamental components of a microcontroller are [47]:

- **Central Processing Unit (or CPU)**: The CPU is the "brain" of the computer or microcontroller and is in charge of controlling everything that the computer or the microcontroller does. In other terms, the CPU is a microprocessor with a series of associated circuitry that controls the computer's software programs. The CPU extracts each program instruction from memory and executes that instruction. After completing the next instruction, the CPU moves on to the next and in most cases, it can operate on more than one instruction at the same time (depending on the number of cores and threads). This extraction and execution are repeated until all instructions are carried

out.

- **System Clock Speed**: An essential component of any computer is its clock, this system controls the speed of the different operations the CPU handles. A microcontroller also has a system clock based on an oscillator. This is one of the essential features to consider when building projects. The oscillator's speed will determine how fast the microcontroller will run which can impact the resolution of the response or power consumption. Clock speed is measured in Hz.

- **Memory**: A microcontroller also has a memory, just like a PC has a hard drive (ROM) or RAM. The memory allows the microcontroller to store information so it can be used at a later time. Besides, just like a PC, a microcontroller's memory allows it to store programs to perform specific tasks. Memory capacity is measured in bytes (i.e. MB).

- **Peripherals**: These items are used to have bidirectional communication with the microcontroller. A mouse, keyboard or any sensor module are used to input information. For instance, a monitor ca the output of the system. Peripherals on a microcontroller are usually associated with a particular pin on a package.

  The most common peripheral interfaces for microprocessors are USB, High-Speed Internet or UART. However, a microcontroller uses interfaces such as I^{2}C, SPI or UART.

Apart from the above, there are other supporting elements a microcontroller usually includes [46] [47] such as:

- **Analog to Digital Converter (ADC)**: An ADC is a circuit that converts analogue signals to digital signals. This device allows the processor at the centre of the microcontroller to interface with external analogue devices, such as sensors. In other words, an ADC performs a conversion from analogue (continuous, infinitely variable) signals to digital (discrete-time, discrete-amplitude) signals. It does this by mapping the continuous set of values to a smaller (countable) set of values by rounding or interpolating. Thus, this conversion will always involve some noise.

- **Digital to Analog Converter (DAC)**: A DAC performs the inverse function of an ADC and allows the processor at the centre of the microcontroller to communicate its outgoing signals to external analogue components.

- **System bus**: The system bus is the connective wire that links all components of the microcontroller together where data is sent and received.

- **Serial port**: The serial port is one example of an I/O port that allows the microcontroller to connect to external components. Data travels through the serial port.

There is a common misconception between microprocessors and microcontrollers. Briefly, microcontrollers can be said to function on their own, with a direct connection to sensors and actuators and incorporating RAM and ROM memories as well as the CPU in a single place. Whereas microprocessors are designed to maximise the compute power on the chip with internal bus connections to support hardware such as RAM and serial ports (rather than direct I/O) [46].

Parameters	Microprocessors	Microcontrollers
Storage	Hard Disk (128 GB to up to 2 TB)	Flash memory (32 kB up to 2 MB)
Internal Structure	Memory and I/O devices connected externally	CPU, memory and IO devices connected internally
Clock Speed	High clock speed (1 GHz to 4 GHz)	Lower clock speed (1 MHz up to 300 MHz)
Memory (RAM)	512 MB to 32 GB	2 kB up to 256 kB
Power Consumption	High	Low
Cost	High	Low
Bit Size	It is available in 32-Bit and 64-bit.	It is available in 8-bit, 16-bit, and 36-bit.
Peripheral Interfaces	USB, High Speed Internet, UART, etc.	I^2C, SPI, UART, etc.
Applications	Gaming, web browsing, document writing, etc.	Specific tasks (i.e. camera, washing machine, etc.)

Table 5.1 Comparison between Microprocessors and Microcontrollers.
Source: Components101 [48].

There are different microcontrollers and microprocessors in the market. The most famous ones are the Raspberry Pi microprocessor, Arduino microcontroller and STM32 microcontroller. These devices are widely used for different purposes.

(a) Arduino Nano. Source: [49].   **(b)** Raspberry Pi. Source: [50].   **(c)** STM32 Bluepill. Source: [51].
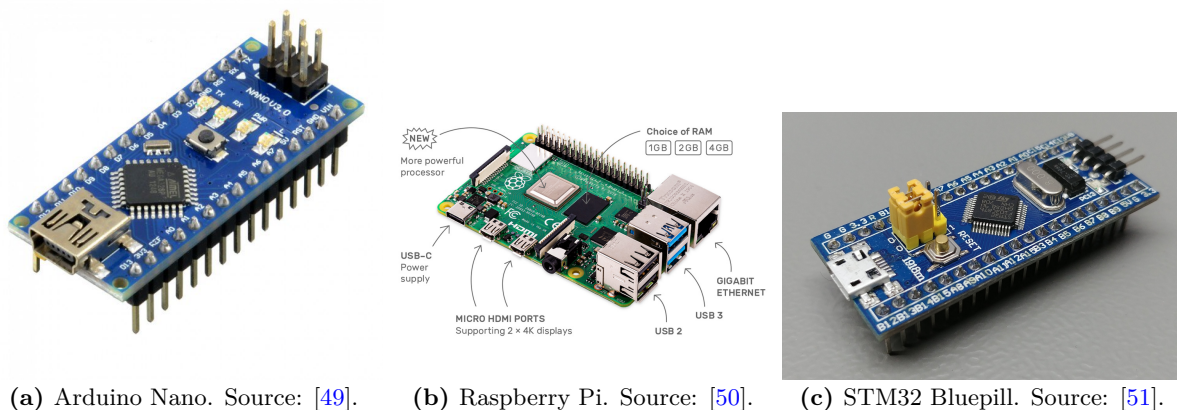
Figure 5.1 Microcontrollers and Microprocessors. Source: Arduino [49],
Raspberry Pi [50], STM32 [51].

When selecting the microcontroller for the project, several aspects were taken into account. First and foremost, the microprocessor must fit into a PCB and leave room for other components. Even though the Raspberry Pi is by far the most powerful option, the size and weight of both the Arduino Nano and

Bluepill are most suitable for the CubeSat. Therefore, the comparison was restricted to these two boards.

The Arduino Nano is one of the most popular microcontrollers on the market. It offers a large amount of flexibility and purposes. Since this is from Arduino's family, it uses the Arduino IDE, one of the comprehensive open-source interfaces for microcontroller programming. The built-in microprocessor is the ATmega328, which is highly powerful for the initial phase of the project (see Appendix B for more detailed information).

Conversely, on the other hand, the Bluepill microcontroller shares a similar dimension to the Arduino Nano board. This microcontroller, though, has a slightly bigger dimension than the Arduino Nano board but offers faster clock speed numbers, RAM and flash memory. Since ST-Microelectronics is the manufacturer of this board it uses its own interface. Nevertheless, this board can be programmed to work with the Arduino IDE using an ST-Link device and a custom library (see Appendix B).

While the Arduino is powered via 5 V, thus, it will need a logic level converter to convert the 3.3 V to 5 V and vice-versa to work with the motor driver as this latter one functions with 3.3 V.

Besides, it is important to highlight the usage of the pins. Since the board will be connected to multiple devices at the same time it must ensure to have a sufficient amount of pins to manage the communication with the On-Board Computer and also the actuators. Later, in section 5.1.3 it is shown how the drivers use I^2C protocols and also the communication with the OBC. To avoid disturbances and perturbations, it is highly recommended to use different channels for communication and actuators. As seen in Figure B.2, there are multiple I^2C pins with different channels. In the case of the Inertial Measurement Unit, while the MPU9250 can make use of the SPI protocol and I^2C, the BNO055 is limited to work only with I^2C.

Finally, balancing the advantages and disadvantages of both boards, the STM32 Bluepill (see Figure 5.1 (c)). microcontroller suits best for the needs of the project. It has more communication pins as well as faster clock speed, namely, faster computing speeds. This microcontroller has been programmed using Arduino IDE with ST-Link device (check Appendix B to see how it has been programmed.)

Table 5.2 Comparison between Arduino and Bluepill. Source: Arduino and Bluepill [49] [51]

	Arduino Nano	**Bluepill**
Manufacturer	Arduino	ST-Microelectronics
Microcontroller	ATmega328	STM32F103C8T6
Clock Speed	16 MHz	32 MHz($72 MHz max$)
Flash Memory	32 kB	64 kB
SRAM	2 kB	20kB
Operating Voltage	5V	3.3 V
Analog IN Pins	8	10
Digital I/O Pins	22 (6 have pwm	29 (10 have pwm)
I2C buses	1	2 + 1 (I2C2)
SPI buses	1	3
Size	18×45 mm	23×53mm
Weight	7g	20 g

5.1.2 Inertial Measurement Unit

An Inertial Measurement Unit is a device in charge of obtaining the attitude data for the ADCS. As shown in section 2.2.2, several devices are used to retrieve the attitude data. Moreover, these devices are selected depending on the needs in each situation and missions usually holds more than one attitude determination instrument for precision purposes.

The IMU is perhaps the core of the ADCS subsystem. It contains sensors such as accelerometers, magnetometers, gyroscopes and even thermometers. To increase the reliable output movement, IMUs collects data from numerous distinct sensor:

- **Accelerometer**: A device that monitors velocity and acceleration.
- **Gyroscope**: determines rotation and rotational rate.
- **Magnetometer**: identifies cardinal directions (directional heading).

However, despite the corrections and all the sensors, IMU's have a majors disadvantage that has to be taken into account. The main downside of an IMU is that it is prone to error accumulation over time, which is also known as often known as "drift" (check tests in Section 8). This happens because the IMU is continually monitoring changes relative to itself (rather than triangulating against an absolute or known outside device), it continuously rounds off tiny fractions in its computations, which add up over time. If left unaddressed, these minor inaccuracies might compound up to large mistakes. To address this problem, one way of solving it is to use other attitude instruments as a complement to constantly correct the tiny errors that may appear.

Two different IMUs were studied for PLATHON's project, these two IMUs are: the BNO055 and the GY-9250 module which incorporates the sensor MPU9250. To analyze them, magnetorquer's team studied MPU9250 while the reaction wheels team performed an analysis with BNO05.
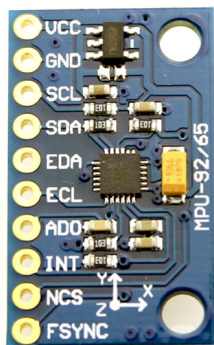
(a) GY-9250 module with MPU-9250 sensor. Source: Components101 [52].
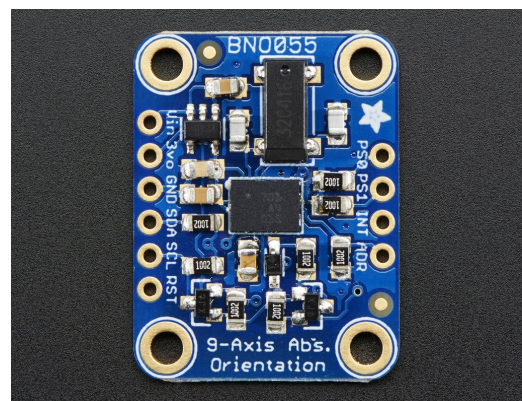
(b) BNO055 module sensor. Source: Adafruit [53].

Figure 5.2 Inertial Measurement Units. Source: [52] [53].

The first IMU, BNO055 from Adafruit was selected for the capabilities it provided. This sensor has 9-DOF and offers a fusion technique that combines accelerometer, magnetometer, and gyroscope data

to provide consistent three-axis orientation output. By combining a Micro Electrical-Mechanical System (MEMS) accelerometer, magnetometer, and gyroscope on a single built-in die with a high-speed ARM Cortex-M0 processor to process all sensor data, the abstraction of the information in quaternions, Euler angles or vectors are just extremely fast and accurate. One small downside of the BNO055 relies upon the fact it only operates with I²C communication protocols, which later in the project will the reason to shift to an alternative IMU that has SPI interface.

As the project moved on, the team found that there was a crucial problem with respect to the drivers and microcontroller. To summarize, the problem was due to the coupling of channels on the I²C address of the IMU and the drivers. Several tests were done to find out where the problem resided. In the end, the team opted for another IMU with SPI interface.

The second tested IMU was the GY-9250 module. Similar to the prior one, the MPU9250 sensor has 9-DOF and both SPI and I²C protocols. Just as the BNO055, this IMU has a built-in Digital Motion Processor (DMP) which makes the IMU able to make calculations and corrections and the measurements instantaneously (check Appendix C for details).

5.1.3   Motor Driver

The Cubesats need to generate rotational inertial to orientate in space. As seen before, one way of doing such control is using reaction wheels. Nevertheless, reactions wheels alone are just a mass with a certain amount of inertia and thus, it does not have the capacity of generating a moment of inertia. To do so, it is needed a motor driver.

A **motor driver** is a device that is used to control the power and therefore, the velocity of a motor. The principle behind a motor driver can be simplified as a device that turns a low current signal into a high signal for driving the motor. The main reason why is that motors cannot be connected directly to a microcontroller pin is that they do not provide sufficient power from those pins. For instance, an Arduino nano standard 3.3 V or 5, 5 V pin can output only a maximum of 50 mA of current [54] which is insufficient for most of the motors that are used in different applications, especially, on startup and stall currents. Hence, the main goal is to drive that larger load from the small signal of the microcontroller and amplify that to be able to control a much larger load. The motor driver acts like a gateway where it accepts this small input and can control larger loads using a special type of transistors called MOSFET (*Metal–Oxide Silicon Transistor*). The MOSFET transistor acts as a switch, when a current is applied onto the base connector, the transistor lets the current flow through it. In fact, MOSFETs are usually integrated within the motor driver.

It's important to remark that when referring to motor drivers, the motor itself must be taken into account. A brushed DC motor does not require any control or timing whereas servo motors require a specific type of signal to operate correctly [55].

The correct way of controlling a motor is to connect the motor driver to the motor first and then supply power to the motors using an external battery through the motor driver. Besides, the most common form of speed control systems is the **H-bridge**.

As a brushed motor always rotates in one direction, it is mandatory to provide the motor with the ability

to rotate both in a clockwise direction and counterclockwise direction. Precisely, an H-bridge system configuration permits the motor to rotate in either direction. The motor driver selected for this project uses this configuration (see Figure 5.3).
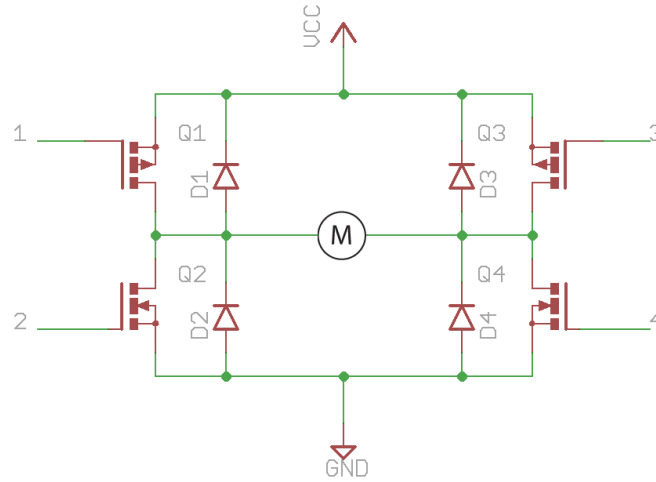
Figure 5.3 H-bridge configuration schematic. Source: Core Electronics [55]

The electrical schematic (Figure 5.3) shows an H-bridge with two pairs of MOSFETs to control the flow of the electrical current through the motors. Consequently, switching on and off a specific combination of MOSFETs will permit the rotor to rotate in the desired direction. Notice how two pairs of diodes are added as well, these diodes restricts the flow in a certain direction and are commonly added for safety issues to protect against transient current and voltages produced by the motor as it turns. Depending on the application, the H-bridge can have another set of variants with more components. In the Figure above, the additional resistors were left out to make it easier to understand.
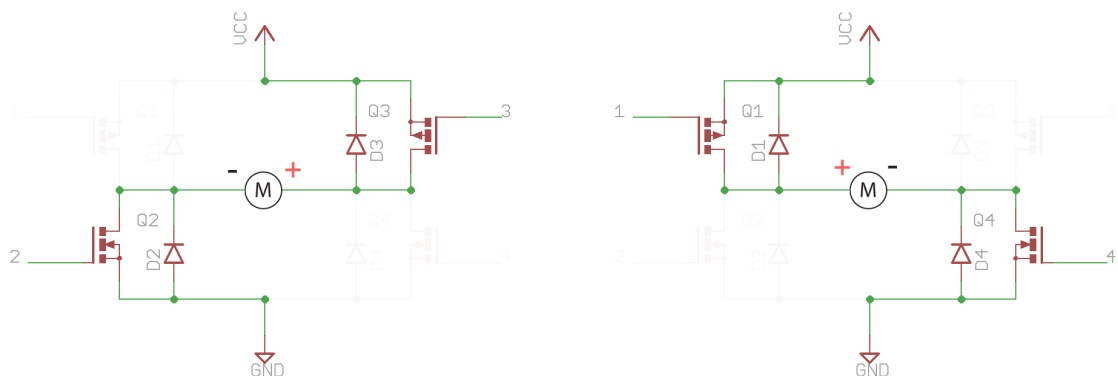
(a) H-bridge configuration schematic in clockwise direction. Source: Core Electronics [55]

(b) H-bridge configuration schematic for counterclockwise direction. Source: Core Electronics [55]

Figure 5.4 H-bridge clockwise and counter clockwise rotation direction.
Source: Core Electronics [55]

Let's look at Figures 5.4a and 5.4b, the motor is placed in the middle and the flows of the current through the motor takes one way or other depending on the MOSFETs. In the first state, the motor's positive potential is located on his right side and the lower potential is located on the left side. On the other hand,

the second scenario depicts the same flow but has interchanged the positive and negative connectors.

There is another configuration that is useful if the user desires the motor to stop rotating. Despite turning off the motor from the power, there is a remaining momentum of the rotor that prevents it to stop immediately. so it will gradually come to a complete stop. Thus, to perform a complete stop and set the rotor in a static position it is needed to brake the rotor. By applying an equal voltage to both sides of the rotor it will make sure the rotor stops its motion instantly.
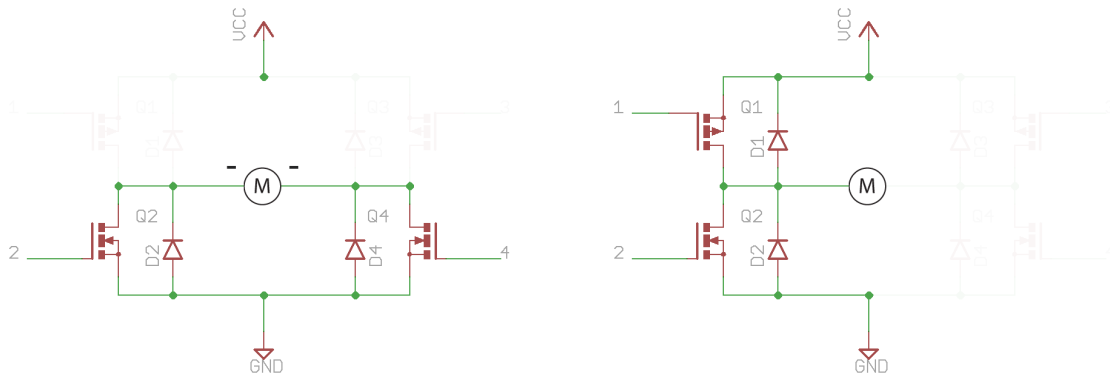
(a) Real-time Python visualization. Source: Core Electronics [55].

(b) Real-time Python visualization. Source: Core Electronics [55].

Figure 5.5 Braking configuration (left) and incorrect use-case configuration (right). Source: Core Electronics [55]

One important use-case that should be avoided at all times is shown in Figure 5.5b which will generate a direct path from voltage supply to ground. Not only this will shortcircuit out the circuit but also damage other peripherals connected to it. The same reasoning applies to the other side.

The driver that was chosen for the project was the SparkFun Qwiic Rob-15451 motor driver [56] which has an H-bridge configuration and suits the requirements of the project with its dimensions, weights and the communication protocol it has which can be either I^{2}C or SPI.
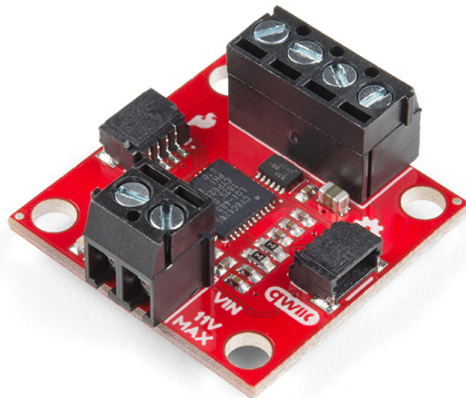
Figure 5.6 SparkFun Qwiic Motor driver ROB 15451. Source: SparkFun
[57].

The Sparkfun Qwiic motor driver will use I^{2}C communication protocol which translates to a fast, simple, high speed and cost-effective protocol for communication [58]. Furthermore, taking into account the PCB that must be less than 10×10 cm, its size (2.5×2.5 cm) and weight fits perfectly in the PCB with all the other components. It allows using two I^{2}C connections via two ports at the sides simultaneously, meaning that various drivers can be connected one to another in the same I^{2}C buses. Each driver can connect 2 devices, which translates that for a full 3-axis control of a CubeSat, it would be needed 4 drivers (3 magnetorquers and 3 reaction wheels). The driver's address can be changed by changing a pin (see Appendix G). The output voltage ranges from 3 to 11 V, and powered at 3.3 V. The size of these drivers is relatively small, measuring 1×1 in (25.4×24.5 mm) which gives more versatility, reduces weight and space.

As regards the connection of the motor to the driver, the motor will be connected directly to the two outputs it has. On the other hand, the control signal will be connected to a Qwiic connector located on the side, so no soldering is required to connect it to the rest of the system. The Qwiic connector part of the Sparkfun ecosystem and gathers the needed pins for both I^{2}C and SPI protocols. As shown later, this reduces the amount of required PCB space.

Among other properties it has, it is notable that this motor driver supports 1.2 A of steady-state drive per channel with 1.5 A peak. Since the driver is a 3.3 V logic device, is compulsory to use a logic level converter to interface to 5 V. Also, the I^{2}C address of the Qwiic Motor Driver is default to $0x5D$, however, the address can effortlessly be changed with its jumper to another address (please refer to Appendix G for more information about the documentation of the motor and the connections).

5.1.4 Motor

The motor is the fundamental piece for the project. The motor has to be able to generate a sufficient amount of torque to rotate the entire CubeSat and also offset the external disturbance torques.

The motor selection must agree with the type of driver it has to function with. There are two types of motors in the market: Brushed DC motors (BDC) and Brushless DC motor (BLDC):

- **Brushed DC motor**: The armature of a brushed DC motor is a structure of coiled wire coils that acts as a two-pole electromagnet. The commutator, a mechanical rotary switch, reverses the directionality of the current twice every cycle. As a result, the electromagnet's poles pull and push against the permanent magnets along the exterior of the motor, allowing current to flow through the armature. The commutator then switches the polarity of the electromagnet in the armature as its poles cross the poles of the permanent magnets.

- **Brushless DC motor**: This type of motors, by contrast, has a permanent magnet as its exterior rotor. Furthermore, it employs three stages of driving coils as well as a specific sensor that measures rotor position. The sensor delivers reference signals to the controller as it tracks the rotor position. In turn, the controller systematically activates the coils - one phase after the other.

Overall, both types of motors have their advantages and disadvantages. The advantages of the brushed motors are that they are more economic, simple and uses inexpensive controllers whereas the major downside is that they are usually less efficient than brushless motors as constant switching action with the commutators creates electromagnetic noise and they wear out due to the perpetual physical contact with the shaft. However, in accordance with the requirements set for this project, a BDC motor is sufficient. In the next generation of the project, a BLDC will be used.

Considerations and priority on the motor research:

- **Cost**. The primary goal of the CubeSat is to be as economic as possible, so the cost motor shall be reduced.

- **Efficiency** / Low consumption: Low consumption is essential in space missions. When in standby mode, the motor's consumption must be as reduced as possible. This translates into better performance and saves energy in case of occlusion.

- **Dimensions**. The sizing of the motor is a key factor. Although the vast types of CubeSats, the motor has to be as flat as possible and compact. The motor must leave enough space for other components in the PCB.

- **Great torque**. The torque this motor must supply shall be enough to rotate the entire CubeSat when loaded with all the components.

- **Lifespan**. For space missions, the lifespan of a CubeSat is rather small. However, the motor needs to be fully operative during the time of the mission and even more than the missions lifespan. Thereby, it shall not require any maintenance.

- **Great range of velocities**. This one has to be fulfilled otherwise it would limit CubeSat's rotation. The faster it spins, the less time it spends on manoeuvres, but it will more energy consumption in a shorter period of time. It shall have notorious responses and the intention is to reduce the control time. A greater range of velocities leaves more margin of manoeuvrability.

Two motors were analysed for the project, the first one is the Mabuchi RF-500TB-14415 and the second motor is the Mabuchi RF-300EA-1D390. The RF-500TB-14415 was studied during the initial state of the project.

The Mabuchi RF-500TB-14415 [59] was bigger in size with a diameter of $\varnothing$ 32 × 19.5 mm and a mass of 45 g. Its maximum efficiency is around 55% depending on the torque, which is expected for brushed motors. In terms of speed, it is able to rotate at incredibly high revolutions per minute, achieving a maximum of 3100 rpm without loads. Although this motor had a great amount of torque, its size was far larger than the limits of the PCB, which restricted the amount of maximum diameter available for the motor.

Figure 5.7 Mabuchi RF-500TB-14415 (left) vs Mabuchi RF-300EA-1D390
(right). Source: Own.

Next motor analysed was the Mabuchi RF-300EA-1D390 [60]. This second motor solves the size problem of the prior motor and possesses high speeds up to 4400 rpm with lower nominal voltage consumption of 3.9 V rather than 5 V. However, due to the smaller size restriction, the amount of torque it can generate also is affected and reduced to 0.47 mN · m. The size of this motor is $\varnothing$ 24 × 12.8 mm and only 22 g of mass, half the mass of the prior one. As regards the performance, the efficiency remains practically the same at around 52%. Eventually, as the bench tests demonstrated great results, this motor was selected to work with the driver.

Table 5.3 Characteristics of the two motors analyzed for the project. Source: Mabuchi [59] [60].

	RF-500TB-14415	RF-300EA-1D390
Voltage Range	$1.5 - 9.0$ V	$2.8 - 7.0$ V
Nominal Voltage	5 V	3.9V
No Load Speed	3100 rpm	4400 rpm
No Load Current	26 mA	21 mA
Maximum Efficiency	55.0%	52.5%
Max Efficiency Speed	2540rpm	3100rpm
Max Efficiency Current	120 mA	84 mA
Max Efficiency Torque	1.23 mN $\cdot$ m	0.47 mN $\cdot$ m
Stall Torque	6.86 mN $\cdot$ m	2.35 mN $\cdot$ m
Stall Current	54 mA	34 mA
Size	$\varnothing\ 32 \times 19.5$mm	$\varnothing\ 24.4 \times 12.8$ mm
Weight	45 g	22 g

Hence, the Mabuchi RF-300EA-1D390 was selected for driving up the reaction wheel. In section 5.2.3 is explained how to connect the motor to the microcontrollers.

5.1.5 Logic Level Converter

Most devices are designed to be powered at 5 V. Albeit, the newest components use the newest standards (5 V). The main advantage of it is that as the voltage is higher, noise susceptibility will be significantly lower since higher noise levels are needed to disturb the 5 V. The major downside is power consumption. See Appendix D).

The bi-directional logic level converter is a small device that securely levels down $5V$ signals to 3.3 V while simultaneously stepping up 3.3 V to 5 V which allows you to adapt the SPI, I^{2}C, UART or any digital signal. The board needs to be powered by the two voltages to be converted and only works with digital signals (high and low levels). This model does NOT work with analogue signals.
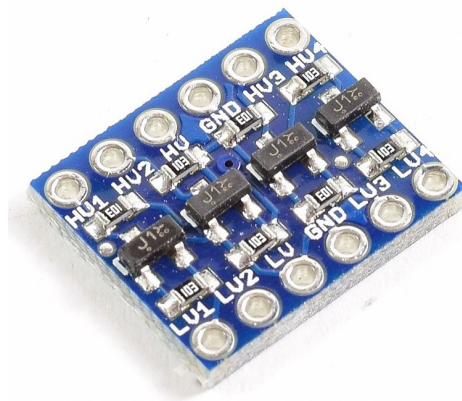
Figure 5.8 Logic level converter. Source: Naylamp Mechatronics SAC [61].
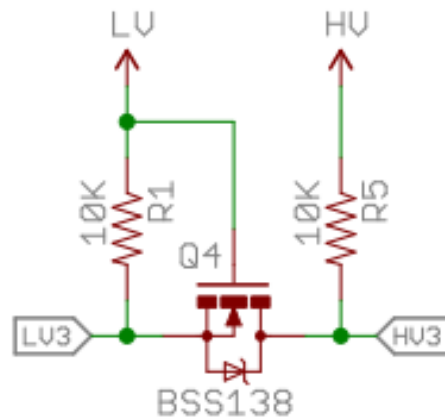
Figure 5.9 Logic level converter schematic. Source: Naylamp Mechatronics
SAC [61].

Despite Arduino Nano operates with 5 V supply, the Bluepill still operates with a 3.3 V supply. When connecting the STM32 Bluepill via USB, the USB outputs 5 V voltage. One could ask how is it possible for the STM32 Bluepill to output a 5 V supply, this is due to the fact that it has an internal logic level converter that controls the entire microcontroller.

5.1.6 Bluetooth Module

The next step is to establish wireless communication. Thus, this will be extremely useful for sending data without the need of a physical wire connection to the OBC as the final step is to test it on an air bearing and send it to outer space.

The fundamental advantage of satellite communication is that it gives communication services to any location on the planet. Besides, satellites are not affected by distance. Nonetheless, the following issues are related to satellites.

There are several methods for accomplishing wireless communication for testing purposes in the ground station, such as WiFi (Wireless Fidelity), IR (Infrared), microwave or mobile communications such as 4G or 5G. However, the design of the communication using high band antennas is out of the scope of this project. Furthermore, there are many problems related to satellite communication, including high propagation delays, poor bandwidths compared to terrestrial media, and noise due to the effect of rain and atmospheric disturbances.

Instead, to get wireless communication working, the team has decided to take advantage of Bluetooth communication, since wireless communication was a requirement of the project. The Bluetooth module HC-06 from DSD Tech will be used for telemetry and telecommunication.

The HC-06 module allows any Bluetooth device, such as a computer or a smartphone, to connect wirelessly to the Arduino Nano or the STM32 Bluepill. It has a 3.3 V supply voltage, though it is usually equipped with a regulator that allows for a supply voltage of 3.6 to 6 V.

The Bluetooth module HC-06 has 4 pins, 2 for power and 2 to establish a connection. Project PLATHON's PCB is designed to work with either the HC-06 or an HC-05. The only difference between both modules is that the HC-05 can be used as a slave module (equivalent to HC-06) and as a master module which means that it can pair itself to another device on its own. Due to this fact, the HC-05 have 2 more pins, one that enters to configuration mode and the other one to connection state (check Appendix E for a detailed data-sheet and configuration).

The communication protocol of the HC-06 is UART (Universal Asynchronous Receiver/Transmitter) which uses an asynchronous serial transmission at a variable speed. This means that unlike I^2C for SPI, there is no clock signal to synchronize the transmitting device's output bits to the receiving end.

In Bluetooth communications, there are two types of roles: host and slave. This module is a slave, which means it waits for the host to issue the two execute command. The host, on the other hand, selects the slave with whom it wishes to communicate. Finally, the host/slave, which can play both roles depending on its configuration in the code, combines these two roles.

The module is factory setup as a "Slave" device. However, using an HC-05 which the same as the HC-06 but with the added ability to act as a master as well. Since the laboratory have both of them, the PCB was designed to support both devices as they share the same pins.

5.2 ADCS board assembly

Once reviewed all the electrical components, the following section intends to show how to connect every component to the microcontroller and prepare it for testing both the hardware and control software before assembling all the pieces into a custom PCB as well as the code for its correct setup and configuration.

As explained in prior sections, the microcontroller used for this part is the Bluepill, however, several tests were performed using the Arduino board. A lot of component and code testing was done during the initial stage of the project. There are two ADCS teams (Reaction wheels team and Magnetorquer team) within the ADCS Subsystem. Both teams analysed different components and once the component testing phase was settled, the global team decided which component to use. Figure 5.10 presents the diagram of all the components the ADCS board must have and the communication protocols.
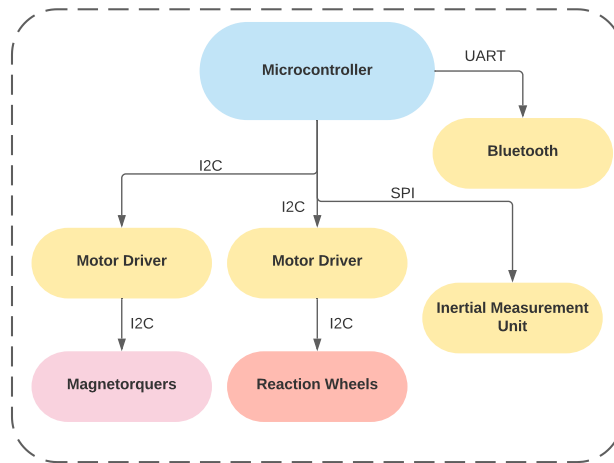
Figure 5.10 ADCS Board diagram. Source: Own.

5.2.1   The Inertial Measurement Unit

BNO055 IMU assembly

At the initial stage, the IMU tested was the BNO055 from Adafruit. This IMU was first considered since the characteristics of the Arduino board and the rest of the components have not been dealt with in-depth yet. Figure J.1 and L.1 shows the Arduino board and the BNO055 IMU.
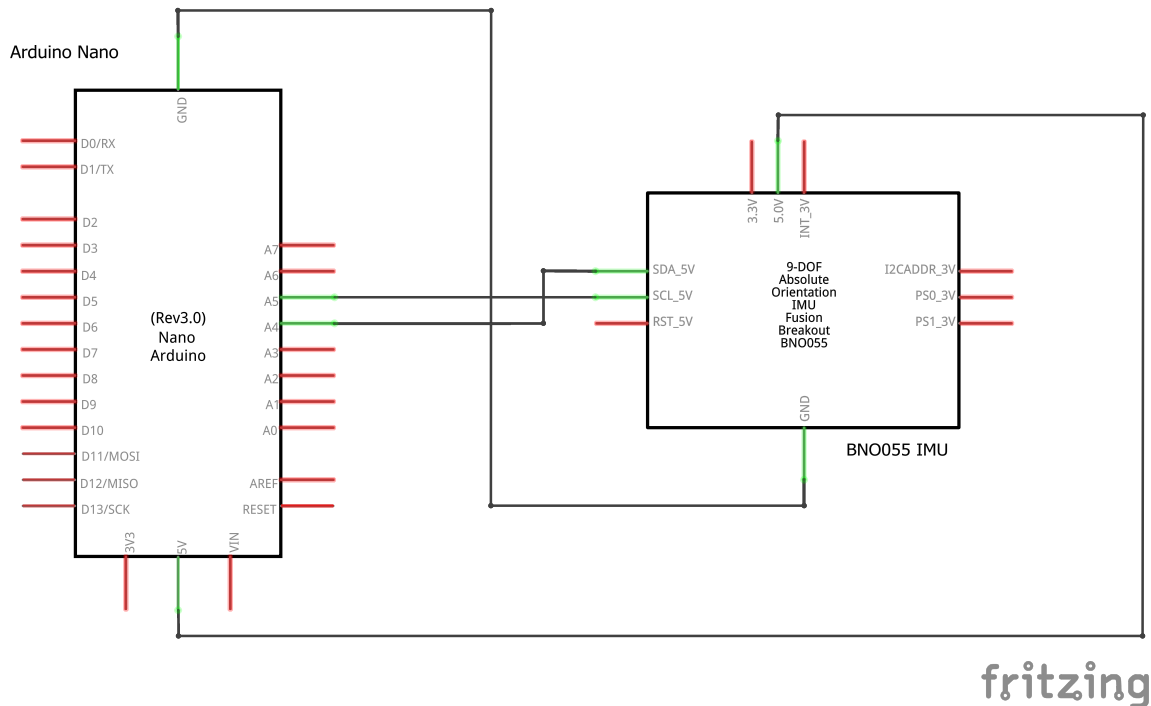
Figure 5.11 Schematic of the BNO055 and the Arduino Nano connections.
Source: Own.

As for the connections, the BNO055 uses I^{2}C communication protocol and A4 and A5 pins of the Arduino board supports this communication. The synchronization and data transmission are in separate pins, where SDA is the line for the master and slave to send and receive data and SCL refers to the serial clock port that carries the clock signal.

Table 5.4 Arduino Nano and BNO055 IMU connections. Source: Own.

Arduino Nano	BNO055 IMU
5 V	Vin
GND	GND
A4	SDA
A5	SCL

I^{2}C, like SPI, is synchronous, which means that the output of bits is synced to the sampling of bits by a clock signal shared by the master and slave. The master is always in charge of the clock signal.

Firstly, the IMU must be calibrated and configured (see Appendix L). Once the IMU is configured and calibrated, the next step is to get all the raw data from all the sensors, namely, accelerations in all three axes (accelerometer), magnetic field measurements in all three axes (magnetometer) and angular velocity (gyroscope). Note that these measurements are in the satellites body reference frame (see 3.2.1).

After this, one can proceed to calculate the Euler angles from the raw data of the sensors. Nevertheless, the IMU itself has also a built-in Digital Motion Processor (DMP) which can provide the Euler angles and even quaternions directly without any computation. The complete code for this test is detailed later in Section 8.

MPU9250 IMU assembly

The connections for the MPU9250 follows the same procedure, however, this time, an SPI communication protocol has been chosen since it is able to transfer data at faster rates. However, it is also possible to use the I^{2}C protocol. In this case, the MPU is connected via the 3.3 V. The data transfer and synchronization are done by SPI serial data ports where MOSI and MISO are the pins for sending and receiving data, SS enables the transmission data transmission circuit, SCK serves as the serial clock and finally, FSYNC is the pin for digital input frame synchronization. Below is shown the schematic and connections of the Arduino board and the MPU9250.
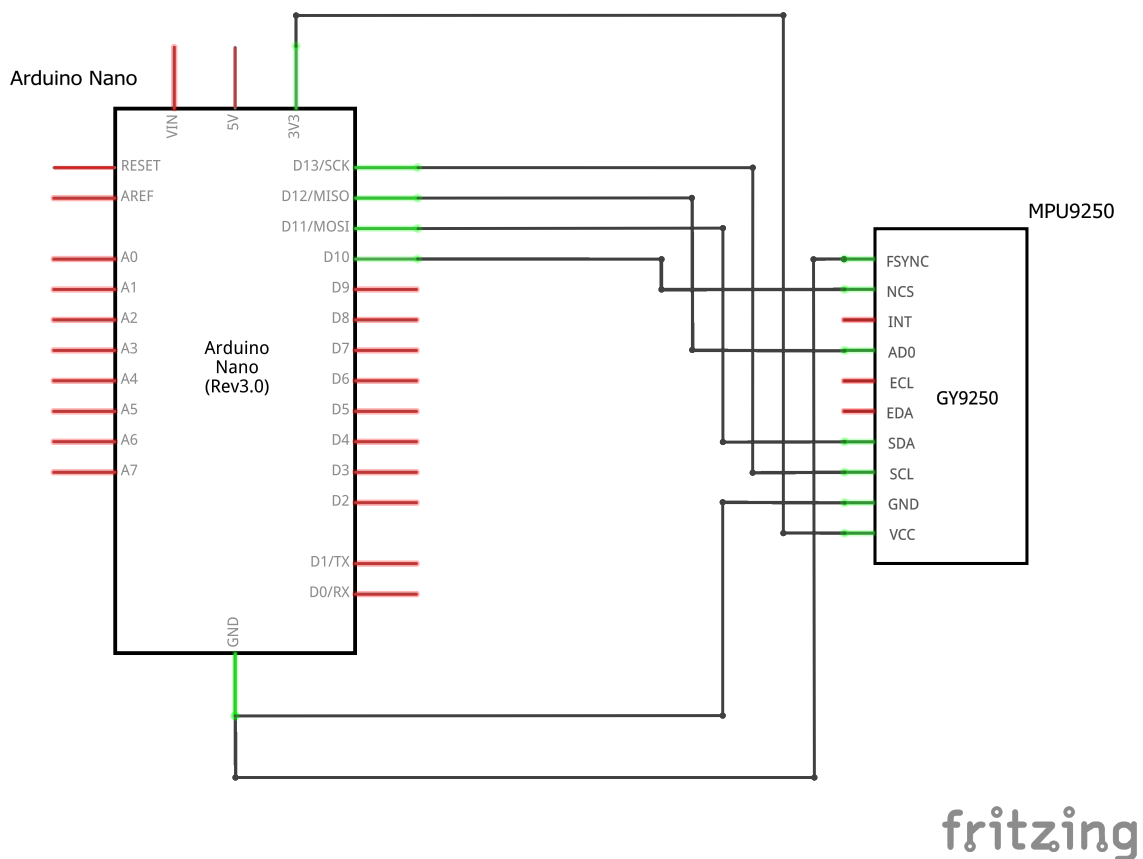
Figure 5.12 Schematic of the MPU9250 and the Arduino Nano connections.
Source: Own.

The connections are summarized in the table below 5.5:

Table 5.5 Arduino Nano and MPU9250 connections. Source: Own.

Arduino Nano	BNO055 IMU
3.3 V	VCC
GND	GND
D10 (SS)	NCS
D11 (MOSI)	SDA
D12 (MISO)	AD0
D13 (SCK)	SCL
FSYNC (SS)	GND

Analogously, the microcontroller used at the end of the project was the STM32 Bluepill. The connections remain the same but the pins for the data transmission protocol are different.
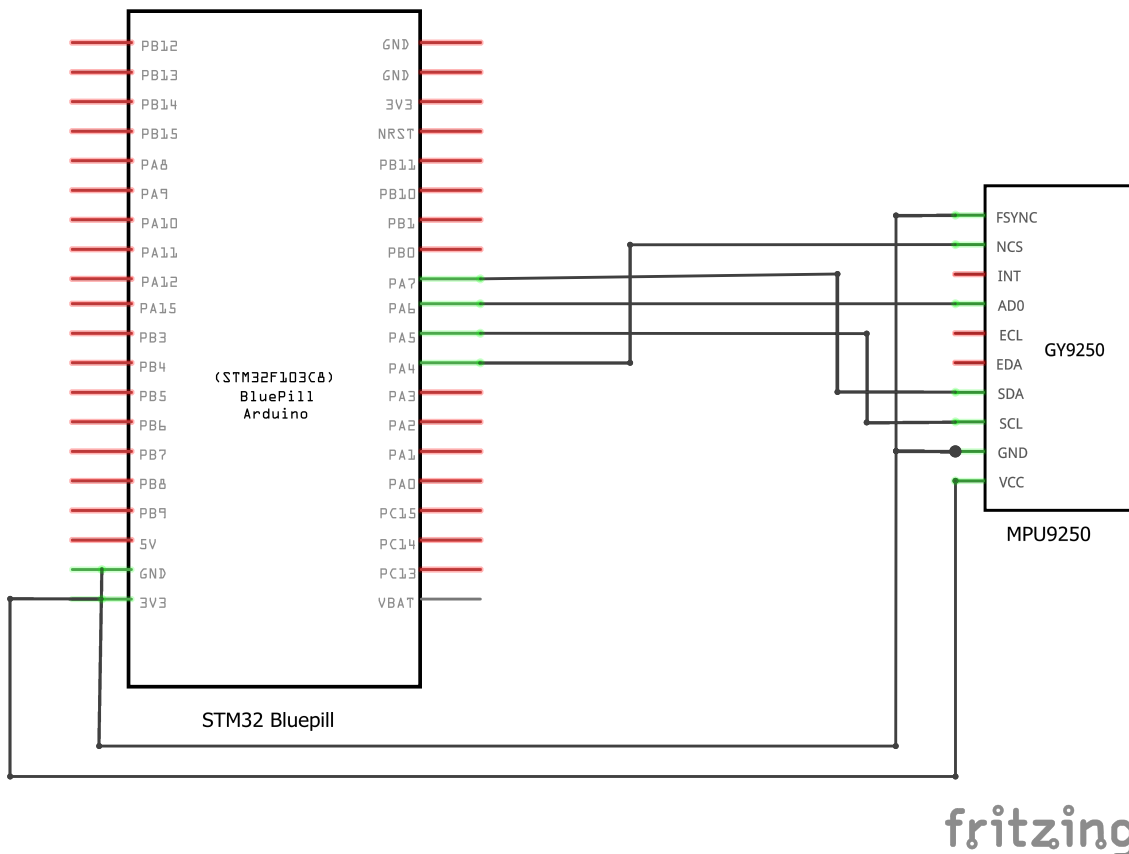
Figure 5.13 Schematic of the MPU9250 and the STM32 Bluepill connections. Source: Own.

Eventually, the pin connections is outlined in the table below 5.6:

Table 5.6 Arduino Nano and MPU9250 connections. Source: Own.

STM32 Bluepill	BNO055 IMU
3.3 V	VCC
GND	GND
A4 (SS)	NCS
A5 (SCK)	SCL
A6 (MISO)	ADO
A7 (MOSI)	SDA
FSYNC	GND

5.2.2   Bluetooth module assembly

Bluetooth communication technology is designed as a short-range connectivity solution for personal, portable and handheld electronic devices such as smartphones, laptops and smartwatches [62]. This technology supports both synchronous and asynchronous data flow transmission over links with speeds of 1 Mb/s. It operates in the Industrial, Scientific and Medical (ISM) radio band spectrum of 2.4 GHz utilizing low transmit power radios and Frequency-Hopping Spread-Spectrum technique (FHSS). The FHSS is a signal modulation technique that involves that a signal generated with a particular bandwidth is deliberately spread within the frequency band thus reducing interference and jamming and most importantly, preventing detection or interception. This communication protocol functioning is illustrated in Figure 5.14,
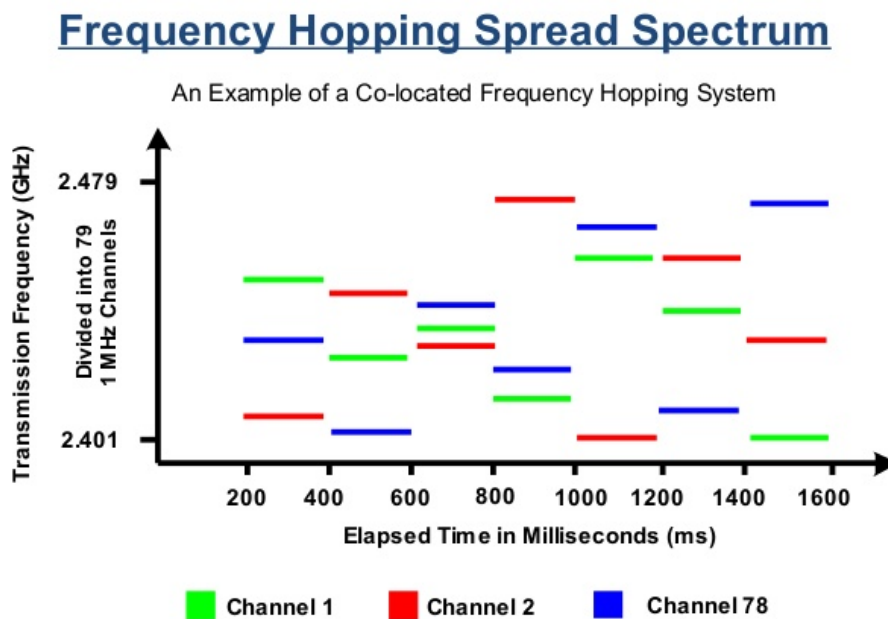
Figure 5.14 Frequency-Hopping Spread-Spectrum diagram. Source: Sharda University [63].

67

The HC06 Bluetooth module uses UART communication protocol, thus, the setup of this module is done by sending AT commands (Attention Commands). AT commands mode is used to modify the Bluetooth module's default settings. To alter the BT device name, device role such as master or slave, the password of the device BT module must be established in AT Command mode, and the default settings can be changed. This configuration code can be found on E.1.
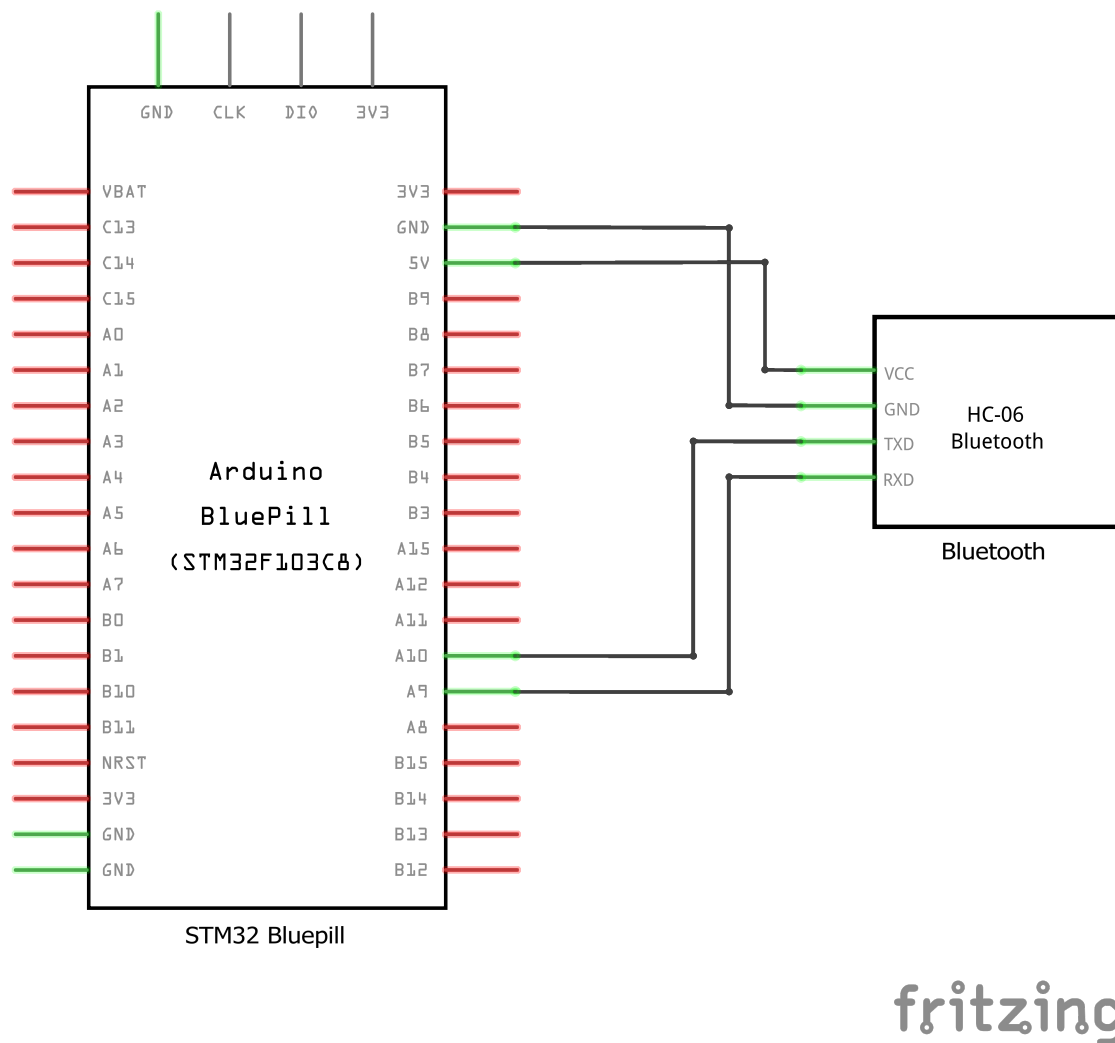
Figure 5.15 Bluetooth setup schematic with STM32 Bluepill. Source: Own.

It is important to mention that the HC06 is connected to a 5 V output pin from the Bluepill. Depending on the manufacturer, they mention using a 3.3 V pin to supply power to the module. Hence, as regards the Bluepill microcontroller is as simple as switching to a 3.3 V output pin. Nevertheless, when using an Arduino Nano microcontroller, one has to implement a voltage divider since there is no 3.3 V output pin. Even so, the one used for the project is capable of working at 5 V.

Figure J.4, schematic 5.15 and the Table 5.7 illustrate how to connect the pins with the Bluepill micro-controller:

Table 5.7 STM32 Bluepill and HC06 Bluetooth module connections. Source: Own.

STM32 Blue Pill	HC06 Bluetooth module
3.3 V	VCC
GND	GND
A9	RXD
A10	TXD

To connect with the STM32, UART1 interface has been used where pins A9 (TXD) and A10 (RXD) is connected to the Bluetooth module's RXD and TXD pins, respectively. Once everything is set up, the next step is to send the data from the Bluepill to any device that supports Bluetooth.

5.2.3 Motor and Motor Driver assembly

To assembly both the motor and motor driver, the diagram and schematic below present the connections to the STM32 Bluepill (see Figures J.5 and 5.16).

The motor driver, as seen previously, has the role of powering the motor since the output pins of the microcontroller do not have enough range of voltage to use the motor at its full operational range potential. The testing control code can be found in G.8. For both IMU configurations (SPI and I^2C) the code is very similar as the only parameters changed are from the IMU readings.

The driver is connected to the Bluepill microcontroller via I^2C port. However, Sparkfun has developed its own pin for this type of connection, the Qwiic cable provides the needed functionalities of I^2C but reduces the space needed for a PCB assembly (check Appendix G for more details).

Diagram J.5 corresponds to 5.16 in which the IMU is connected via SPI protocol.
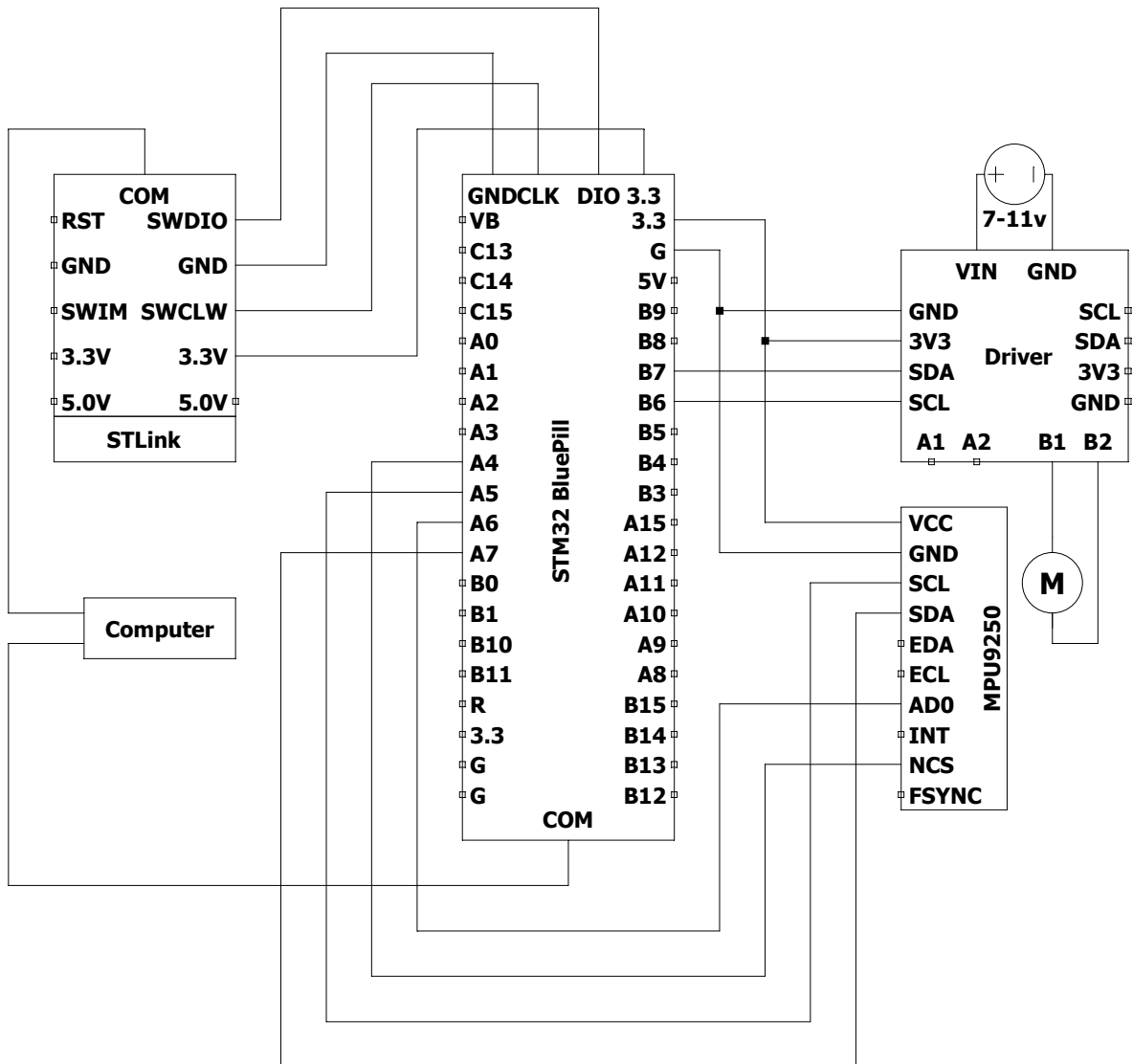
Figure 5.16 Schematic of the motor driver and motor and the STM32
Bluepill connections with SPI. Source: Own.

The connections are (check Appendix G for Qwiic cable pinout):

Table 5.8 STM32 Bluepill and Qwiic motor driver. Source: Own.

STM32 Blue Pill	Qwiic motor driver
3.3 V	VCC
GND	GND
3.3	3.3 V
B6	SDA
B7	SCL

Other connections left of this driver is the motor, which can be connected to either A1 and A2 pins or B1 and B2 pins of the driver. This means that more than one motor can be used at the same time as well as two possible configurations can be used depending on the distribution of the PCB. In order to change the configurations is as simple as setting either a 0 or a 1 in the setDrive() function. The first parameter of this function represents the peripheral, the second one is also a Boolean which represents the direction of the rotation and the last parameter is an integer that ranges from $[0, 255]$ which is the amount of power given to the motor. Finally, an external battery connected to VIN and GND will provide sufficient energy to move the respective motor. In contrast to the Arduino Nano microcontroller, where a DC-DC converter is necessary, for the Bluepill there is no DC-DC converter since the output voltage pin 3.3 V is the same as the necessary voltage of the driver.

To use the driver, a set of libraries must be installed which are provided by Sparkfun itself. These libraries are SCMD.h and SCMD_config.h. Next, the driver object is created. Following this step, the I^{2}C address is set for the communication to finally enable the driver.

Analogously, if the user wants to connect the IMU using I^{2}C, the following schematic shall be considered 5.17:
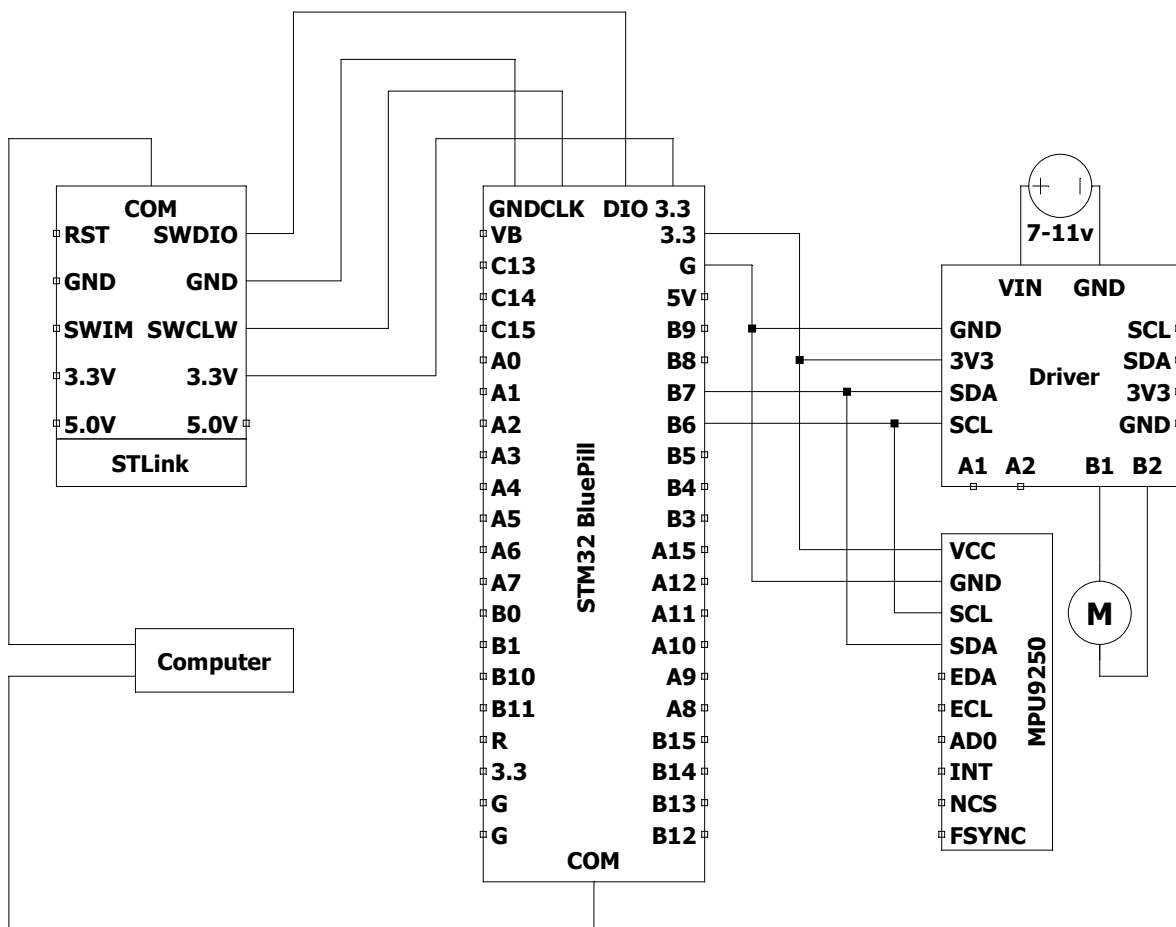
Figure 5.17 Schematic of the motor driver and motor and the STM32
Bluepill connections with I^{2}C. Source: Own.

71

5.2.4   Full assembly

The following diagram shows the final assembly of the components into a breadboard. This information will be used to design the custom PCB that holds all the following components:

- 1× STM32 Bluepill controller

- 1× HC06 Bluetooth module

- 1× Mabuchi motor RF-300EA-1D390

- 1× Sparkfun Qwiic motor driver

- 1× External battery
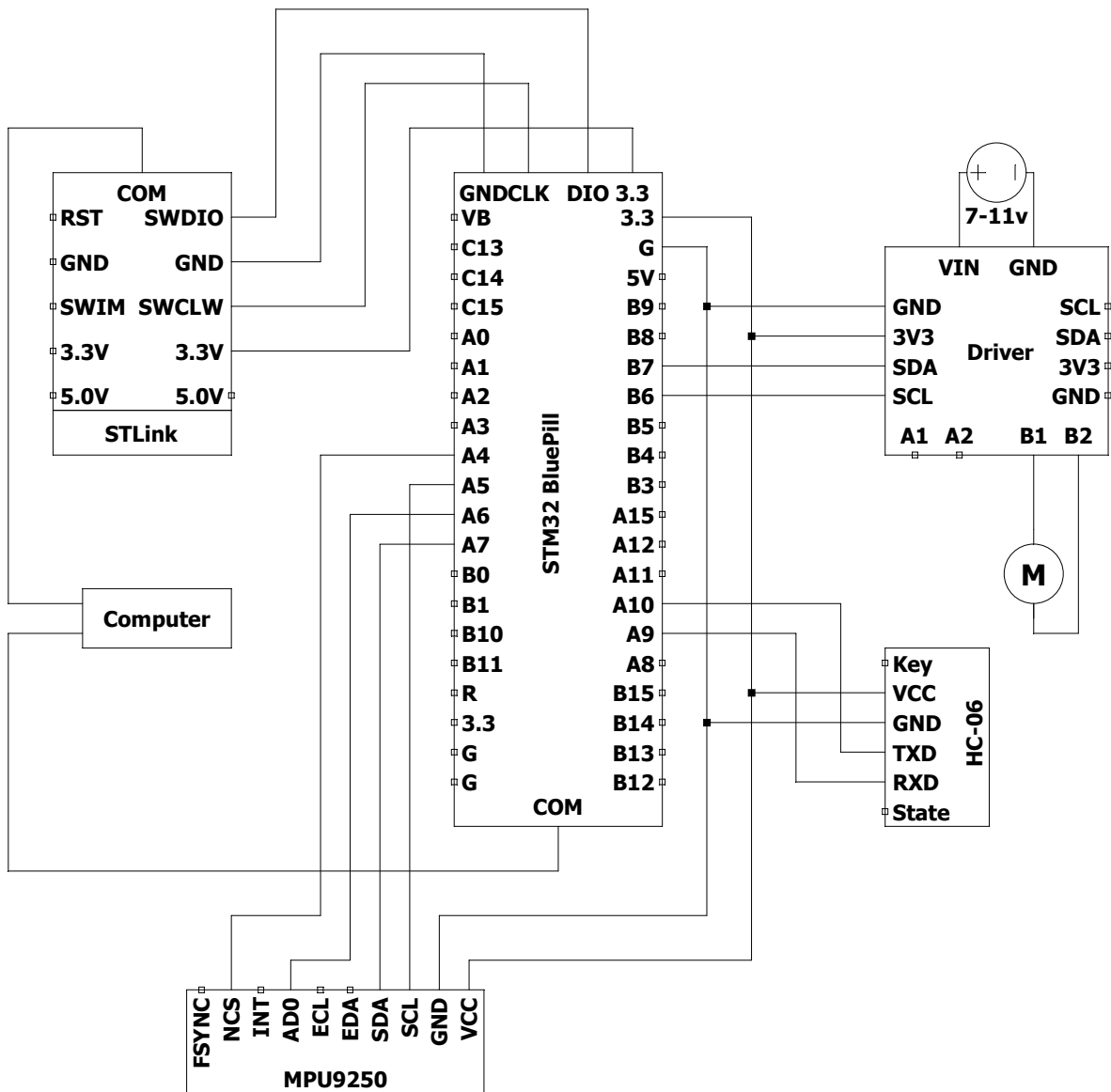
Figure 5.18 Final assembly schematic. Source: Own.
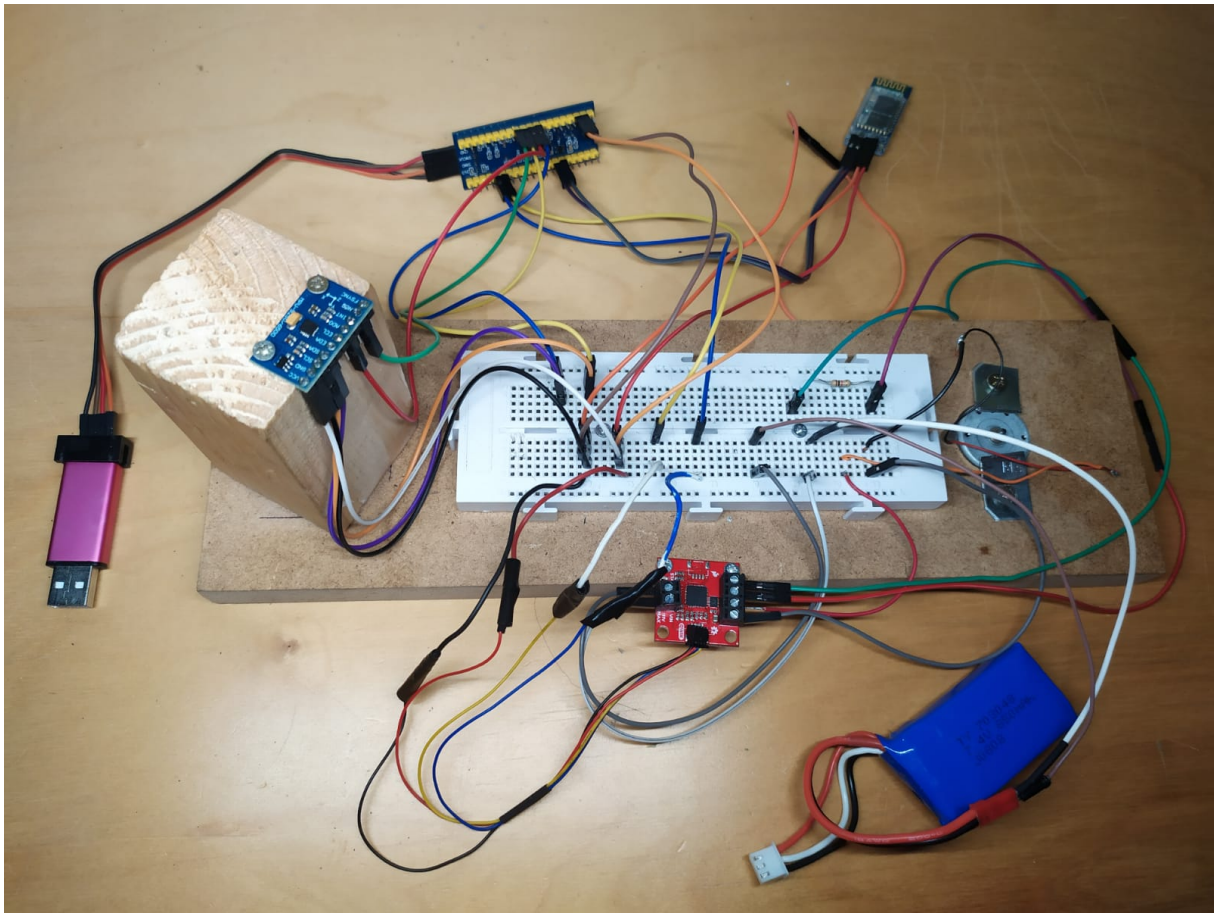
Figure 5.19 Final assembly physical assembly. Source: Own.

Next chapter 6 shows the design of the PCB. This PCB will not only take into account the components for the reaction wheel but also the pair of magnetorquers designed by the magnetorquer team.

Chapter 6

PCB Design

In the following chapter, it is shown the design of the Printed Circuit Board (PCB) and the circuits that will allow the communication.

A printed circuit board, or PCB, is a type of circuit board that is used to physically support and electrically link electronic components by connecting conductive paths, tracks, or signal traces from copper sheets bonded onto a non-conductive substrate. The PCB is specifically designed by PLATHON's project electronics requirements. For a more in-depth overview of the schematics see Appendix I.

6.1   Schematic

Prior to designing the schematic, several CubeSat prototypes were already available within PLATHON's project (see Figures 8.3). The particularity of the 8.3 Cubesat prototype resides in its ability to access the PCB with ease, namely, this CubeSat can be classified as a modular 1U Cubesat. For instance, whenever the user is willing to change the PCB or make some rearrangements the process is seamless.

This Cubesat complies with the structural restrictions listed in A. Its dimensions is 9.1×9.3 cm and the components chosen for the project has already been thought to fit inside the PCB.
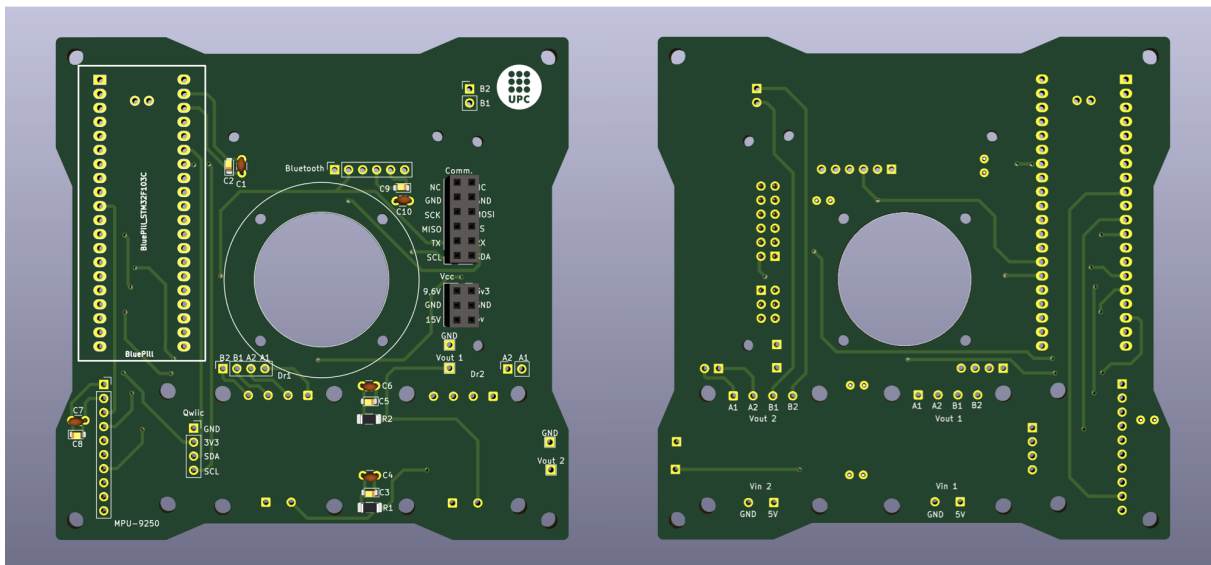
Figure 6.1 PCB layout top view (left) and bottom view (right). Source: Own.

Notice how the PCB is not fully squared and a slight material is missing in its sides. The reason behind this design was to enable the power cables to connect from the solar panel cells to the main board and supply power to the system.

The PCB is designed following the corresponding technical specifications. Each pin is connected via copper paths with a separation width of 0.3 mm and a path width of 0.6 mm. This model was chosen because it is the most feasible and practical ways to avoid potential errors, i.e. when paths are too close there might be some interactions between pins that can cause short circuits.

Additionally, as regards the PCB, it consists of 4 layers in which all connections are settled on side of the board. This design facilitates the welding process of the different components such as the motor and the IMU.

(a) PCB Gerber file. Source: Own.

(b) PCB Excellon file. Source: Own.
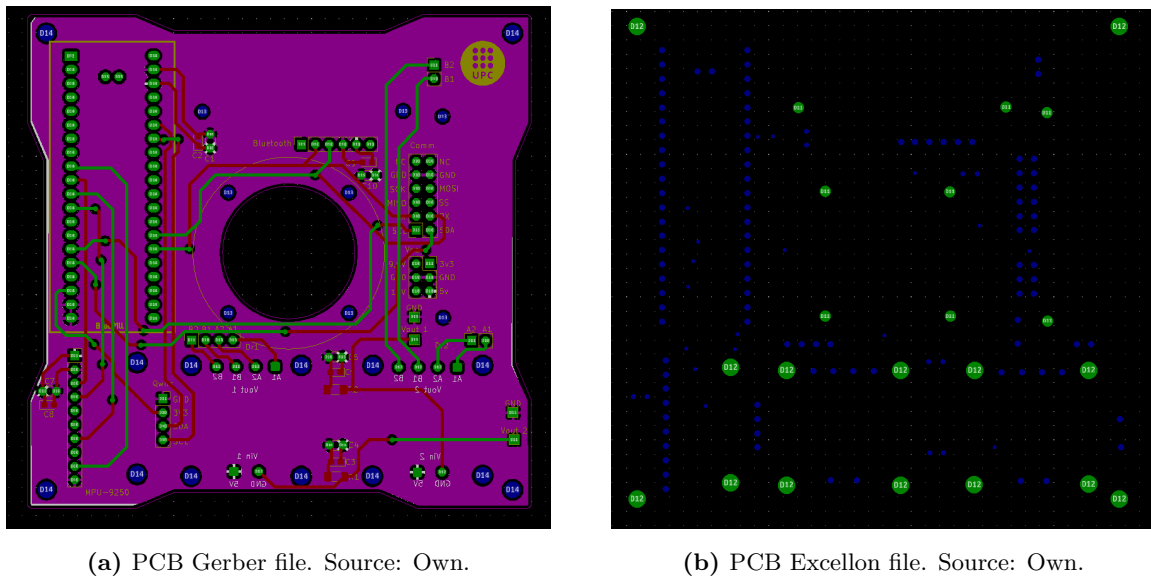
Figure 6.2 PCB Gerber and Excellon files. Source: Own.

Gerber files are ASCII vector files that provide information about each physical board layer of the PCB design. This include, for instance, copper paths, solder mask and silkscreen graphics and they are all represented by a drawing code and specified by a sequence of vector coordinates. *Excellon* drill format is a file format used for drilling and routing equipment and it is an industry standard. Both files are necessary for manufacturing the PCB. These files were sent to *JLC PCB*, a company specialized in manufacturing all kind of PCBs for different needs and applications.

(a) Physical PCB (front). Source: Own.

(b) Physical PCB (back). Source: Own.
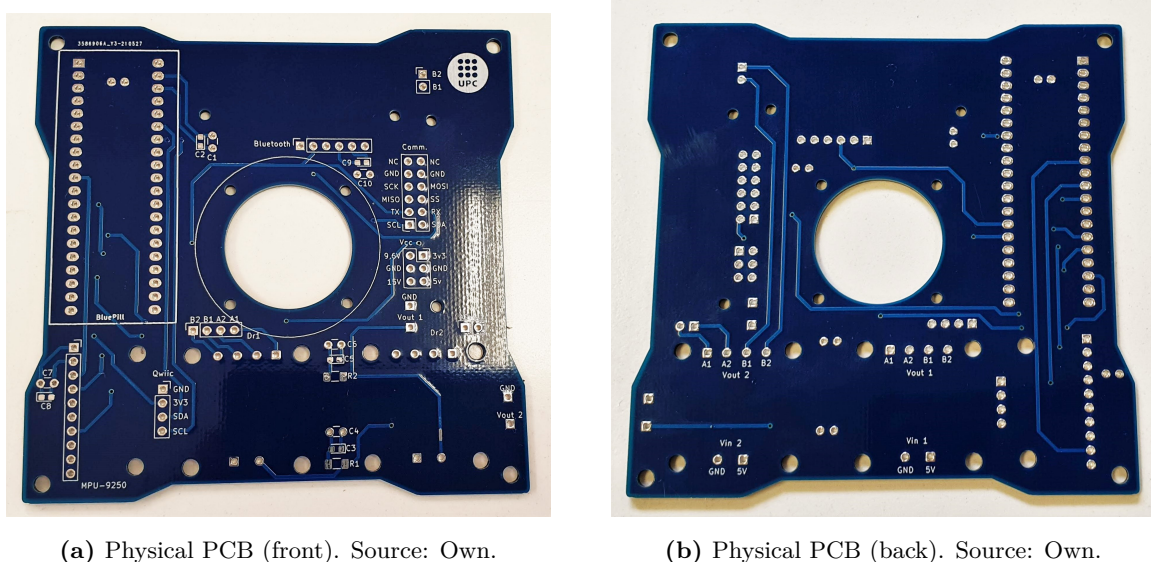
Figure 6.3 Physical PCB. Source: Own.

The PCB is made out of an FR-4 sandwich layer between two thin layers of copper lamination. FR-4 is by far the most used material when it comes to manufacturing PCBs. The letter "FR' denotes that the substance is Flame Retardant, and the number "4" indicates that it is woven glass-reinforced epoxy resin. Additional prepreg layers are sandwiched between the central core and the top and bottom copper layers

in this board. FR-4 boards offer good strength and water resistance characteristics that allow them to be used as an insulator in many electrical applications. It insulates neighbour planes and provides bending strength to the structure.

Before engaging in welding the components, several tests were performed to ensure the paths are correctly connected and there are no short circuits.

Two design errors were found during this phase. The first one is that all the system is connected to a 5 V input from an external battery and despite there is a 9.6 V input pin (see Figure 6.1) that pin is connected as well in the 5 V paths and not to the VIN input for the drivers. However, after performing some tests with 5 V. The results shows it is enough to rotate the entire satellite even when using 20% of the supplied voltage (i.e. 20% of 5 V (see 8)).

The second design error resides in the VIN and GND pins of the drivers. These pins are switched. However, a practical solution proposed was to connect the pins with a short cable that connects the pins instead of placing the drivers on the other side of the board as there might be some friction with the reaction wheel.

6.2   Final Design

Eventually, KiCad software enables to render the pieces in a 3D view and the final assembly of the components shall look like as Figure 6.4:
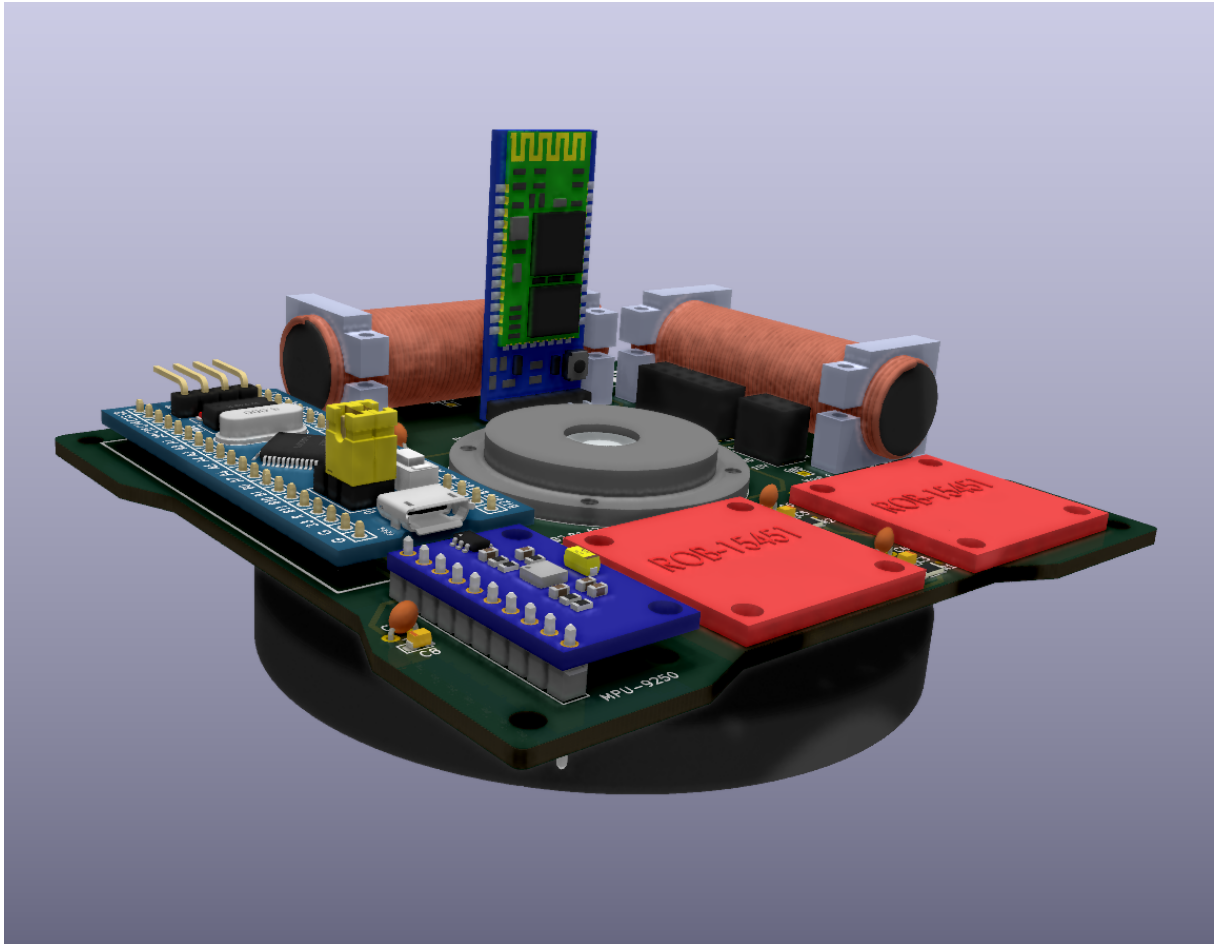
Figure 6.4 Final PCB assembly render. Source: Own.

In Figure 6.4 there is the Reaction wheel on the backside of the board and it carries two magnetorquers at the side of the board. The vertical module in front of the reaction wheel is the Bluetooth module. Also, the two red pieces in the lower right side of the figure are the two set of drivers that controls both the magnetorquers and the reaction wheels.

Finally, Figures 6.5 and 6.6 shows the top and bottom sides of the assembled PCB. Figures 6.7 6.8 shows the physical assembly of the board with all the components.
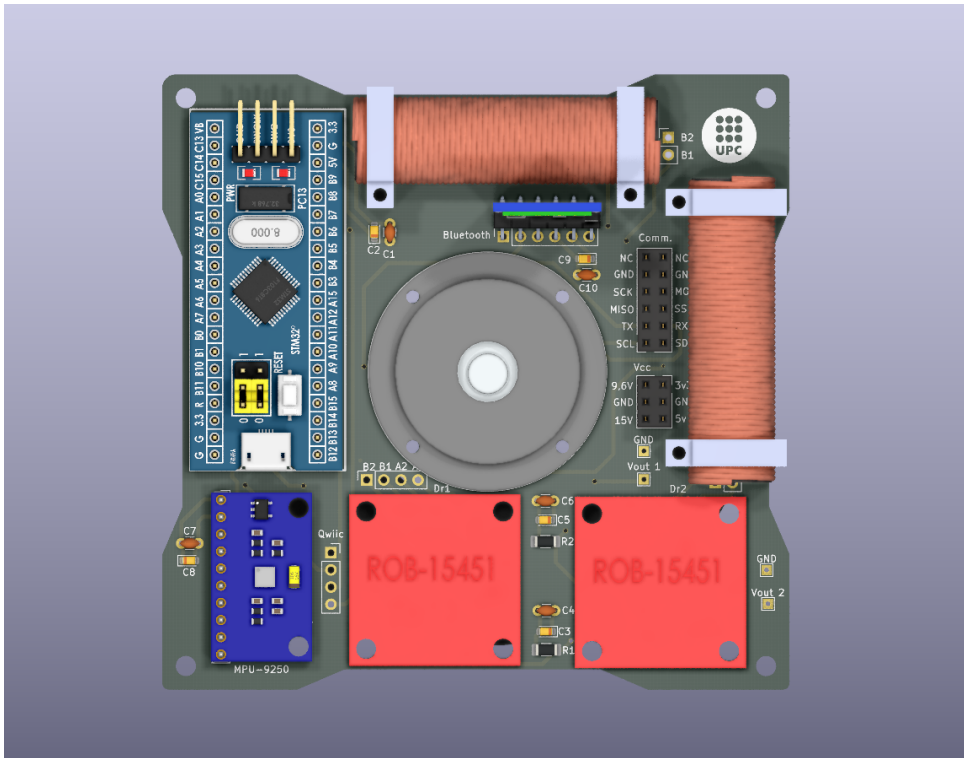
Figure 6.5 Final PCB assembly render (top view). Source: Own.
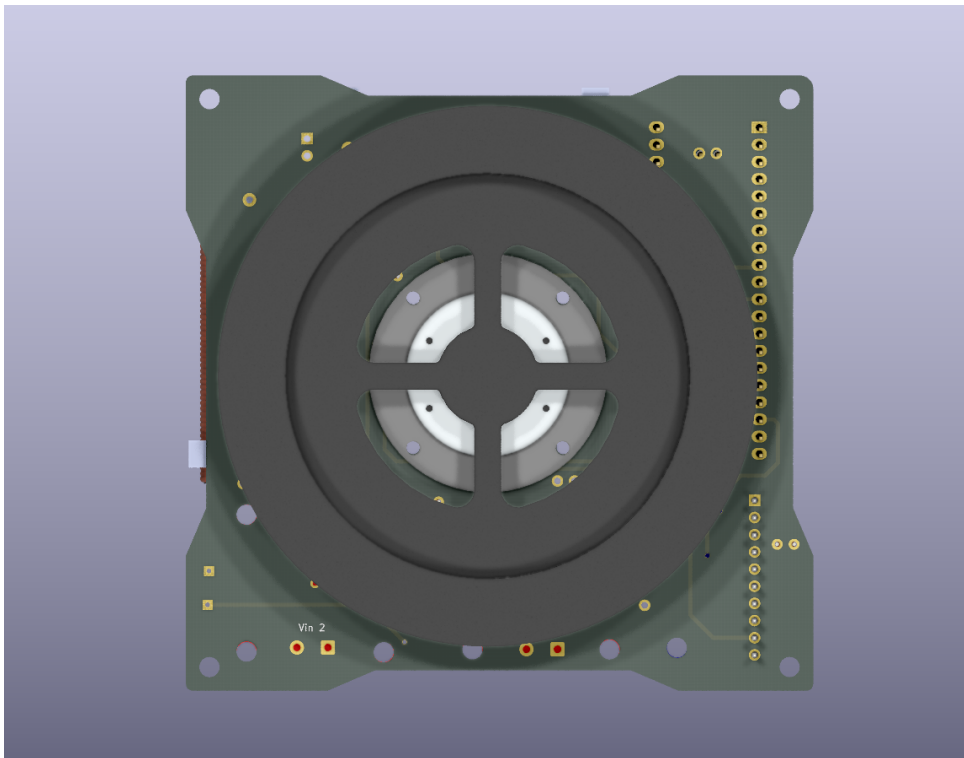
Figure 6.6 Final PCB assembly render (bottom view). Source: Own.
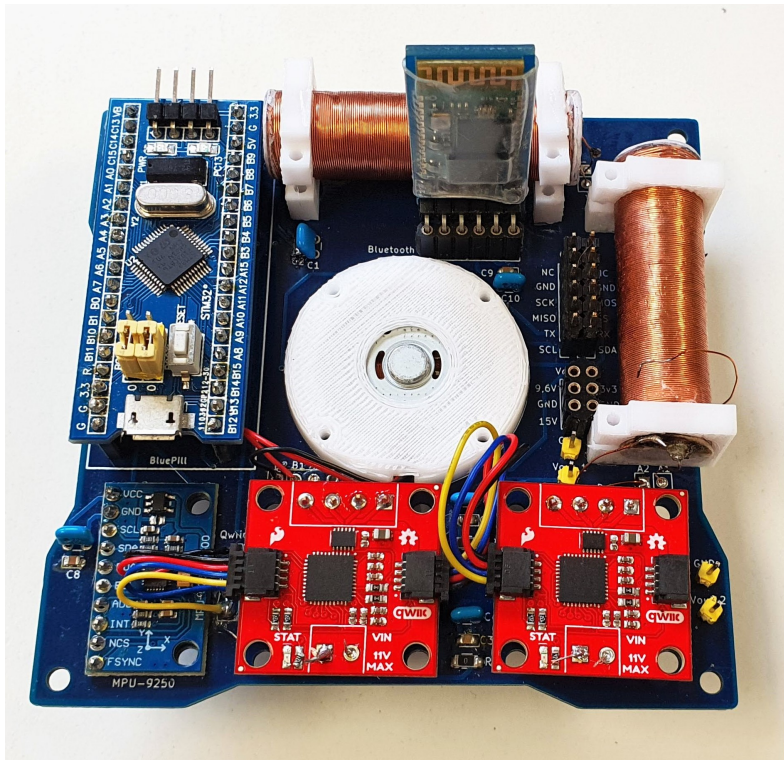
Figure 6.7 PCB layout top view (left) and bottom view (right). Source: Own.
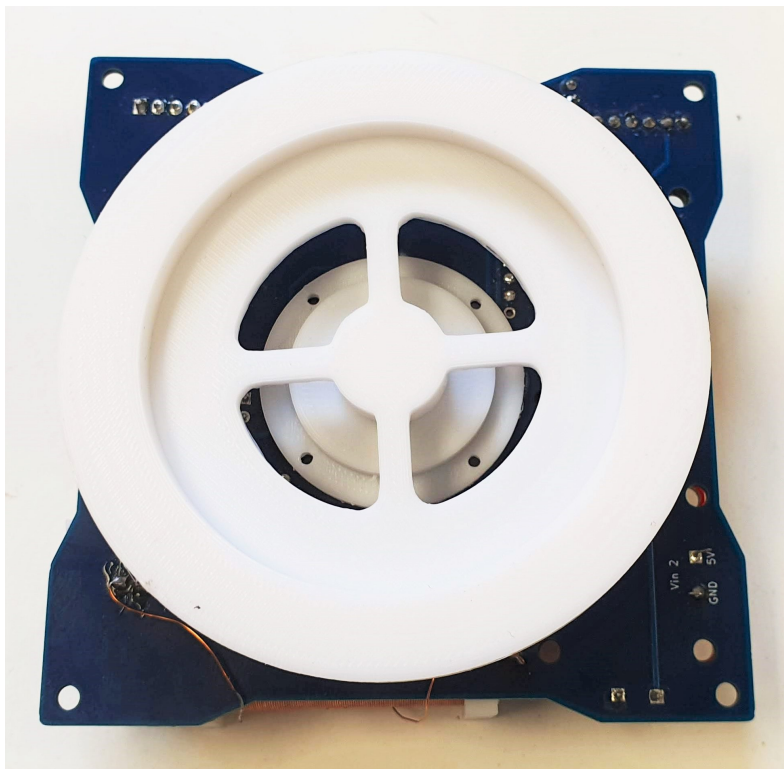
Figure 6.8 PCB layout top view (left) and bottom view (right). Source: Own.
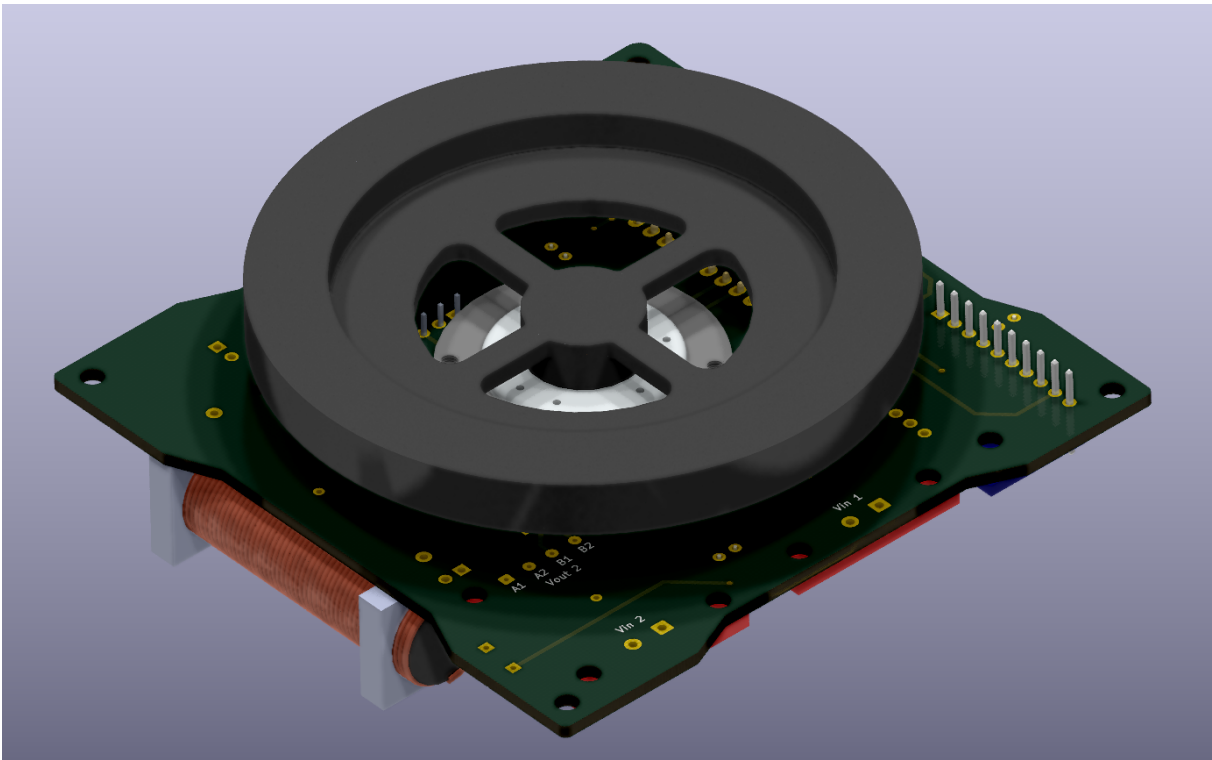
Figure 6.9 Reaction wheel mounted on PCB. Source: Own.

Chapter 7

ADCS software

The following chapter intends to explain the control algorithm developed for the ADCS subsystem of the CubeSat. This control is developed for a 1DOF Cubesat which means it only controls the heading of the CubeSat. However, the code is implemented so it can be easily extrapolated to a 3DOF control in the future. This part of the project is done in collaboration with [64] and [65] which forms the ADCS reaction wheels team. To download the entire control algorithm visit Github.

7.1 Control Algorithm

Firstly, as seen in Section 3.2.1, the reference frame used for the CubeSat is the Orbit reference frame. However, more local references shall be noted. Figure 7.1 illustrates the set of reference systems used for the development of the control software.
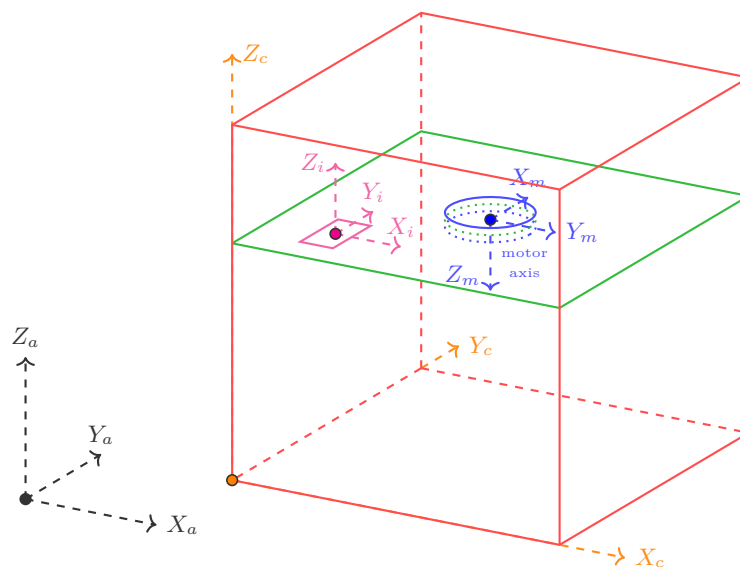
Figure 7.1 CubeSat reference frames. Source: Own.

Four axis systems are shown in Figure 7.1,

- X_a, Y_a, Z_a are the **absolute local system of reference** which corresponds to orbit reference frame but with z-axis pointing outwards Earth center.

- X_a, Y_a, Z_a are the **CubeSat's system of reference** which are aligned with the IMU's reference frames due to symmetry.

- X_i, Y_i, Z_i are the **IMU's system of reference** which are set by the own Inertial Measurement Unit.

- X_m, Y_m, Z_m are the **motor's reference frame**. This reference frame is coupled with the motor's orientation.

The reference systems are extremely important to bear in mind since depending on the orientation or the setup of the board will change the code and some signs will be needed to be changed. Again, the developed code is only valid for a 1U CubeSat in the same orientation as Figure 8.5, 8.6.

Once the assembly is done, the next step is to calibrate the IMU. This calibration process is located within the main code and calibrations are needed for both the accelerometers and gyroscopes (see Code in O).

7.1.1 ADCS Structure

The control software of the ADCS is shown in flowchart 7.2. The system is organized in 4 different modes of operation. Magnetorquer's team is in charge of detumbling mode, nadir pointing whereas Reaction wheels team is in charge of positioning modes since the reaction wheels torque is stronger than magnetorquers.

Take a look at the flowchart in Figure 7.2, each mode is coloured in orange (except emergency mode which is in green).

7.1.2 Control algorithm

Then after all configuration is done, the general control algorithm flowchart is shown in Figure 7.2:
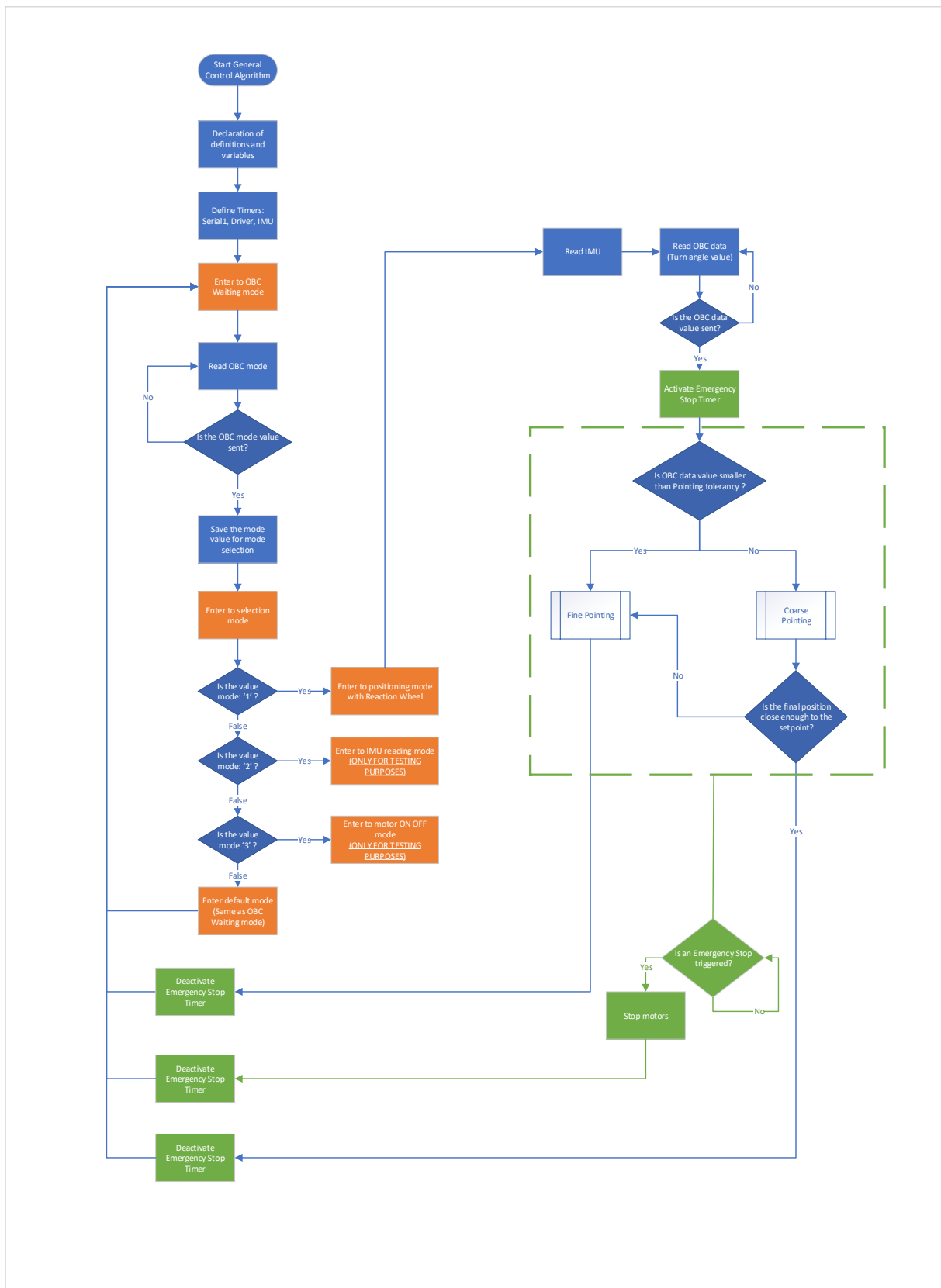
Figure 7.2 ADCS control flowchart. Source: Own.

The control is briefly explained here-under (see Appendix O for the code and O for the Table with all

variables and O for the Table with all functions and their explanation).

First, the program sets up different definitions and parameters. All the variables used in the code are global variables as the program does not make use of functions, instead, Timers were employed for every mode. The reason why the code uses only Timers is that microcontrollers such as Arduino Nano and Bluepill cannot do multitasking. Thus, using Timers can emulate multitasking and perform several actions at the same time (see Section 7.1.3 for a detailed explanation on Timers).

After Timers functions are defined, the microcontroller enters to `mode_OBC_Input_Wait`, which is the default mode for receiving a mode from the On-Board Computer (OBC). This mode constantly checks the `Serial1` port from UART to see if the user has sent any command and then execute the corresponding mode.

Once the microcontroller receives the data from the OBC, depending on the input from the user, it enters to either Positioning mode `mode_Positioning_RW()`, IMU reading mode `mode_IMU_reading` or `mode_motor_on_off()`.

- `mode_OBC_Input_Wait()`: Default mode for OBC reading.
- `mode_Positioning_RW()`: This mode allows the user to turn the CubeSat into the desired position using reaction wheels. The inputs are the direction of turn (whether clockwise or counter-clockwise) and the desired angle to turn. Then inside this mode. Depending on the tolerance, the microcontroller will enter to either Fine or Coarse pointing mode. Once set-point is reached and the CubeSat remains still. The process goes back to Waiting mode.
- `mode_IMU_reading()`: This mode displays the IMU's values to let the user check the orientation values (roll, pitch and yaw).
- `mode_motor_on_off()`: THis mode triggers an impulse to the motor and after a time it turns off.

7.1.3 Timers

First, the program sets up different definitions and parameters. All the variables used in the code are global variables as the program does not make use of functions. Instead, Timers were employed for every mode. The reason why Timers are used is that they can provide virtual "multitasking" to the microcontroller.

Neither the Arduino Nano nor the Bluepill microcontrollers have the ability to real multitasking as they only have 1 core and no operative system. When we talk about timers, in reality, we are taking advantage of the fact that they offer the possibility to temporize sub-tasks without blocking the main task. Thus, microprocessors usually have a set of timers which is a mechanism that permits the execution of an interruption, known as Interrupt Service Routine (ISR). When the program executes this callback function, it goes off the normal flux of the program and executes the associated ISR ignoring all other functions.

There are hardware interruptions that are triggered by a physical change in an event on a physical pin and software interruptions that are in sync with the microcontrollers internal clock.

Interruption functions (ISR) must take as short amount of time as possible as the main program is stopped. Nevertheless, in the code implemented, it is only featured to use interruptions and the main `void()` function is left blank. Eventually, building a program like this takes full advantage of the microprocessor and emulates a "multitasking".

To configure the timers, two concepts must be defined. Dividing the STM32 Bluepill's internal clock's frequency to the Prescale Factor (PF) determines the frequency of the Timer.

$$\mathrm{d}t = \frac{1}{\frac{\text{STM32}_{\text{C}}\text{LOCK}}{\text{PF}}} \cdot \text{Ov} \tag{7.1.1}$$

Multiplying the frequency of the Timer by the Overflow (Ov) one gets the period. In the project, the internal clock is set to 72 MHz, so a PF of 7200 and an Overflow of Ov of 1000 are used to obtain a period of 100 ms. See Figure 7.3 example:
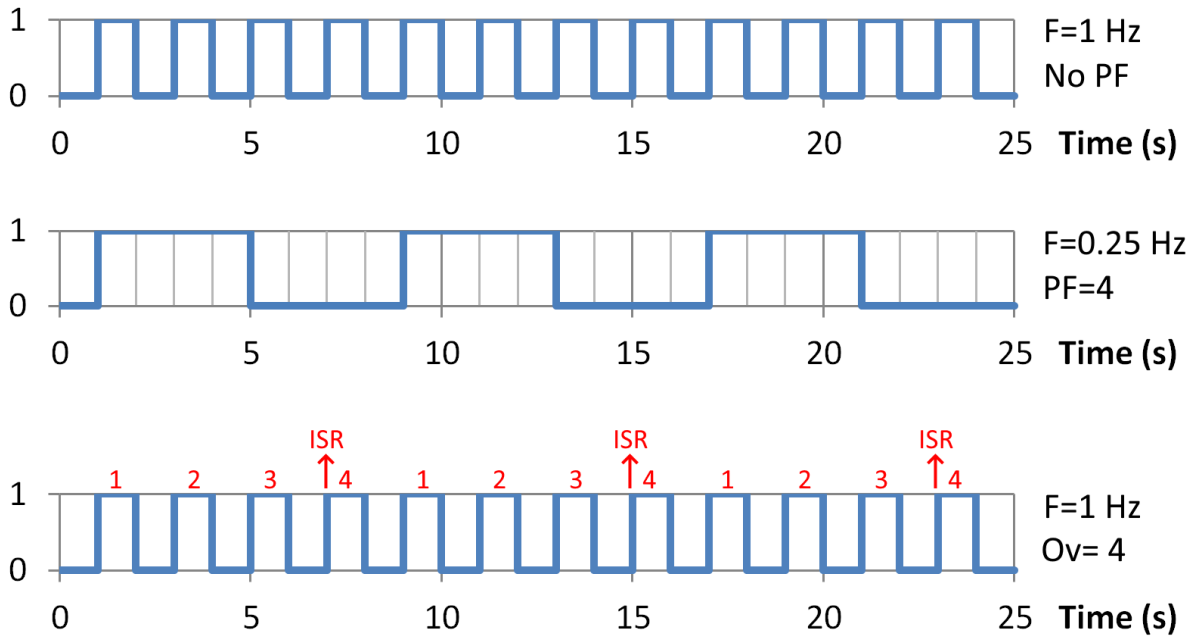
Figure 7.3 Timer and Prescale Factor diagram. Source: Adrià Pérez [64].

The first wave represents the original STM32 clock without any PF nor Ov. The second wave introduces a prescale factor and the third wave introduces the Ov to see the effect on the Timer and the ISR calls.

On important notice is that a Timer can have multiple channels. Timer1 was used for the program and channels CH3 and CH4 were used for OBC readings and IMU readings, respectively.

7.1.4 Coarse Pointing

Coarse pointing mode is the manoeuvre that allows the CubeSat to turn into the desired angle in a fast way. The principle behind it is to give a big impulse to rotate the CubeSat very quickly which produces an acceleration phase. Then to stop the motor when it is approaching the desired angle, the same impulse is injected in the opposite direction of rotation provoking a deceleration.
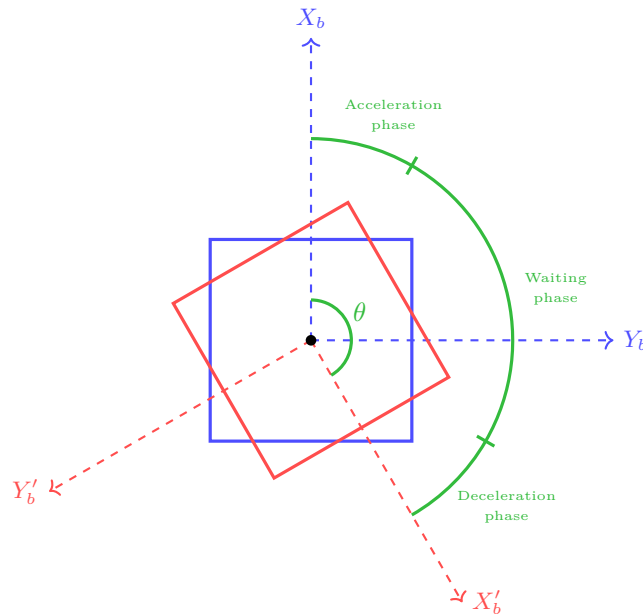
Figure 7.4 Coarse pointing mode manoeuvre. Source: Own.

Hence, the coarse pointing mode is divided into three phases:

1. **Acceleration phase**: An impulse is generated to arrive at the angle set-point.

2. **Waiting phase**: In this phase, the acceleration is zero and the motor rotates at a constant velocity. If the current yaw angle is higher than half of the desired angle it goes to the deceleration phase.

3. **Deceleration phase**: This phase is analogous to the acceleration phase but with an acceleration in the opposite direction.

There are many possible cases in the acceleration phase whether the turn is clockwise (CW) or counter-clockwise (CCW). Let's illustrate an example in Figure 7.5. The $\Delta\psi$ is the final yaw angle minus the initial yaw angle $\psi_{\text{final}} - \psi_{\text{initial}}$.

- If the CubeSat goes from 30° to 330° in the CubeSat CW direction, the difference in angle is $330 - 30 = -300°$.

- If the CubeSat goes from 30° to 330° in the CubeSat CCW direction, the difference in angle is $-(330 - 30) = -300°$, or 60°.

- If the CubeSat goes from 330° to 30° in the CubeSat CW direction, the difference in angle is $30 - 330 = -300°$, or 60°.

- If the CubeSat goes from 330° to 30° in the CubeSat CCW direction, the difference in angle is $-(30 - 330) = 300°$.
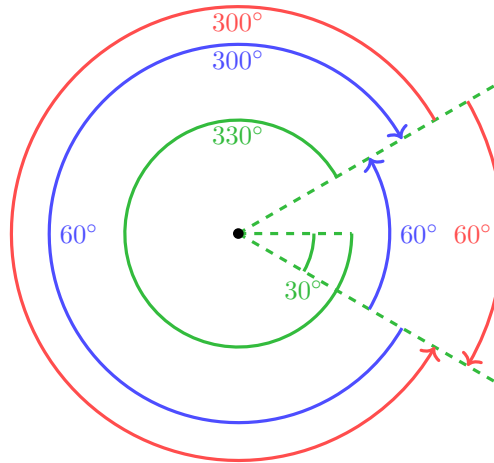
Figure 7.5 Turn diagram. Source: Own.

If the CubeSat orientation passes through the zero angle, the Yaw angle would decrease its value drastically from 360 to 0 (see Figure 7.6).

(a) Yaw angle diagram when the CubeSat passes through 0° angle (Sketch). Source: Own.

(b) Yaw angle diagram (Sketch). Source: Own.
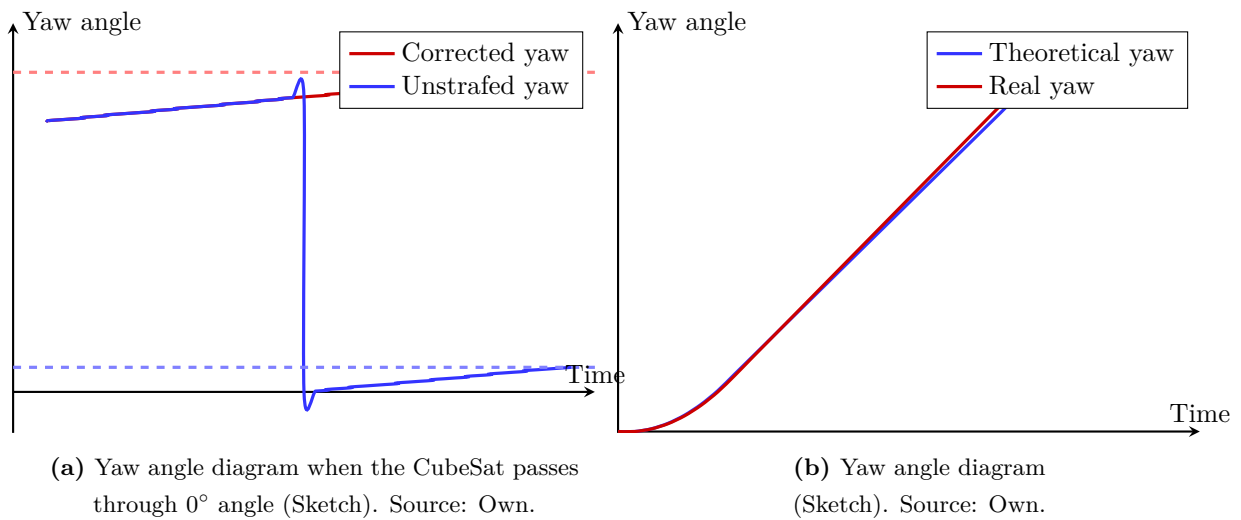
Figure 7.6 Yaw angle diagrams (Sketch). Source: Own.

Figure 7.7 shows the angular speed of the system associated with the three phases clearly visible which are acceleration, waiting and deceleration phase.
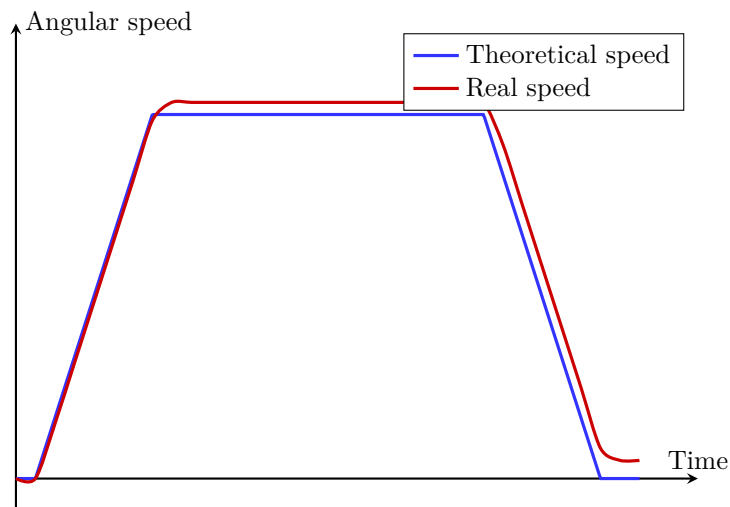
Figure 7.7 Angular speed diagram (Sketch). Source: Own.

As regards the angular acceleration, Figure 7.8 shows the diagram impulse and theoretical and real acceleration values. In Section 8 we'll see if the results coincide with the theoretical diagram.
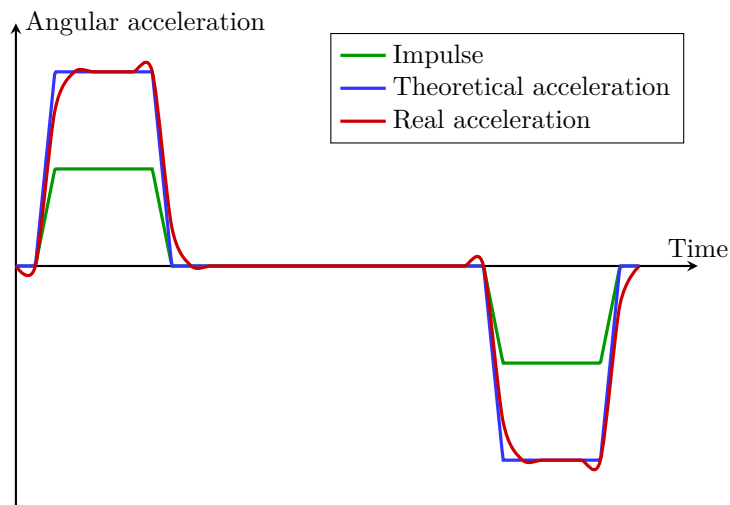
Figure 7.8 Angular acceleration diagram (Sketch). Source: Own.

At the end of the Coarse manoeuvre, if the CubeSat is not correctly oriented the next manoeuvre that it will enter is the Fine pointing mode. If the CubeSat happens to get to the desired angle within a tolerance set in the Huge impulses will rarely meet the results and consequently, thus, fine control will try to minimize the error with a PD control.
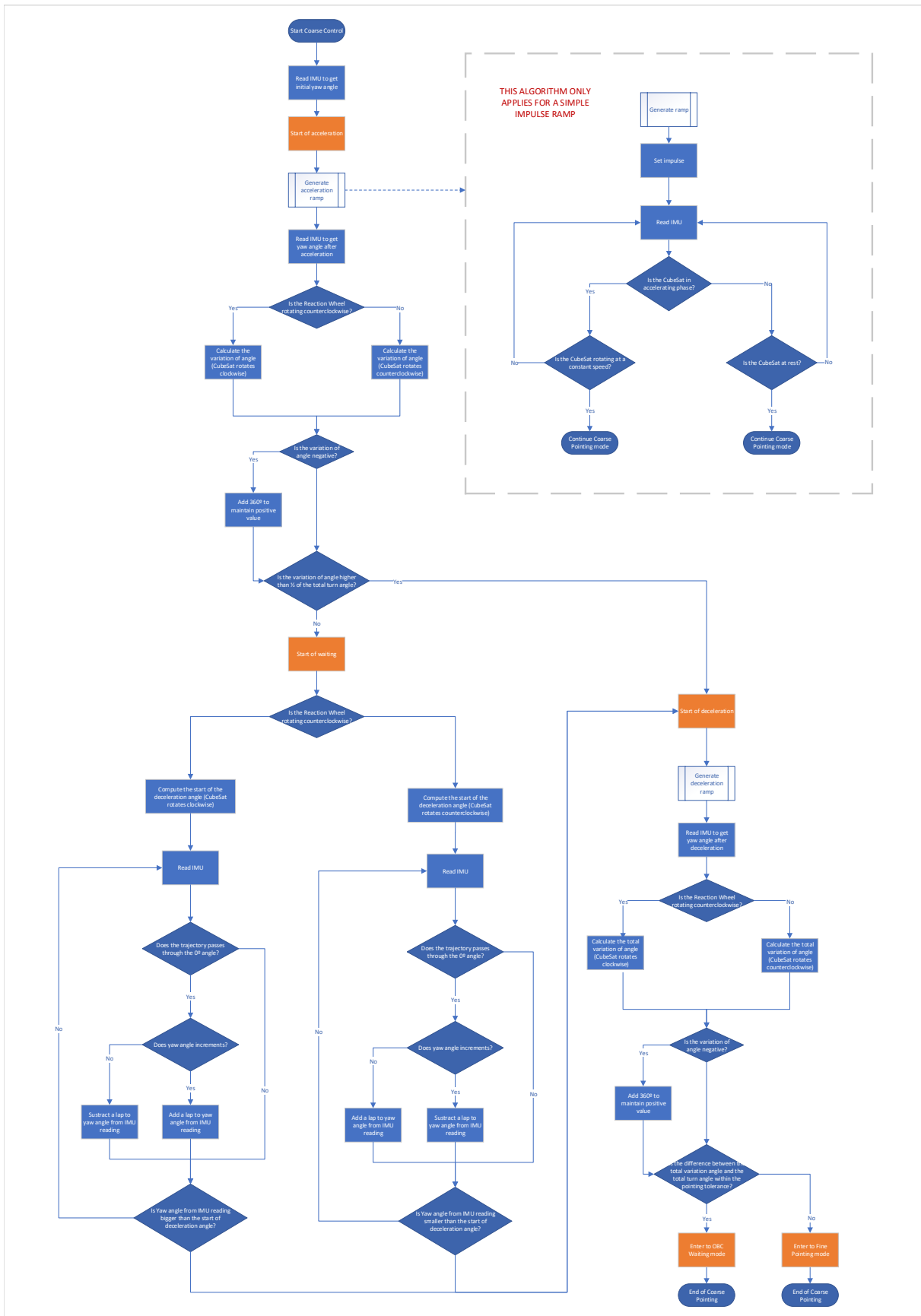
Figure 7.9 Coarse pointing mode flowchart. Source: Own.

7.1.5 Fine Pointing

Fine pointing mode activates when the desired angle and the actual CubeSats angle are $< 5°$. Fine pointing mode's goal is to increase the accuracy of the orientation that an impulse might not achieve.
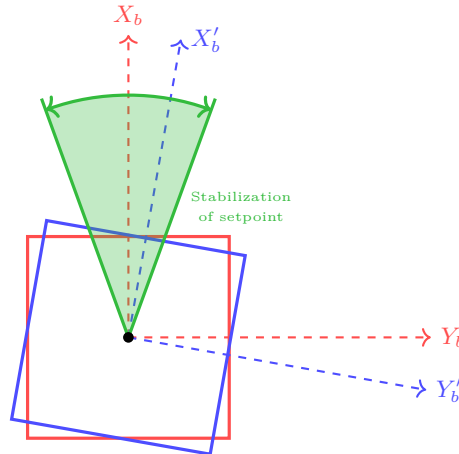
Figure 7.10 Rotation cubesat fine full. Source: Own.

7.1.6 PID controller

A PID algorithm consists of the three components before, a proportional constant, integral and a derivative constant which are tuned to get the optimal response.

The basic principle of this algorithm is to calculate the proportional, integral and derivative responses to sum all of them to the output response [66]. A *close loop system* control is a system that is constantly reading sensors to provide constant feedback and calculates the desired output continuously at a fixed loop rate. The *process variable* is the parameter that the system must control, in this case, the process variable is the yaw angle ψ. And the user inputs the desired set-point.

The formula of a PID controller is defined as follows:

$$u(t) = \underbrace{K_p e(t)}_{\text{Proportional}} + \underbrace{K_i \int e(t)\mathrm{d}t}_{\text{Integral}} + \underbrace{K_d \frac{\mathrm{d}e}{\mathrm{d}t}}_{\text{Derivative}} \tag{7.1.2}$$

where

$u(t)$: PID control variable.

$e(t)$: Error value.

K_p: Proportional gain.

K_i: Integral gain.

$\mathrm{d}e$: Change in error value.

$\mathrm{d}t$: Change in time.

Proportional contribution depends only on the difference between the actual measurement and the setpoint. The proportional gain is multiplied by the error term so, for instance, if the error has a value of 10 and the proportional constant has a value of 5 it means the system will produce a response of 50. The higher the response the faster the systems responds. However, this value cannot be too high since the system can enter an oscillatory phase with no control.

Integral contribution fixes the steady-state error left in the system since it sums the error over time. Steady-state error can be defined as the final difference between the set-point and the actual measurement. In the case of the CubeSat, the integral contribution is not applied since this force usually tends to destabilize the spacecraft instead of correcting it.

Derivative contribution considers the rate of approximation of the process variable and the set-point. If the process variable is increasing rapidly, the derivative component will decrease the rate of change of the process variable. A small step size dt provides more accurate and fast responses. For bigger step sizes, the process variable will certainly go faster to the set-point and hence increase the overall speed of the system. An important issue to note is that for very slow control rates or a noisy signal, the derivative response will make the control unstable.

The process of choosing the best constants to match the desired performance is called *controller tuning*. Ziegler and Nichols proposed criteria for tuning PID controllers as one of the most used ways to tune the variables. The basic idea relies on trial and error technique. First, both integral K_i and derivative K_d are set to 0. Subsequently, the proportional gain is tuned so it oscillates a bit to a point where the response overshoots but not for too much. Then, the critical constant $K_{p,cr}$ and period $T_{p,cr}$ are noted to calculate the other two controller's gains using Table 7.1.

Table 7.1 Ziegler–Nichols Tuning Rule Based on Critical Gain $K_{p,cr}$ and Critical Period. Source: Ogata [66].

Control	P	Ti	Td
P	$0.5K_{p,cr}$	-	-
PI	$0.45K_{p,cr}$	$T_{p,cr}/1.2$	-
PID	$0.60K_{p,cr}$	$0.5T_{p,cr}$	$T_{p,cr}/8$

The response of the controller can be shown in Figure 7.11:
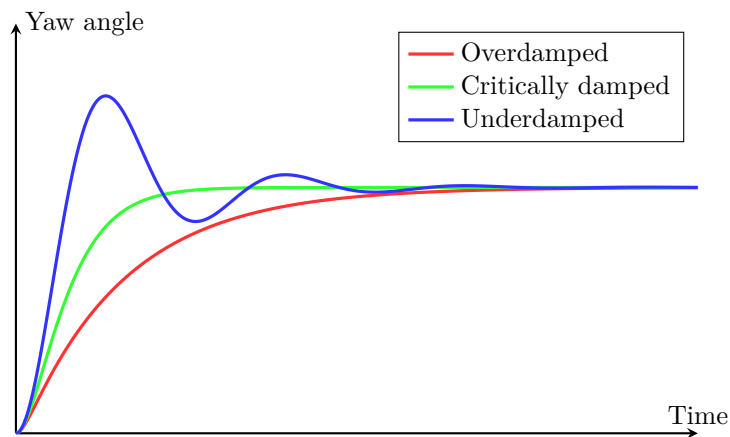
Figure 7.11 System's possible response outcomes [yaw angle] (Sketch).
Source: Own.

The CubeSat enters the Fine mode when the angle difference between the set-point and the process variable is $< 5°$. The final position of the coarse phase shall be set to the initial fine position. Two considerations are made for the final position:

1. The angles of the local reference frame are limited from $[0°, 360°]$. This is done to avoid huge values and has a visual representation of the number of laps.

2. The second aspect to take into consideration is whether the CubeSat passes through the $0°$ angle. In this case, if not accounted for, the CubeSat's PD controller can set a huge impulse when the error is actually small. For instance, if the set-point of the CubeSat is small:

   - If the current Yaw angle is $1°$ which is inside the tolerance set for the Fine control. PID is triggered instead of the coarse pointing mode and the PD response will be small, hence, stabilizing the CubeSat.
   - If the current Yaw angle is $359°$, the difference between the angles is very large so the PD will trigger a huge impulse to control it. However, in reality, $359°$ and $0°$ are below $< 1°$ of difference so the PD shall set a small impulse. To solve this problem, the angles are shifted $180°$. Consequently, the setpoint now becomes $180°$ and the actual yaw angle is $179°$ which will produce a difference of $1°$.
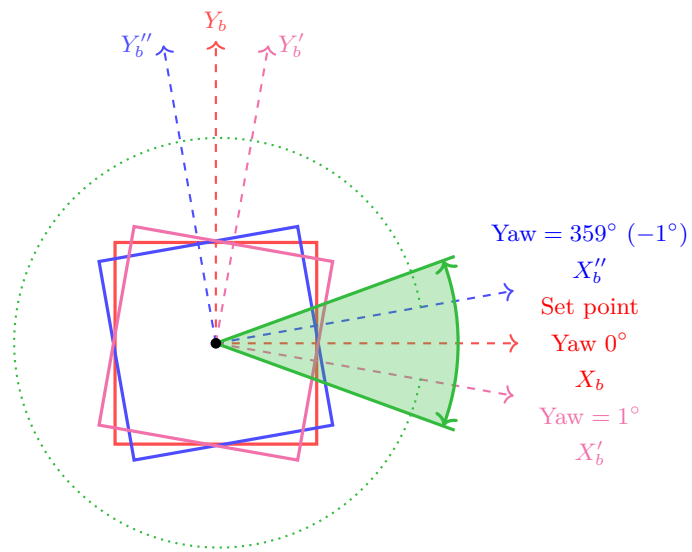
Figure 7.12 Fine Pointing diagram. Source: Own.
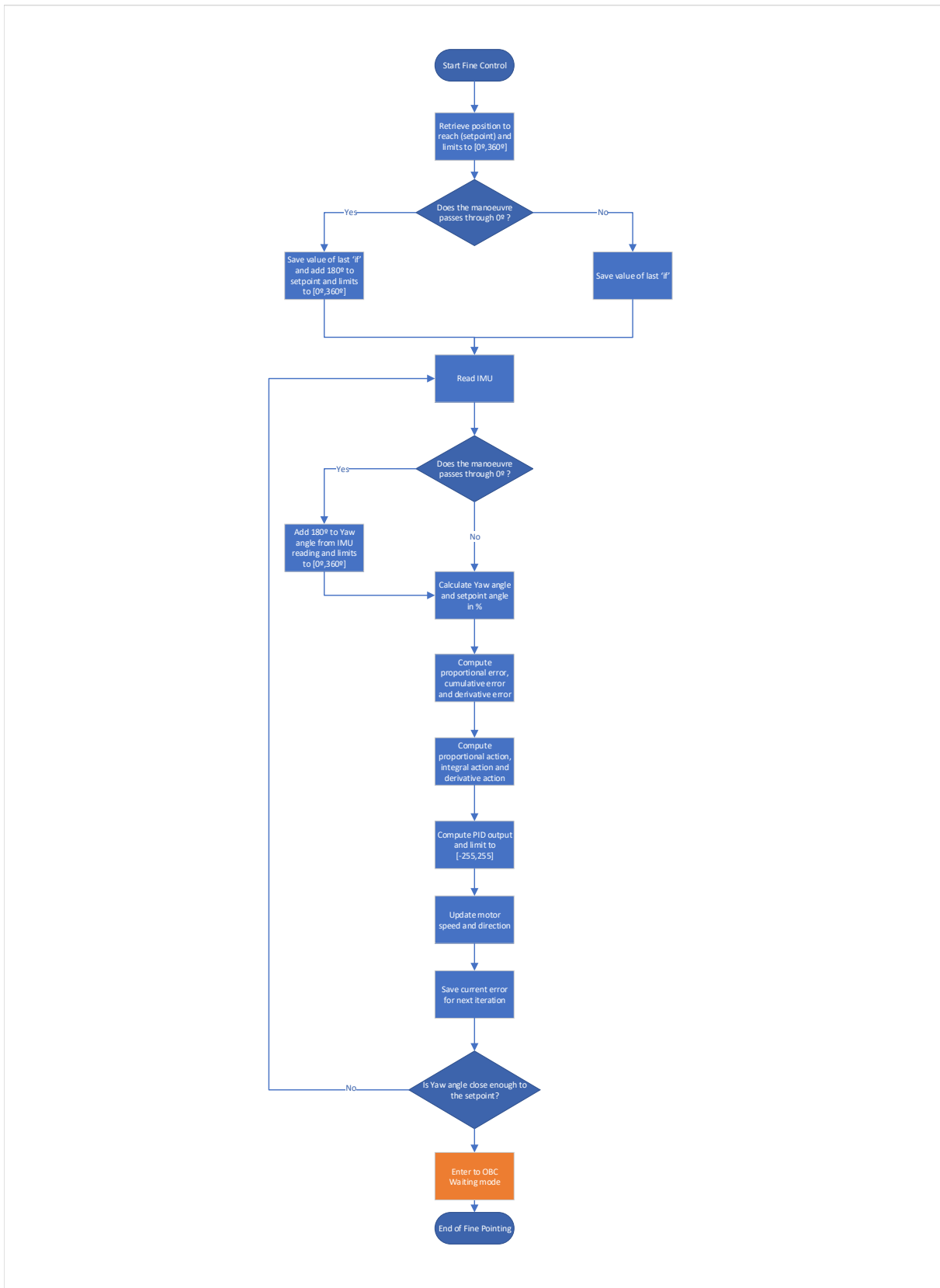
Figure 7.13 Fine pointing mode flowchart. Source: Own.

Chapter 8

Performance and Tests

This last chapter's objective is to perform tests emulating an orbit environment. The CubeSat is placed on an air bearing that will emulate outer space conditions. However, limitations on the compressor's capacity restricted the duration time of the simulations.

8.1   3D visualization

STM32 or Arduino output data from the Serial port can be pass into a Python script for real-time 3D visualization. Also, real-time orientation angles can be extracted to plot the evolution over time of those variables. However, limitations on Bluetooth communication restricted the number of bits of information sent.

The program can extract the data using the following code (this code works with the IMU_data_acquisition_orientation_all_a Arduino file from Serial port).

Projecting the yaw and pitch angles onto vPython's reference frame,
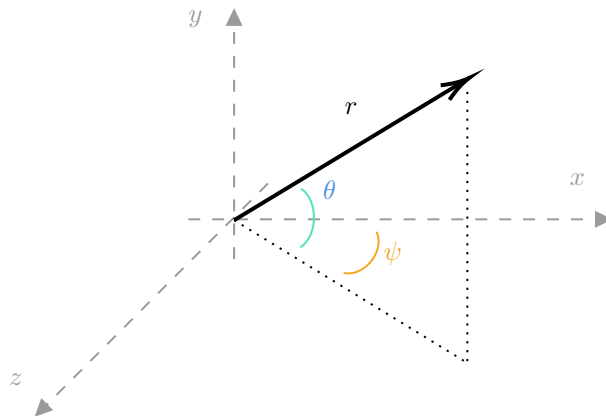
Figure 8.1 vPython reference frame. Source: Own.

The x, y and z position of any point can be expressed as:

$$x = r \cos \phi \cos \theta \tag{8.1.1}$$

$$y = r \sin \phi \cos \theta \tag{8.1.2}$$

$$z = r \sin \theta \tag{8.1.3}$$

To get the Euler angles from quaternions, Euler-Rodriguez formulas were used.
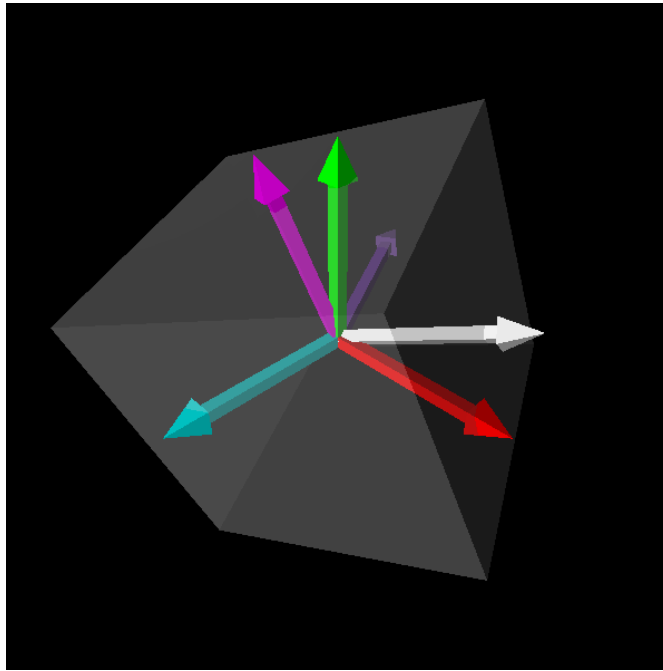
Figure 8.2 Real-time Python visualization. Source: Own.

This python code can be modified to use Euler angles directly with the complementary filter implemented in the code M. However, in the future, it is highly recommended to use quaternions as they provide more accurate results.

8.2 Setup

Figures 8.3 shows the Air Bearing used in the project. This testbed includes a pivoting platform that allows the free movement of a nanosatellite and the design of a CubeSat 1U fully equipped to control its spatial orientation [67].

(a) Air bearing structure. Source: Own.

(b) Air bearing with CubeSat mount. Source: Own.

Figure 8.3 Air bearing. Source: Own.

A compressor is located below the air bearing, the compressor blows air into the air bearing structure so when the CubeSat placed on the Air bearing a near-zero friction environment system is created. This system will allow reproducing the CubeSat condition in space. Take a look at Figures 8.4 where the fully modular CubeSat is shown with its solar panels deployed and closed. The final assembly of the CubeSat is represented in Figures 8.5 and 8.6.
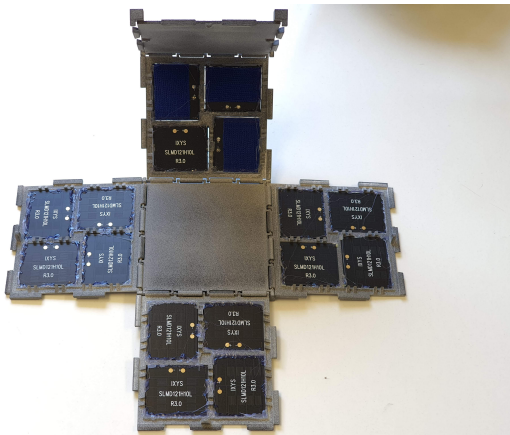
(a) Cubesat solar panels. Source: Own.
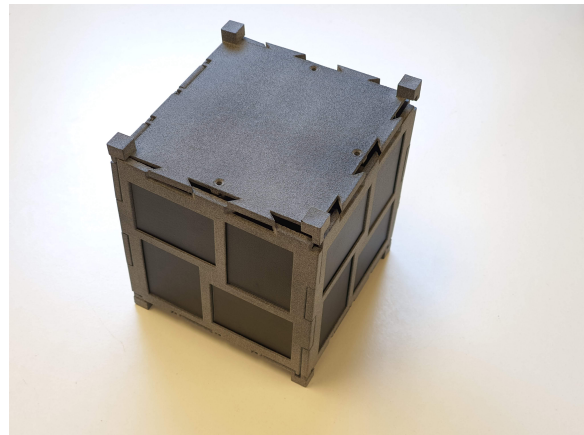
(b) CubeSat structure. Source: Own.

Figure 8.4 Modular CubeSat. Source: Own.
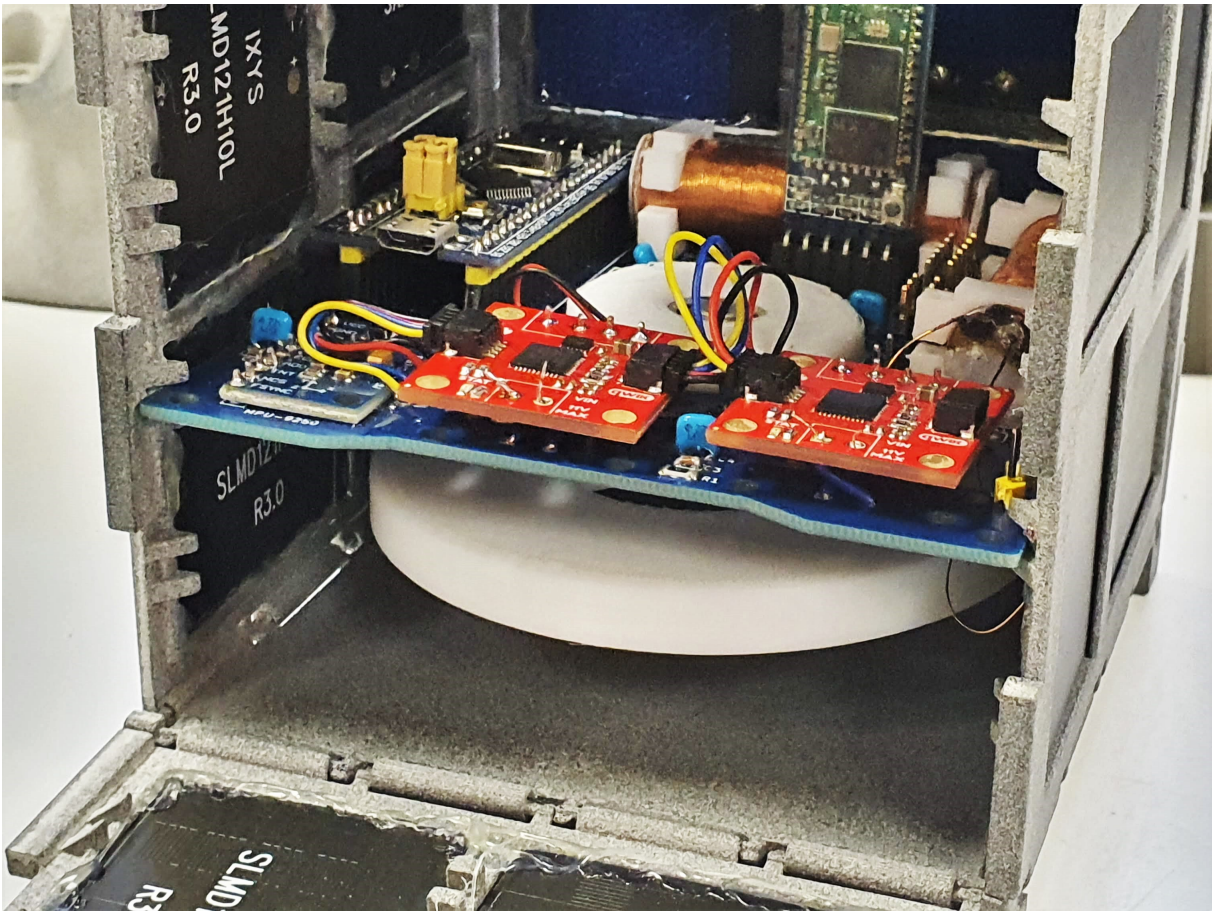
Figure 8.5 PCB inside the CubeSat. Source: Own.
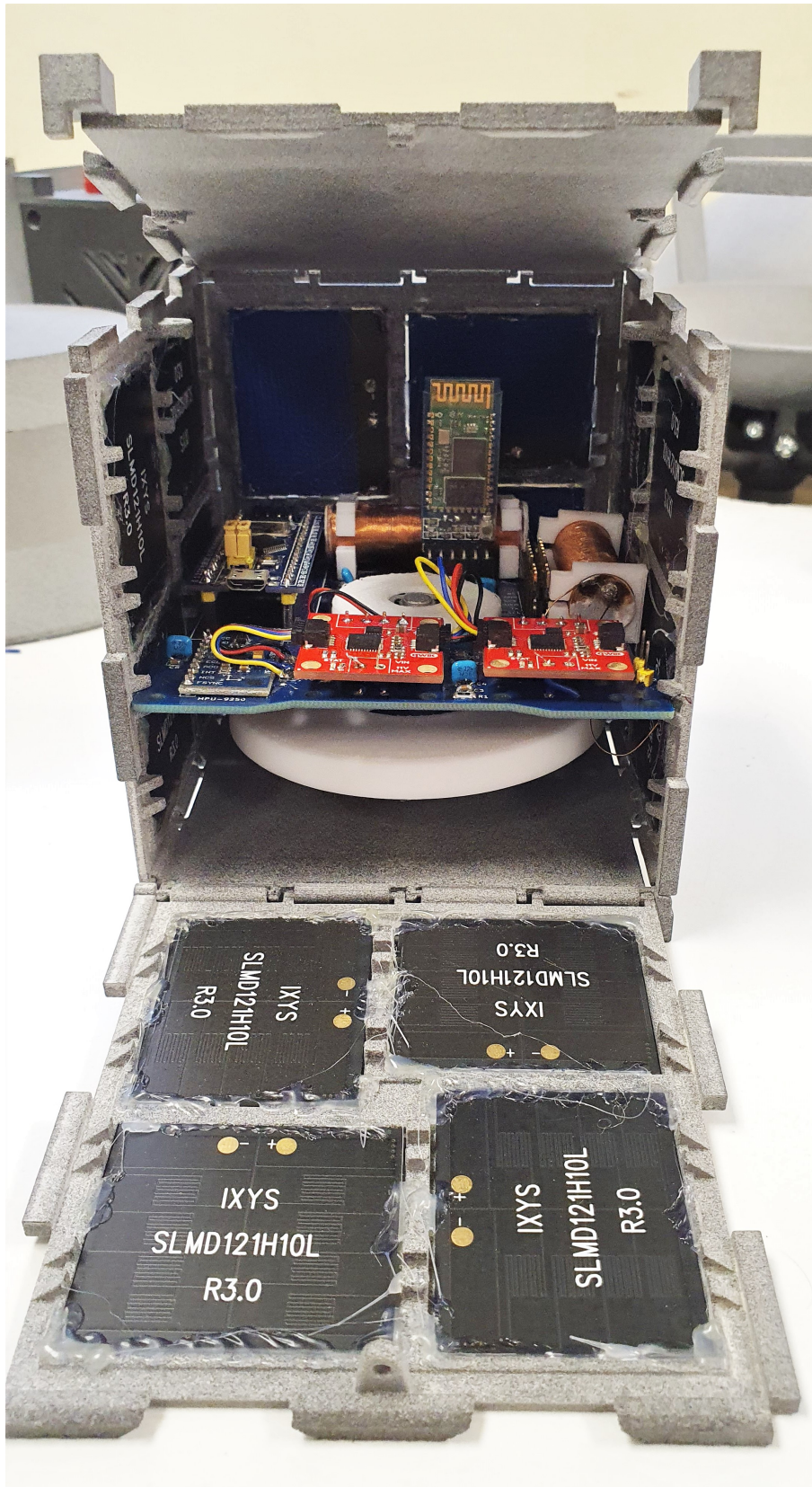
Figure 8.6 PCB inside the CubeSat (opened). Source: Own.

The final mounted CubeSat on in the air bearing is shown in 8.7. This will permit 3DOF control.

However, the project focuses on a 1DOF control. Consequently, the testing phase was done using 8.8.
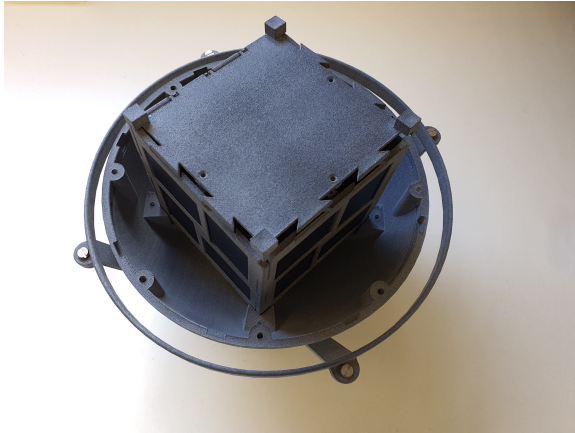
(a) Cubesat mounted on Air Bearing. Source: Own.

(b) Full CubeSat structure mounted on Air Bearing. Source: Own.

Figure 8.7 Cubesat on Air Bearing. Source: Own.
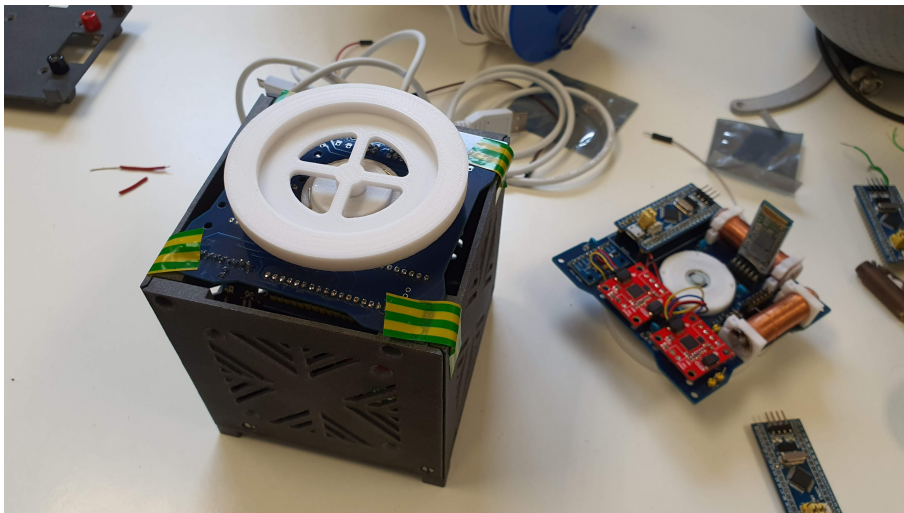
Figure 8.8 Reaction Wheel control test with CubeSat configuration. Source: Own.

8.3   Testing phase

First, before engaging in the testing phase, the user must calibrate the magnetometer and acceleration sensors. In order to calibrate the magnetometer, one must follow O guidelines inside the code.

Magnetorquers are calibrated by drawing an "eight" in the air. The calibration usually takes 1-2 minutes and if succeeded the values of calibration are returned to the user so the user must annotate the newest values to the code.
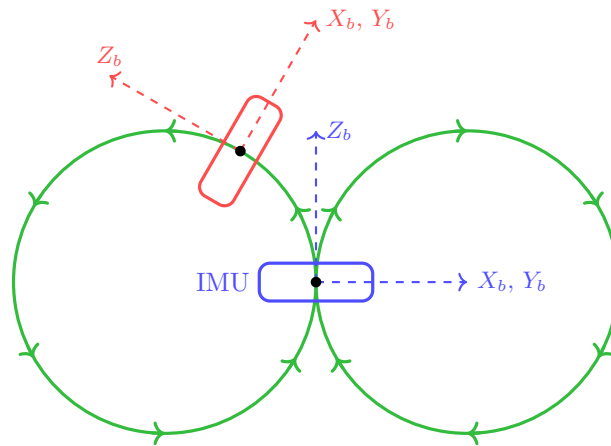
Figure 8.9 GY-9250 IMU magnetometer calibration. Source: Own.

It is important to mention that sending very large messages at a high transfer speed rate may cause the Bluetooth module to freeze, thus, some tests were done twice so in the first one we obtain the orientation angles and in the second one, we obtain accelerometer and gyroscopes measurements as well.

8.3.1   IMU Reading mode test

The first test performed was a IMU reading mode mode_IMU_reading(). This test aims to examine the Air bearing's stability and how is the data from the different sensors.

Figure 8.10 shows how the CubeSat begins to rotate after 12 seconds when the air bearing was engaged. The conclusions are that if the user does not inject air to the air bearing smoothly, there might be an initial perturbation due to the asymmetry of weight distribution that causes the CubeSat to rotate. Thus, the air injection must be performed gradually and smoothly.
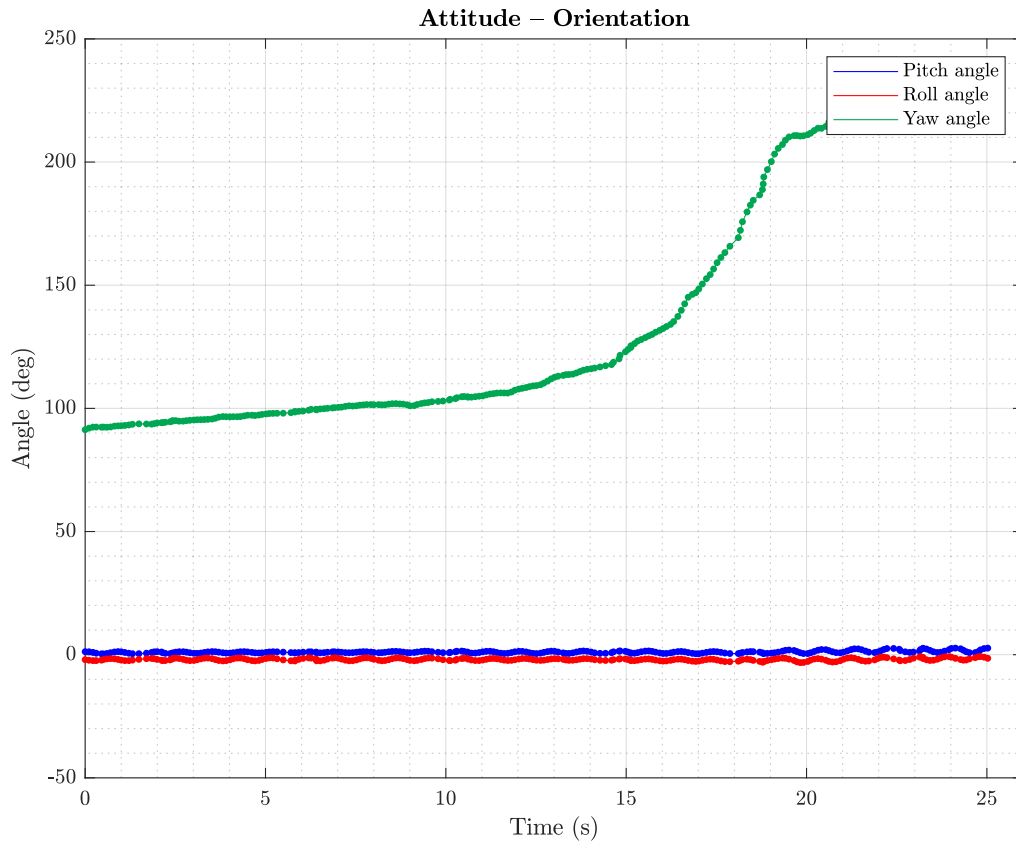
Figure 8.10 Attitude orientation. Source: Own.
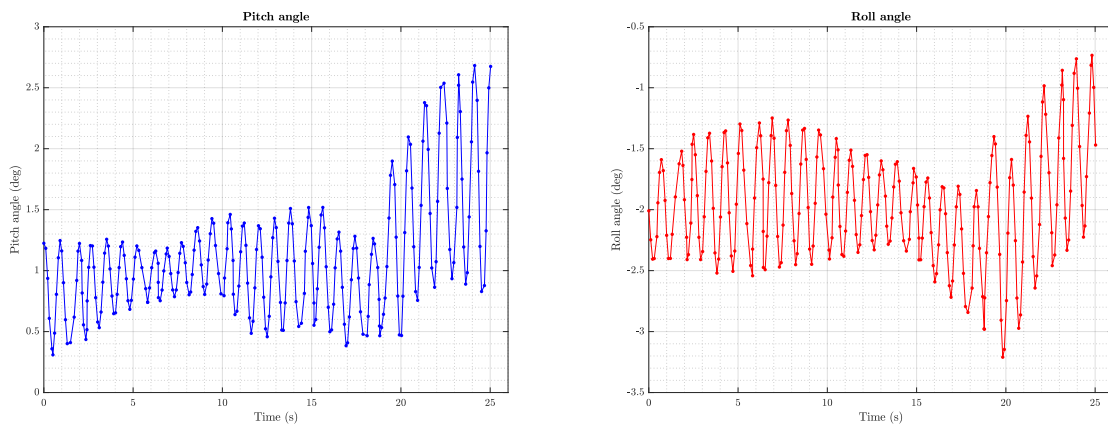
(a) Pitch angle. Source: Own.

(b) Roll angle. Source: Own.

Figure 8.11 Pitch and roll angles. Source: Own.

Notice how pitch and roll angles are oscillating back and forth between a set of angles. These results thus need to be interpreted with attention. This phenomenon is due to the vibrations of the CubeSat. As the motor is spinning, the vibrations are not damped by any means so the whole PCB vibrates and, indeed, the IMU attached to it. However, these angles are nearly 0, showing that the CubeSat is certainly

horizontal.

8.3.2   Motor ON OFF mode test

The following test aims to turn on the motor until it reaches some velocity and then stops the motor. Notice Figure 8.12, at time $t = 0$ s the user sets the motor to rotate at it's maximum speed and it is clearly seen how the yaw ψ angle goes from the initial angle $\psi_0 = 140°$ approximately to $\psi = 360°$ completing a lap. When the emergency stops kicks in at $t = 12$s approximately, the CubeSat's angular rotation slows down, which can be interpreted as the slope of the yaw angle. Contrary to our expectations, in which the team thought the CubeSat to maintain a constant angular velocity once the emergency stop is activated, Results show the CubeSat not only slows down but rotates in the opposite direction.
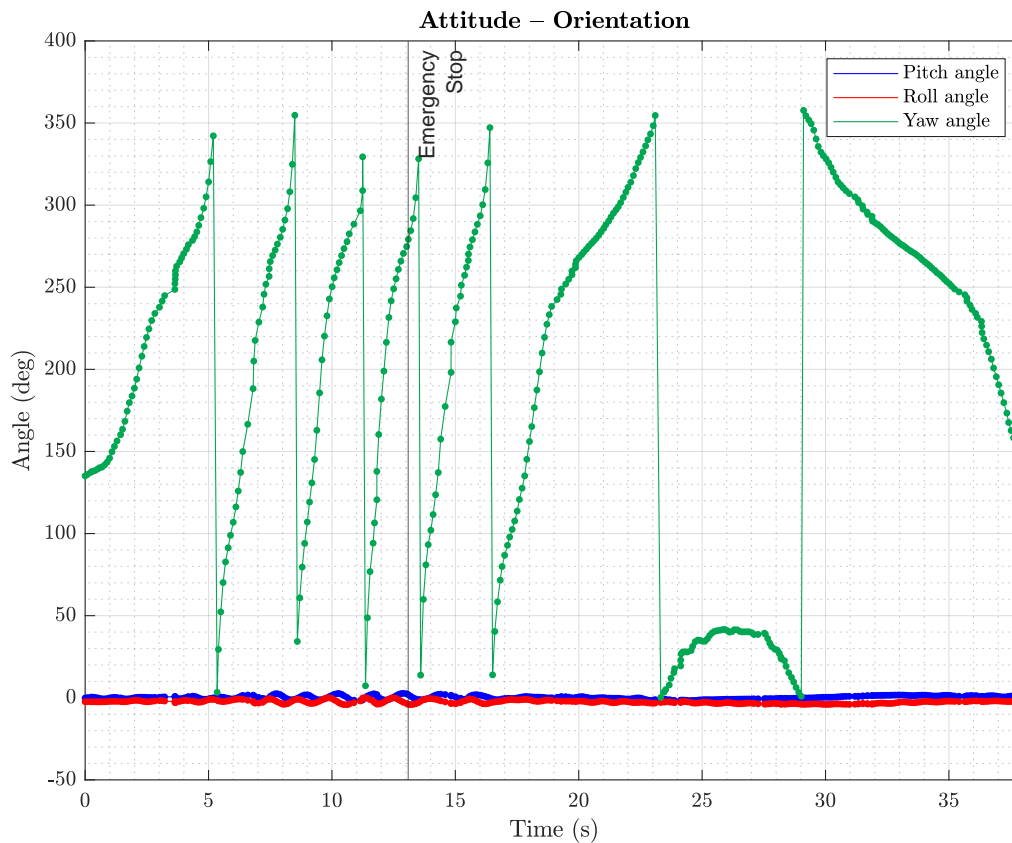
Figure 8.12 Attitude orientation. Source: Own.

(a) Pitch angle. Source: Own.   **(b)** Roll angle. Source: Own.
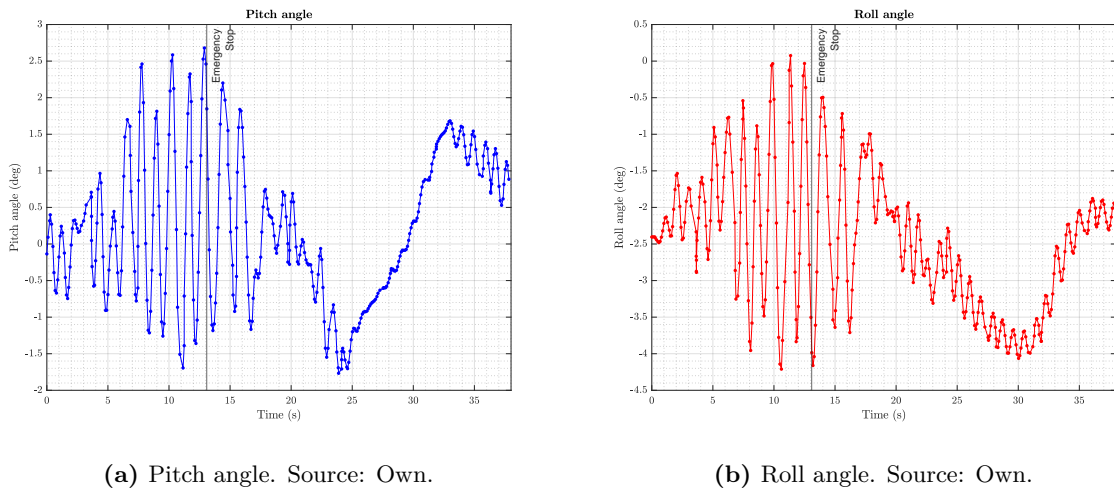
Figure 8.13 Pitch and roll angle. Source: Own.

The physical interpretation of the occurrence can be understood in the following way. After the motor is turned, the CubeSat eventually will arrive at an equilibrium phase where the entire CubeSat and the reaction wheels inertia is countervailed. However, when the reaction wheel slows down its velocity rotating it causes a disruption in the equilibrium of forces and after a while, the CubeSat's rotation due to leftover inertia is bigger than the reaction wheel's leftover inertia. Thereby, the CubeSat rotates in the opposite direction.

8.3.3 Motor ON OFF mode test

Beneath is presented the same motor ON and OFF mode as the prior test. However, this time, other set pf data are recovered instead of the three orientation angles, accelerometer measurements on x and y axis are shown as well as the angular velocity in z direction.

When the motor is on, one can clearly notice how the accelerometers are induced a lot of noise.
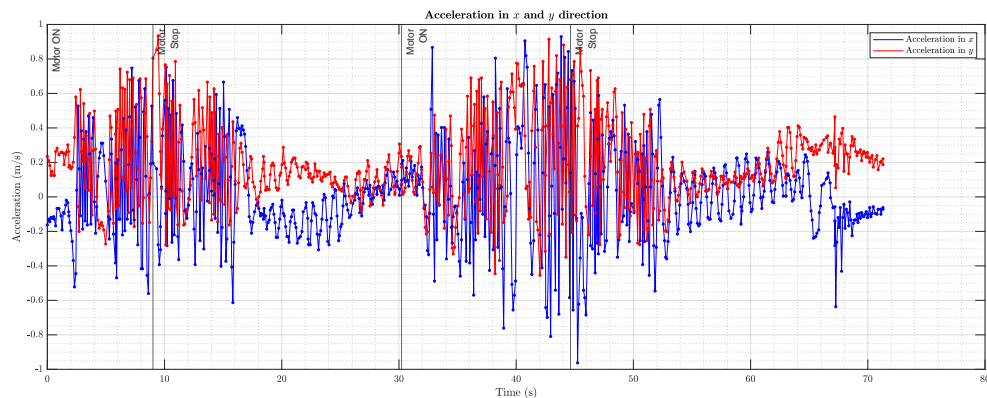
Figure 8.14 Acceleration in x and y direction. Source: Own.
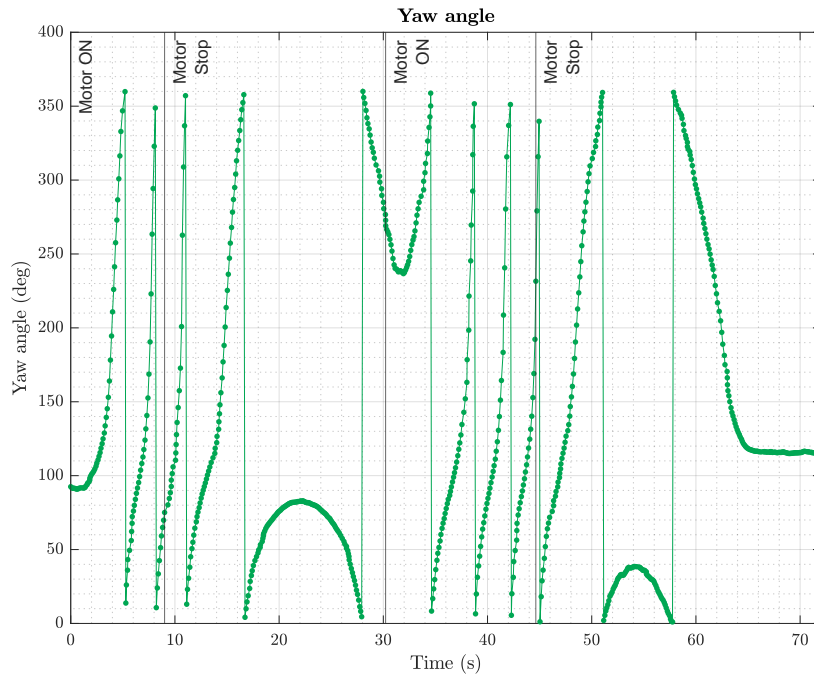
Figure 8.15 Yaw angle. Source: Own.
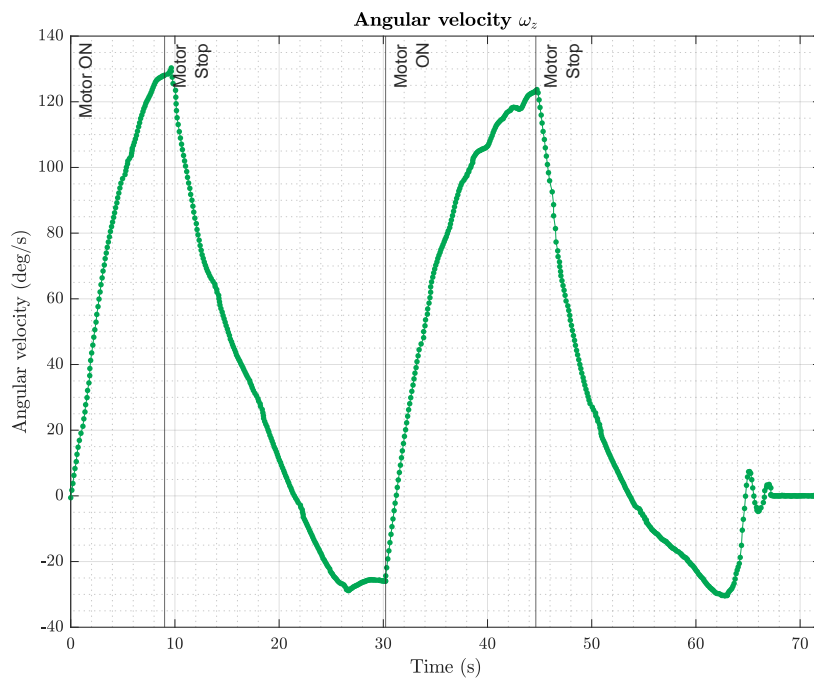
Figure 8.16 Angular velocity in z direction. Source: Own.

In Figure 8.16 we can observe clearly when the motors are stopped where the angular velocity drops down. Taking a look at the yaw angle, one notice how the CubeSat turns makes 4 complete laps before stopping at $t \approx 22$ s to rotate in the opposite direction.

8.3.4 Coarse pointing control mode test

The next test encompasses the ADCS control. For this test, the intention is to perform a full coarse and fine control. The initial angle $\psi_0 = 150$ approximately and the desired angle is to rotate 100 degrees in the clockwise direction to achieve 250 degrees approximately.
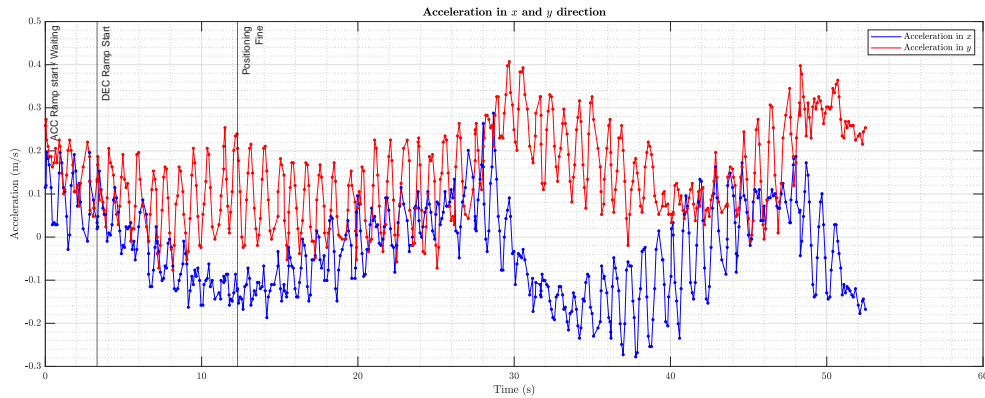
Figure 8.17 Acceleration in x and y direction. Source: Own.
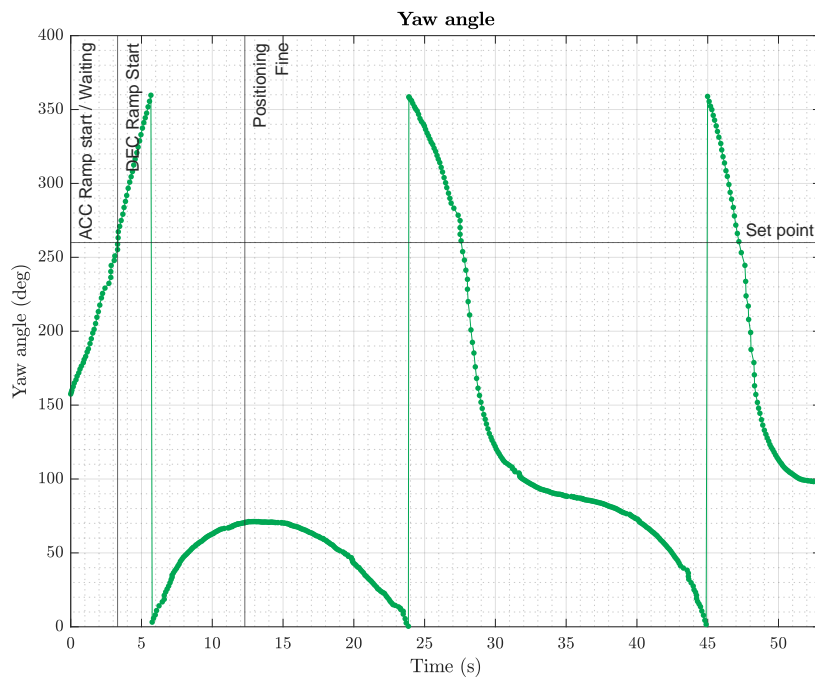
Figure 8.18 Yaw angle. Source: Own.
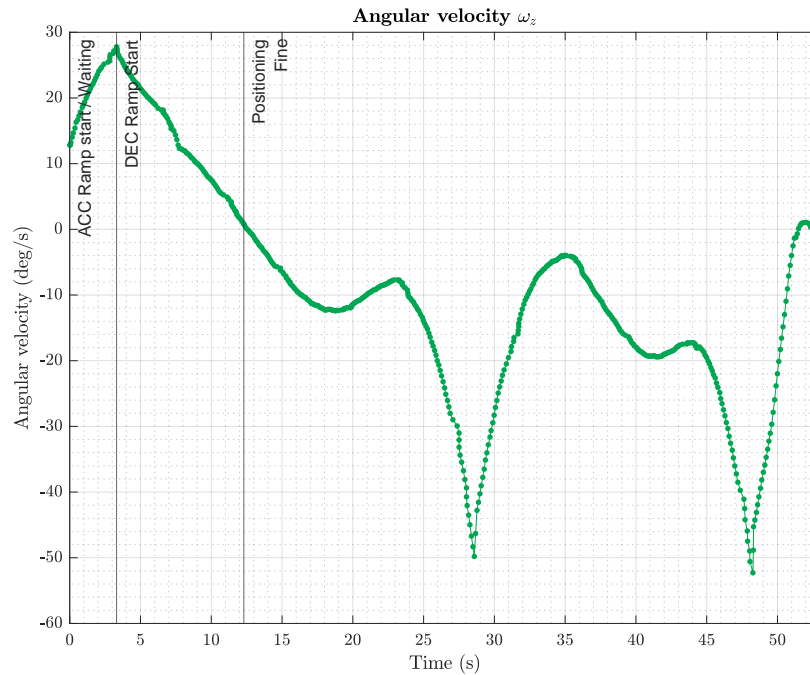
Figure 8.19 Angular velocity in z direction. Source: Own.

An appreciable change in yaw and angular velocity happens after the deceleration ramp is started (DEC Ramp start). The angular velocity rapidly decreases in $t \approx 2.5$ s as it surpassed the set point. One important remark is that the impulse gives to the motor is just an impulse. A simple impulse has practical constraints as it sets the motor to its maximum speed. However, ramps will provide better precision. Then when fine pointing mode is activated at $t = 12 - 15$ s where PD control kicks in.

Nevertheless, the proportional, derivative constants are yet still to be corrected since the CubeSat oscillates back and forth. One method to reduce the oscillation is by increasing the derivative contribution so it reduces the speed if it is approaching the setpoint too fast.

8.3.5 Fine pointing control mode test

This time, only the Fine control was tested on the CubeSat. To do so, the initial turn angle input was less $< 5°$. This way, as the angle is less than the fine tolerance angle, it will perform a Fine control just from the beginning. The initial angle is $\psi \approx 350$ and the setpoint is $346°$ approximately.
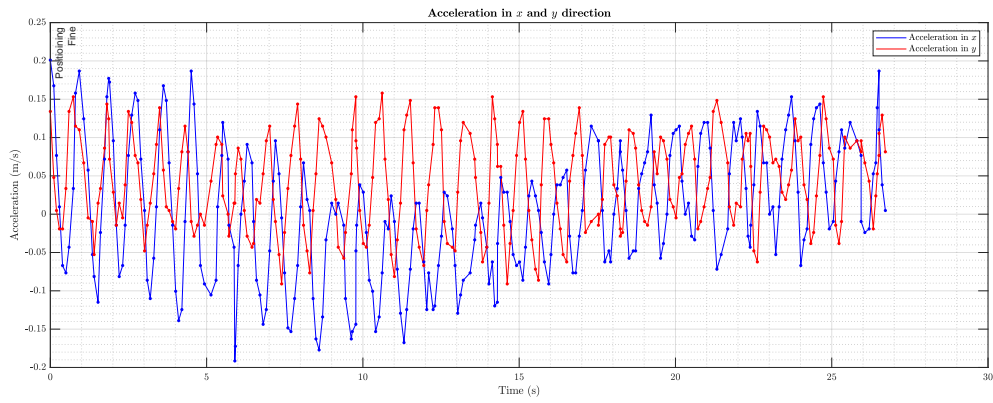
Figure 8.20 Acceleration in x and y direction. Source: Own.
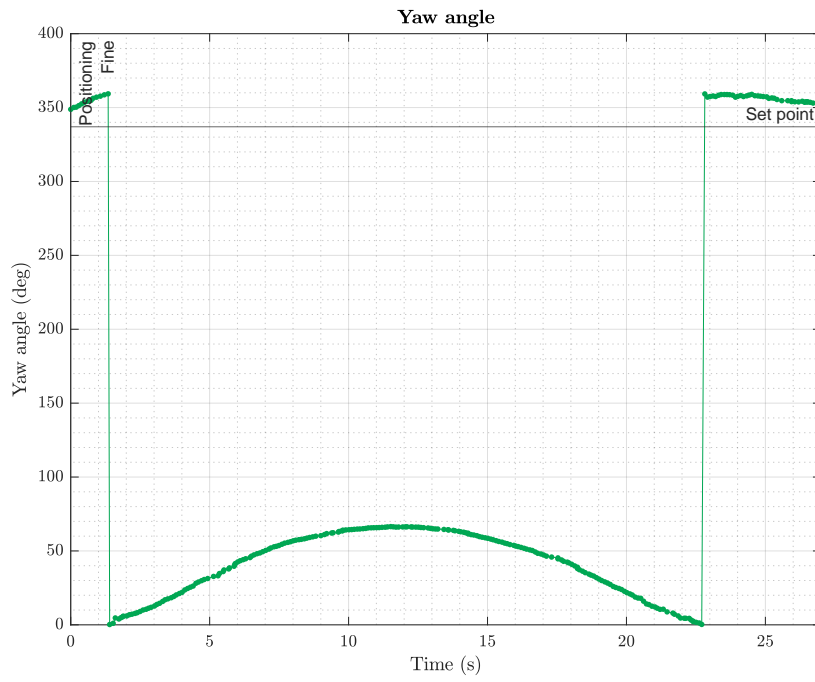
Figure 8.21 Yaw angle. Source: Own.
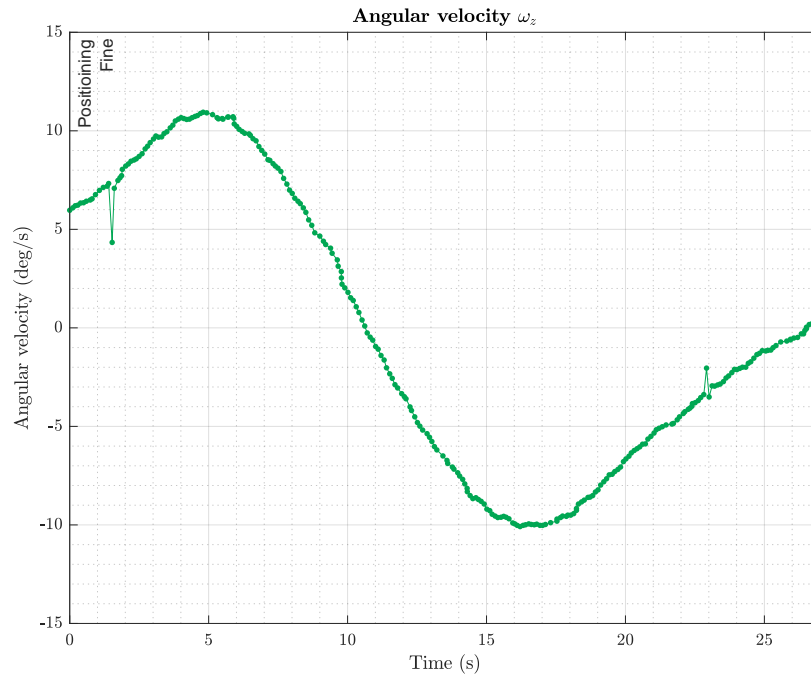
Figure 8.22 Angular velocity in z direction. Source: Own.

Figure 8.22 shows how the CubeSat tries to stay still within the setpoint zone. As mentioned earlier, PD constants need to be tuned accordingly since the proportional gain is yet very strong and overshoots the setpoint.

8.3.6 Fine pointing control mode test (PCB with magnetorquer)

Finally, the last test intends to add the weight of the magnetorquers to the CubeSat configuration. As the magnetorquers were placed in one corner of the PCB, it destabilizes the CubeSat. This design error is not complex to solve. The solution is to place some counterweight on the other side of the CubeSat.
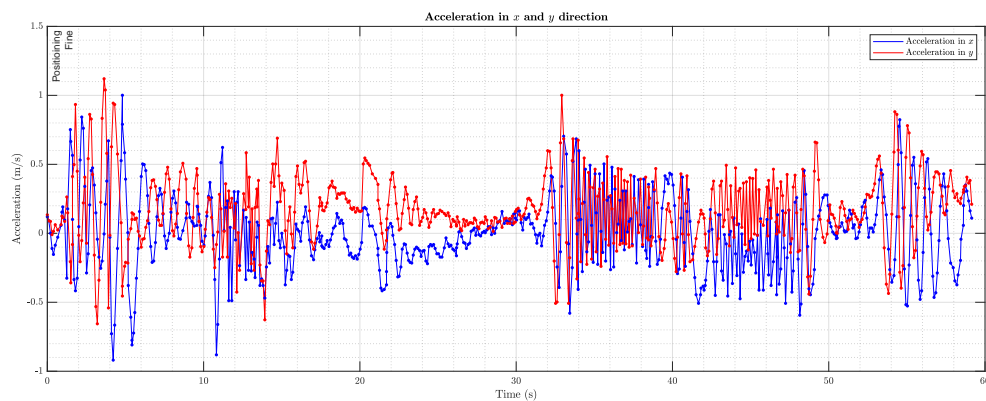
Figure 8.23 Bluetooth setup schematic. Source: Own.
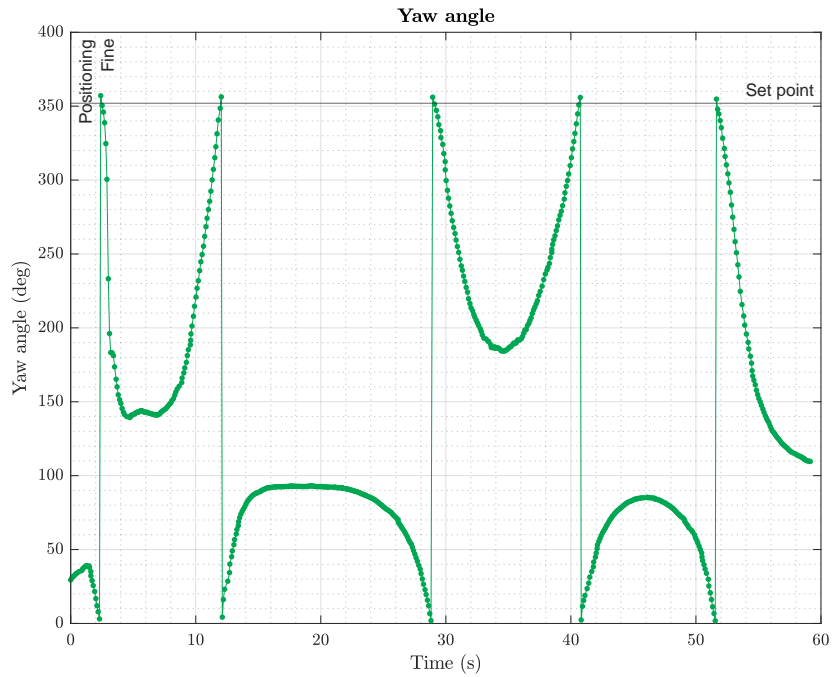
Figure 8.24 Bluetooth setup schematic. Source: Own.
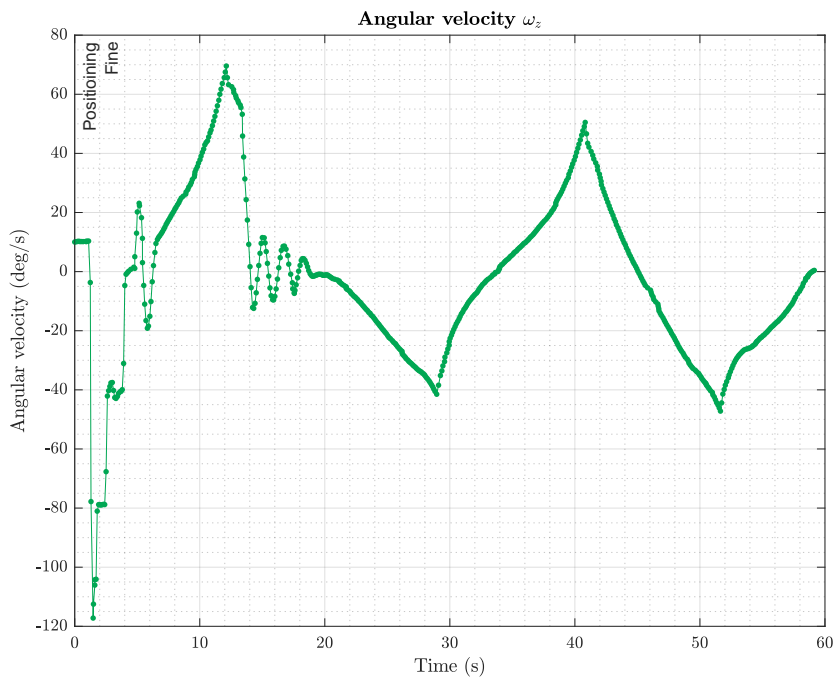
Figure 8.25 Bluetooth setup schematic. Source: Own.

Eventually, the CubeSat's PD controller seems to work as expected. Again, the PD controller still needs some tuning since more weight is added to ensure the CubeSat avoid overshooting and oscillatory motion.

Chapter 9

Planning of the project

This chapter illustrates the whole planning for the project. A Work Breakdown Structure is developed as well as a Gantt Chart to reproduce the schedule and deadlines for the project.

9.1 Work Breakdown Structure

The deliverable-based Work Breakdown Structure demonstrates the relationship between the project deliverables (i.e., results, products) and the scope (i.e., work to be executed).

9.1.1 Tasks identification

Table 9.1 shows the tasks performed during the project:

Table 9.1 Work Breakdown Structure task identification. Source: Own.

TASK CODE	TASK IDENTIFICATION
1.	**Cubesats**
1.1.	State of the art
1.2.	Applications
1.3.	ADCS Instruments
2.	**Attitude Determination and Control Subsystem**
2.1.	Orbital Mechanics
2.2.	Inertial Measurement Unit
2.3.	Euler Angles and Quaternions
2.4.	Disturbance torques
2.5.	Control Algorithm
3.	**Communication and Telemetry**
3.1.	Tracking
3.2.	Point-to-point Communication
3.3.	Data acquisition and processing
3.4.	Real-time Data plotting
4.	**Reaction wheels**
4.1.	Physics analysis
4.2.	Motor Selection
4.3.	Motor Driver
4.4.	CAD Design
5.	**Assembly**
5.1.	PCB Design
5.2.	Final Assembly
6.	**Optimization**
6.1.	Testing
6.2.	Data Verification
6.3.	Sensibility and precision
7	**Report writing**

The amount of hours spend in each task can is shown in Table 9.2.

Table 9.2 Tasks and working hours. Source: Own.

TASK CODE	Duration
1.1.	5
1.2.	5
1.3.	5
2.1.	5
2.2.	10
2.3.	5
2.4.	10
2.5.	20
3.1.	5
3.2.	5
3.3.	5
3.4.	5
4.1.	10
4.2.	5
4.3.	5
4.4.	5
5.1.	5
5.2.	10
6.1.	15
6.2.	10
6.3.	10
7.	222
TOTAL	382

9.1.2   Work Breakdown Structure
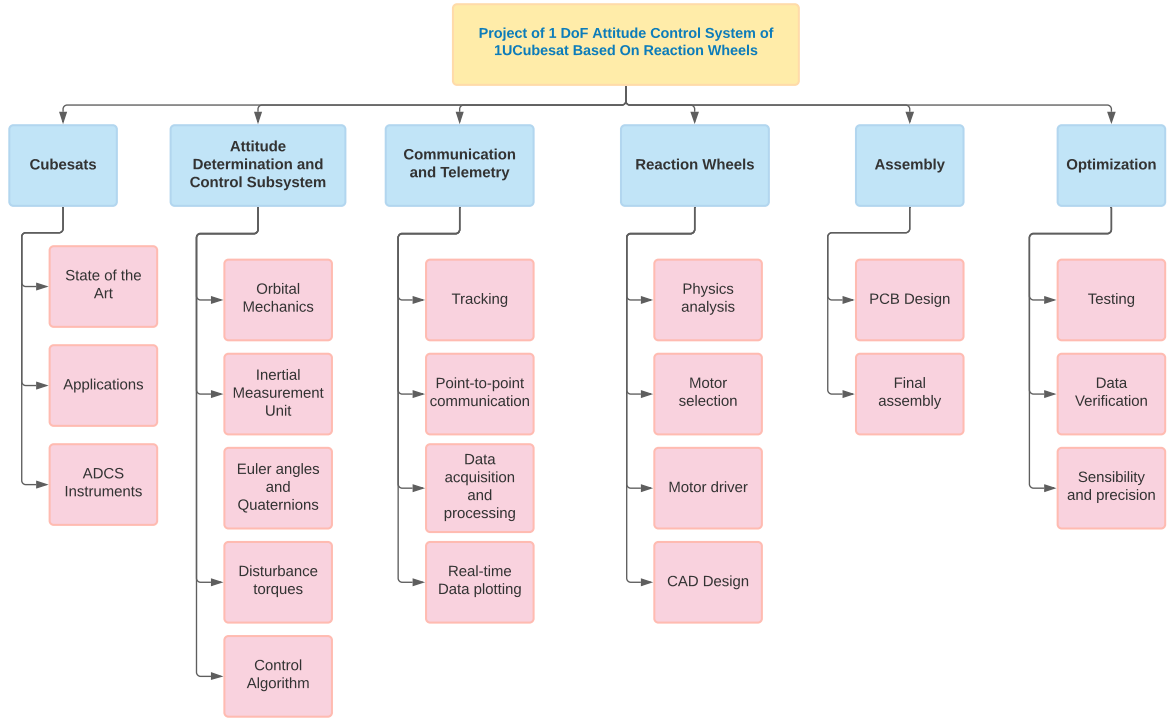
Figure 9.1 Work Breakdown Structure. Source: Own.

9.2 Gantt Diagram

The following Gantt diagram is used to plan the project management and schedule. This tool permits to show the activities (i.e., tasks and events) displayed against time. On the left of the chart is a list of the activities and along the top is a suitable and appropriate time scale. Each activity is presented with its time allocation (see next page).

This allows to get an overview of the number of activities there are, when they begin and how long each activity is scheduled to last and also keep in mind when the project is intended to end.

Thesis - ADCS Cubesat	start	end
Project Charter	**28/01/21**	**19/03/21**
Documentation and information Res...	28/01	11/03
Aim of the Project	08/02	23/02
Justification	08/02	28/02
Scope of the Project	08/02	11/03
Project Requirements	08/02	11/03
State of the art	01/02	16/03
Work Breakdown Structure	12/03	16/03
Gantt Diagram	12/03	16/03
Project Charter Submission	19/03	19/03
Development	**08/02/21**	**18/06/21**
Structural requirements	08/02	27/02
Attitude Control physics and mathe...	24/02	17/04
Disturbance torques	27/02	17/04
Inertial Measurement Unit and code	10/03	14/05
Bluetooth Module and code	10/03	28/04
Real time data plot and code	10/03	16/04
Reaction Wheel Motor & Driver	11/03	24/04
Reaction Wheel Design	21/03	07/05
Telemetry and Telecommands	03/04	14/05
Design of a custom PCB	03/04	21/05
Coarse and Fine pointing mode	03/04	26/05
Control Algorithm	03/04	14/06
Error, stability, detumbling and man...	16/05	18/06
Optimization	16/05	18/06
Thesis Report Writing	**01/02/21**	**22/06/21**
Introduction	01/02	28/02
Documentation and information Res...	01/02	19/06
State of the art	01/02	07/04
Mathematical description of the phys...	01/02	06/04
Attitude Determination and Control ...	10/03	30/04
Hardware and Software Development	23/03	18/06
Budget	01/06	18/06
Environmental aspects	01/06	18/06
Report Revision	14/06	20/06
Report Writing	02/02	20/06
1st Follow-up	16/04	16/04
2nd Follow-up	07/05	07/05
3rd Follow-up	28/05	28/05
Final Submission	22/06	22/06
Presentation	**12/07/21**	**12/07/21**
Project Exposition	12/07	12/07

Chapter 10

Environmental Impact

This chapter intends to analyse the environmental impact of the scope of the project. Despite the major advances in the aerospace area, there are still secondary effects that need to be taken into account.

Sustainability is defined as the capacity of preserving an activity within a long period of time without harming the environment. However, the most accepted definition of **sustainability** is provided by the *World Commission on Environment and Development Definition.*

The major dilemma of putting satellites into orbit resides in the space debris it produces. At the same time, one of the major concern among scientists around the world is not only the impact it might have on other exploration missions but also the light pollution it has in the celestial vault.

Any project that implies a satellite shall analyse the effect among other outer space activities. for instance, the accumulation of space debris can lead to a chain reaction that can cause severe damage to other missions. Besides, the carbon footprint of the project is nearly zero as it has solar panels and most of the operational time it will not rely on any other source.

Since this project is undertaken entirely in the laboratory there are no more environmental issues than the CO_2 emitted during the analysis and development of it. See Table K

$$m_{CO_2} = \text{Impact factor} \left[\frac{\text{kg } CO_2}{\text{kW} \cdot \text{h}} \right] \times \text{Energy} \left[\text{kW} \cdot \text{h} \right] \tag{10.0.1}$$

Table 10.1 Power consumption and kg CO_2 emission

Activity	Time [h]	Power kW	Impact factor [kg CO_2/kg · h]	kg CO_2
Laptop	382	0.30	0.2	22.92
3D manufacturing	32	0.35	0.2	2.24

Thus, an approximate value of 25.16 kg of CO_2 was emitted during the development of this project.

Chapter 11

Budget

This section intends to analyze the budget of the personnel time invested in working in this investigation performed and the overall budget to conduct the present project.

11.1 Materials budget

To undertake the project, several instruments were used and many components were utilized. Table 11.1 shows the electrical components bought to assembly the ADCS Subsystem.

The price of the components are subject to vary depending on the shopping site. The corresponding values are taken from amazon.es.

Table 11.1 Materials budget. Source: Own.

Electrical components			
Concept	**Quantity**	**Unitary price (€)**	**Total price (€)**
Breadboard	2	4.99 €	9.98 €
Cables	1	6.99 €	6.99 €
Arduino Nano	1	21.78 €	21.78 €
STM32 Blue Pill	1	13.49 €	13.49 €
USB ST-Link V2	1	10.27 €	10.27 €
BNO055 IMU	1	34.95 €	34.95 €
GY-9250 IMU	1	8.23 €	8.23 €
SparkFun Qwiic Motor Driver	1	12.52 €	12.52 €
Qwiic cable	1	1.50 €	1.50 €
Motor Mabuchi RF-300EA-1D390	1	6.91 €	6.91 €
Logic Level converter	1	9.88 €	9.88 €
Bluetooth module HC-06	1	10.99 €	10.99 €
Air storage tank Kunshan Abama	1	200 €	200 €
TOTAL			**347.49 €**

As regards the additive manufactured pieces the associated costs depend on the manufacturing techniques. The air bearing system was manufactured using *Hewlett-Packard* Multi-Jet Fusion 3D printing technologies with PA12CB as the material. However, reaction wheels and supports were made using PLA.

Table 11.2 3D printed pieces budget. Source: Own.

3D printed pieces			
Concept	**Quantity**	**Unitary price (€)**	**Total price (€)**
Reaction Wheel	2	1.00€	2.00 €
Reaction wheel support ring top	2	0.10 €	0.20 €
Reaction wheel support ring bottom	2	0.08 €	0.16 €
Magnetorquer support	8	0.02 €	0.16 €
Air Bearing	1	800 €	800 €
TOTAL			**802.52 €**

The PCB was entirely designed by PLATHON's team, however, the manufacturing was sent to JLCPCB as the university could only make 1 layered PCBs.

Table 11.3 PCB manufacturing budget. Source: Own.

PCB			
Concept	**Quantity**	**Unitary price (€)**	**Total price (€)**
PCB	5	1.312 €	6.56 €
Shipping cost (JLCPCB)	1	30.40 €	30.40 €
TOTAL			**36.96 €**

Regarding the software tools used to develop the control algorithm, the CAD design and the simulation analysis are shown below 11.4. However, being a student of the Technical University of Catalonia - BarcelonaTech these software tools were no charged since they were educational licensed, nevertheless, for external investigators or companies these budgets must be accounted.

Solidworks was used to design the reaction wheel model and the support pieces. MATLAB was used to develop the atmospheric model and the disturbance torques calculations. Arduino IDE was used to develop the control algorithm for the CubeSat. VS Code was used to perform python 3D simulation. Fritzing tool was used to develop the diagrams and finally, KiCad was used to develop the PCB.

Table 11.4 Software licences budget. Source: Own.

Software licenses			
Concept	**Quantity**	**Unitary price (€)**	**Total price (€)**
Solidworks	1	1295 €/year	1295 €/year
MATLAB	1	800 €/year	800 €/year
Arduino IDE	1	0 €	0 €
VS Code	1	0 €	0 €
Fritzing	1	8 €	8 €
KiCad	1	30.40 €	30.40 €
TOTAL			**2133.40 €**

According to the Idescat (Statistics Institute of Catalonia), the mean hourly salary for an professional investigator, scientist or engineer [68] is around 25.42 €/h. Thus, the time invested in this research project was about 382 hours (see Table 9.2).

Table 11.5 Personnel salary budget. Source: Own.

Personnel salary budget			
Concept	Hours (h)	Hourly salary (€/h)	Total (€)
Aerospace Engineer	382	25.42 €/h	9710.44 €
TOTAL			**9710.44 €**

Finally, summing up all the budgets of this project, the total budget can be calculated:

Table 11.6 Total budget. Source: Own.

Total budget	
Concept	Budget (€)
Electrical components	347.49 €
3D printed pieces	802.52 €
PCB manufacturing	36.96 €
Software licenses	2133.40 €
Personnel salary	9710.44 €
TOTAL	**13030.81 €**

Chapter 12

Conclusions

To sum up, the present thesis describes the attitude determination and control subsystem for PLATHON's project. According to the requirements set in the initial phase of the project, the reaction wheels were designed.

The first phase of the project encompassed a review of astrodynamics in which it reviews some important concepts such as reference frames and how to describe the attitude with information about Euler angles and Quaternions. After that, the most important disturbance torques were studied. The spacecraft in space is subjected to different disturbance torques which if not considered can cause undesired torques or inaccurate measurements. The main contribution was due to

The second phase of the project provides an in-depth analysis of the Reaction wheels design that must comply with the requirements of the project and develop the code for the control. All the components studied were assembled into the PCB with no issues. Finally, as regards the control algorithm developed has been divided into two principal modes, fine and coarse pointing mode. As for the coarse pointing mode, an impulse is set to achieve the desired set-point. When the CubeSat orientation is near the desired set-point, fine control steps into using a PD controller.

The main objective of the thesis is accomplished, which was to design a fully operational 1DOF attitude control system for a CubeSat. All software is open-source and the different modes of operation work perfectly with some tuning. The code is accessible for further development and can be extrapolated to a 3DOF system.

12.1   Future work and recommendations

The following section gives some suggestions based on the study during this thesis and recommendations for future hardware and software improvements.

- Although GY-9250 IMU module provides accurate measurements, the code provided does not use the internal DMP for extracting the orientation angles. These angles are calculated from raw in-

struments so the error is slightly higher and it loads more code for the microprocessor to calculate. This restriction is faced since there are no libraries that benefit the IMU's DMP using SPI communication protocol but there is a library that uses I^2C protocol which enables DMP. Nevertheless, it is always better to use quaternions for attitude determination and then transforming them into Euler angles.

- Calibration method for the IMU is a slow process and despite the measurements have small errors, the drift on the gyroscope for long-term missions cannot be avoided. Alternative instruments can be used to readjust the gyroscope's error or a self-calibration mode in the control algorithm can be added.

- Additional attitude determination sensors, such as a gyroscope or an Earth horizon sensor, are required to improve angular estimation accuracy during an eclipse.

- Reaction wheel design performs as expected and has a lot of inertia. However, the implemented wheel is for a 1DOF system. If a 3DOF system is added, the reaction wheel shall be smaller and perhaps new motors with higher torques should be considered, for instance, brushless motors.

- If the attachment of the reaction wheel and motor is not perfectly aligned, the Cubesat can rotate off-axis causing precession. This issue can be solved by designing a piece that ensures the alignment of the axis in both objects.

- Results show that vibrations induce the accelerometers measurements to oscillate, thus, a damping system may be useful to retrieve smoother and more accurate data.

- As the code now employs only impulses for activating coarse pointing mode, a ramp input will reduce the overshooting and obtain more precise results.

- Extra functionalities such as a graphical interface should be developed to make a better

- Detumbling and nadir pointing modes were developed by magnetorquer team. A further step is to merge both team's software and develop control with both reaction wheels and magnetorquers which will drastically increase the accuracy of the project.

Bibliography

[1] Gonzalez, David. (2021). Integrated Hardware in the loop simulation PLATform of Optical communications in Nanosatellites. UPC. (Cit. on pp. 4, 5, 6).

[2] California Polytechnic State University. (2021). *CubeSat Design Specification*. Retrieved February 8, 2021, from https://static1.squarespace.com/static/5418c831e4b0fa4ecac1ba cd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf. (Cit. on pp. 8, 10, 133, 134, 136, 137, 138, 139, 140, 141, 143, 176)

[3] Vanreusel, Joost. (2021). Launching a CubeSat: Rules, laws, and best practice. *Cubesat Handbook* (pp. 391–414). Elsevier. https://doi.org/10.1016/b978-0-12-817884-3.00021-7. (Cit. on pp. 9, 134)

[4] ISISPACE GROUP. (2021). *CubeSat information*. Retrieved February 8, 2021, from https://w ww.isispace.nl/cubesat-information. (Cit. on p. 9)

[5] Cubesatworld. (2021). Pre-Uni Bus – CubeSat World. Retrieved March 16, 2021, from https: //cubesatworld.com/product/nanobus-1u/. (Cit. on p. 9)

[6] CERN. (2014). CELESTA (CERN Latchup Experiment STudent sAtellite). Retrieved March 16, 2021, from http://celesta.web.cern.ch/. (Cit. on p. 9)

[7] Cappelletti, C., Battistini, S., & Malphrus, B. (2020). *Cubesat handbook: From mission design to operations*. Elsevier Science. https://books.google.es/books?id=jGy4ywEACAAJ. (Cit. on pp. 10, 12, 20, 21, 31)

[8] Alén Space. (2021). A Basic Guide to Nanosatellites — Alén Space. Retrieved May 16, 2021, from https://alen.space/basic-guide-nanosatellites/. (Cit. on pp. 10, 11, 31)

[9] Space News. (2015). News from the ITU Sympoisum on Small Satellite Regulation. Retrieved June 7, 2021, from https://spacenews.com/news-from-the-itu-symposium-on-small-s atellite-regulation/. (Cit. on p. 12)

[10] Wijker, J. J. (2008). *Spacecraft structures*. Springer Science & Business Media. (Cit. on p. 13).

[11] NASA, & Starin, S. R. (n.d.). *19.1 Attitude Determination and Control Systems* (tech. rep.). (Cit. on pp. 14, 15, 16, 17, 18).

[12] Wertz, J. R. (2012). *Spacecraft attitude determination and control* (Vol. 73). Springer Science & Business Media. (Cit. on pp. 14, 30, 34, 35, 36).

[13] NASA. (1970). *Vehicle Design Criteria - Spacecraft Sun Sensors* (tech. rep.). https://ntrs.n asa.gov/search.jsp?R=19710008281. (Cit. on pp. 14, 15)

[14] Pisacane, V. L. (2005). *Fundamentals of space systems*. Johns Hopkins University/Appli. (Cit. on p. 15).

[15]   NASA. (1969a). *Vehicle Design Criteria - Spacecraft Earth Horizon Sensors* (tech. rep.). (Cit. on pp. 15, 16).

[16]   ESA. (2012). Europe dominating satellite startracker market. Retrieved June 8, 2021, from http ://www.esa.int/Enabling_Support/Space_Engineering_Technology/Europe_do minating_satellite_startracker_market. (Cit. on p. 16)

[17]   We Hack the Moon. (2021). The Inertial Measurement Unit: Mechanical Engineering Wizardry — Hack the Moon. Retrieved June 14, 2021, from https://wehackthemoon.com/tech/inert ial-measurement-unit-mechanical-engineering-wizardry. (Cit. on p. 17)

[18]   Tierno, M. Á. G., Cortés, M. P., & Márquez, C. P. (2012). *Mecánica del vuelo*. Ibergaceta. (Cit. on p. 22).

[19]   Warwick Holmes. (2017). *AERO2705 Frames of Reference, Orbits and Ellipses* (tech. rep.). University of Sydney. (Cit. on p. 22).

[20]   Curtis, H. D. (2013). *Orbital mechanics for engineering students*. Butterworth-Heinemann. (Cit. on pp. 22, 26, 41, 42, 43).

[21]   Fishbane, P. M., Gasiorowicz, S. G., & Thornton, S. T. (2005). *Physics: For scientists and engineers with modern physics*. Prentice-Hall. (Cit. on p. 23).

[22]   Anonymous. (2017). LaTeX Coordinate Transformations - Pastebin.com. Retrieved May 9, 2021, from https://pastebin.com/d1kSJmY9. (Cit. on p. 26)

[23]   Michael Fowler. (2021). Euler's Angles. Retrieved May 9, 2021, from https://galileoandei nstein.phys.virginia.edu/7010/CM_26_Euler_Angles.html. (Cit. on p. 27)

[24]   Kuipers, J. B. (1999). *Quaternions and rotation sequences: A primer with applications to orbits, aerospace, and virtual reality*. Princeton university press. (Cit. on pp. 28, 29, 43).

[25]   Junkins, J. L., & Schaub, H. (2009). *Analytical mechanics of space systems*. American Institute of Aeronautics; Astronautics. (Cit. on p. 28).

[26]   Blanco-Claraco, J. L. (2021). A tutorial on **SE**(3) transformation parameterizations and on-manifold optimization. *University of Malaga, Tech. Rep*, *3*, 6. http://arxiv.org/abs/2 103.15980 (cit. on p. 29)

[27]   NASA. (1971). *Vehicle Design Criteria - Spacecraft Aerodynamic Torques* (tech. rep.). (Cit. on pp. 30, 31).

[28]   Bowman, B. R., Tobiska, W. K., Marcos, F. A., Huang, C. Y., Lin, C. S., & Burke, W. J. (2008). A New Empirical Thermospheric Density Model JB2008 Using New Solar and Geomagnetic Indices. *Astrodynamics Specialist Conference* (cit. on pp. 31, 32, 201, 214).

[29]   SILSO Royal Observatory of Belgium, B. (2021). Monthly and smoothed sunspot number — SILSO. Retrieved May 22, 2021, from http://www.sidc.be/silso/monthlyssnplot. (Cit. on p. 33)

[30]   Larson, W. J., & Wertz, J. R. (1992). *Space mission analysis and design* (tech. rep.). Torrance, CA (United States); Microcosm, Inc. (Cit. on p. 33).

[31]   NASA. (1969b). *Vehicle Design Criteria - Spacecraft Gravitational Torques* (tech. rep.). (Cit. on p. 35).

[32]   NASA. (1969c). *Vehicle Design Criteria - Spacecraft Magnetic Torques* (tech. rep.). http://ww w.dept.aoe.vt.edu/$%5Csim$cdhall/courses/aoe4065/NASADesignSPs/sp8018 .pdf. (Cit. on pp. 36, 37)

[33] Thébault, E., Finlay, C. C., Beggan, C. D., Alken, P., Aubert, J., Barrois, O., Bertrand, F., Bondar, T., Boness, A., Brocco, L., Canet, E., Chambodut, A., Chulliat, A., Coïsson, P., Civet, F., Du, A., Fournier, A., Fratter, I., Gillet, N., ... Zvereva, T. (2015). International Geomagnetic Reference Field: the 12th generation. *Earth, Planets and Space.* https://doi.org/10.1186/s40623-015-0228-9 (cit. on p. 37)

[34] NASA. (1969d). *Vehicle Design Criteria - Spacecraft Radiation Torques* (tech. rep.). http://www.dept.aoe.vt.edu/$%5Csim$cdhall/courses/aoe4065/NASADesignSPs/sp8027.pdf. (Cit. on p. 38)

[35] Kubo-Oka, T., & Sengoku, A. (1999). *Solar radiation pressure model for the relay satellite of SELENE* (tech. rep.). (Cit. on p. 39).

[36] Brown, C. D. (2002). *Elements of spacecraft design.* American Institute of Aeronautics; Astronautics. (Cit. on p. 40).

[37] Griffin, M. D. (2004). *Space vehicle design.* AIAA. (Cit. on pp. 43, 133).

[38] Oland, Espen and Schlanbusch, Rune. (2009). Reaction wheel design for cubesats. *2009 4th International Conference on Recent Advances in Space Technologies*, 778–783. https://doi.org/10.1109/RAST.2009.5158296 (cit. on p. 44)

[39] Simplify3D. (2020). Ultimate 3D Printing Material Properties Table. Retrieved May 2, 2021, from https://www.simplify3d.com/support/materials-guide/properties-table/. (Cit. on p. 46)

[40] Royal Society of Chemistry. (2014). Lead - Element information, properties and uses — Periodic Table. Retrieved May 2, 2021, from http://www.rsc.org/periodic-table/element/82/lead. (Cit. on p. 46)

[41] AZoM. (2021). Properties: An Introduction to Iron. Retrieved May 2, 2021, from https://www.azom.com/properties.aspx?ArticleID=619. (Cit. on p. 46)

[42] International Genetically Engineered Machine. (2021). *Comparison of typical 3D printing materials [1]* (tech. rep.). (Cit. on p. 46).

[43] Kamthai, S., & Magaraphan, R. (2015). Thermal and mechanical properties of polylactic acid (PLA) and bagasse carboxymethyl cellulose (CMCB) composite by adding isosorbide diesters. *1664*, 20038. https://doi.org/10.1063/1.4918424 (cit. on p. 46)

[44] Plastim. (2021). ABS Product Data Sheet. Retrieved May 2, 2021, from https://plastim.co.uk/wp-content/uploads/2019/07/ABS-Technical-Data-Sheet.pdf. (Cit. on p. 46)

[45] Polymaker. (2021). PLA : Technical Data Sheet. Retrieved May 2, 2021, from http://wiki.germanreprap.com/_media/info/pla.pdf. (Cit. on p. 46)

[46] Agenda, I. (2021). *Microcontroller (MCU).* Retrieved February 16, 2021, from https://internetofthingsagenda.techtarget.com/definition/microcontroller. (Cit. on pp. 49, 50)

[47] Floyd, T. L., & Caño, J. G. (1997). *Fundamentos de sistemas digitales.* Prentice Hall. (Cit. on pp. 49, 50).

[48] Components 101. (2021a). *Difference between Microprocessor and Microcontroller.* Retrieved February 16, 2021, from https://components101.com/articles/difference-between-microprocessor-and-microcontroller. (Cit. on p. 51)

[49] Arduino. (2021). *Arduino Nano ATmega328*. Retrieved February 16, 2021, from `https://www.iberobotics.com/producto/arduino-nano-v3-0-atmega328-5v-16mhz-compatible/`. (Cit. on pp. 51, 52)

[50] Raspberry Pi. (2021). *Raspberry Pi 4*. Retrieved February 16, 2021, from `https://www.raspberrypi.org/products/raspberry-pi-4-model-b/`. (Cit. on p. 51)

[51] STM32. (2021). STM32F103C8T6 - Blue Pill — STM32-base project. Retrieved June 14, 2021, from `https://stm32-base.org/boards/STM32F103C8T6-Blue-Pill.html`. (Cit. on pp. 51, 52, 147)

[52] Components101. (2021b). MPU9250 9-DOF MEMS Sensor Module. Retrieved June 11, 2021, from `https://components101.com/sensors/MPU9250-9-dof-mems-sensor-module-datasheet-pinout-features-working`. (Cit. on pp. 53, 151, 152)

[53] Adafruit. (2021a). *Adafruit Unified BNO055 Driver (AHRS/Orientation)*. Retrieved February 17, 2021, from `https://github.com/adafruit/Adafruit_BNO055`. (Cit. on pp. 53, 222, 224)

[54] Components101. (2018a). Arduino Nano Pin Diagram, Features, Pin Uses & Programming. Retrieved April 25, 2021, from `https://components101.com/microcontrollers/arduino-nano`. (Cit. on p. 54)

[55] Core Electronics. (2019). Motor Drivers vs. Motor Controllers - Tutorial Australia. Retrieved April 25, 2021, from `https://core-electronics.com.au/tutorials/motor-drivers-vs-motor-controllers.html`. (Cit. on pp. 54, 55, 56)

[56] Sparkfun. (2019a). Hookup Guide for the Qwiic Motor Driver - learn.sparkfun.com. Retrieved April 26, 2021, from `https://learn.sparkfun.com/tutorials/hookup-guide-for-the-qwiic-motor-driver?_ga=2.144662197.1814996292.1615051654-700532284.1614101358`. (Cit. on pp. 56, 174, 175, 177, 178, 179, 180)

[57] Sparkfun. (2019b). SparkFun Qwiic Motor Driver - ROB-15451. Retrieved April 26, 2021, from `https://www.sparkfun.com/products/15451`. (Cit. on p. 57)

[58] Mankar, J., Darode, C., Trivedi, K., Kanoje, M., & Shahare, P. (2014). REVIEW OF I2C PROTOCOL. *International Journal of Research in Advent Technology*, *2*(1). `http://www.ijrat.org` (cit. on p. 57)

[59] motor, M. (2021). RF-500TB datasheet. Retrieved June 13, 2021, from `https://www.estudioelectronica.com/wp-content/uploads/2018/09/SHT-034.pdf`. (Cit. on pp. 59, 60, 167)

[60] Mabuchi Motor. (2021). RF-300EA datasheet. Retrieved June 13, 2021, from `https://datasheetspdf.com/pdf-file/917203/MABUCHIMOTOR/RF-300EA-1D390/1`. (Cit. on pp. 59, 60, 167)

[61] Naylamp Mechatronics SAC. (2021). Conversor de nivel lógico I2C 5V a 3.3V. Retrieved June 10, 2021, from `https://naylampmechatronics.com/conversores-ttl/67-conversor-de-nivel-logico-i2c.html`. (Cit. on pp. 60, 61)

[62] C. Bisdikian. (2001). An overview of the bluetooth wireless technology. *IEEE Communications Magazine*, *39*(12), 86–94. `https://doi.org/10.1109/35.968817` (cit. on p. 67)

[63] Dr. Ghanshyam Singh. (2015). Spread spectrum technologies. Retrieved April 2, 2021, from `https://www.slideshare.net/gkdelhi8/spread-spectrum-technologies`. (Cit. on p. 67)

[64] Pérez, Adrià. (2021). Project of attitude control system based on reaction wheels. UPC. (Cit. on pp. 82, 86).

[65] Bablon, Olivier. (2021a). Project of an attitude control system for nanosatellites. UPC. (Cit. on p. 82).

[66] Ogata, K. (2010). *Modern control engineering*. Prentice hall. (Cit. on pp. 91, 92).

[67] Bablon, Olivier. (2021b). Estudio, Diseño y construcción de una plataforma pivotante para ensayo del sistema de control de actitud (ACS) de un cubesat. UPC. (Cit. on p. 97).

[68] Idescat. (2021). Idescat. Anuario estadístico de Cataluña. Salario bruto anual y ganancia por hora. Por sexo y tipo de empleo. Retrieved June 19, 2021, from https://www.idescat.cat/pub/?id=aec&n=398&lang=es. (Cit. on p. 122)

[69] Belgian Air and Space Policy. (2021). The Belgian Space Law. Retrieved March 16, 2021, from https://www.belspo.be/belspo/space/beLaw_en.stm. (Cit. on p. 133)

[70] Federal Communications Commission. (2021). Guidance On Obtaining Licenses For Small Satellites — Federal Communications Commission. Retrieved March 16, 2021, from https://www.fcc.gov/document/guidance-obtaining-licenses-small-satellites. (Cit. on p. 133)

[71] International Amateur Radio Union. (2020). Satellites — IARU. Retrieved March 16, 2021, from https://www.iaru.org/on-the-air/satellites/. (Cit. on p. 133)

[72] National Oceanic and Atmospheric Administration. (2017). NOAA CRSRA Application Process. Retrieved March 16, 2021, from https://www.nesdis.noaa.gov/CRSRA/generalApplication.html. (Cit. on p. 133)

[73] NASA. (2019). *U.S. Government Orbital Debris Mitigation Standard Practices* (tech. rep.). https://www.nasa.gov/sites/default/files/atoms/files/usg_orbital_debris_mitigation_standard_practices_november_2019.pdf. (Cit. on p. 134)

[74] United Nations Office for Outer Space Affairs. (2021). Space Law: National Space Law. Retrieved March 16, 2021, from https://www.unoosa.org/oosa/en/ourwork/spacelaw/nationalspacelaw/index.html. (Cit. on p. 134)

[75] European Cooperation for Space Standardization. (2017). ECSS-E-ST-10C Rev.1 – System engineering general requirements (15 February 2017). Retrieved May 31, 2021, from https://ecss.nl/standard/ecss-e-st-10c-rev-1-system-engineering-general-requirements-15-february-2017/. (Cit. on p. 134)

[76] NASA. (2020). NASA Systems Engineering Handbook Revision 2. Retrieved May 31, 2021, from https://www.nasa.gov/connect/ebooks/nasa-systems-engineering-handbook. (Cit. on p. 134)

[77] SEBoK. (2021). Guide to the Systems Engineering Body of Knowledge (SEBoK). Retrieved May 31, 2021, from https://www.sebokwiki.org/wiki/Guide_to_the_Systems_Engineering_Body_of_Knowledge_(SEBoK). (Cit. on p. 134)

[78] ISO. (2015). ISO - ISO/IEC/IEEE 15288:2015 - Systems and software engineering — System life cycle processes. Retrieved May 31, 2021, from https://www.iso.org/standard/63711.html. (Cit. on p. 134)

[79] Marboe, I., & United Nations Office for Outer Space Affairs. (2016). *International Law Perspectives on Small Satellites Activities* (tech. rep.). (Cit. on p. 134).

[80] CubeSat Launch Initiative, N. (2017). *CubeSat 101: Basic Concepts and Processes for First-Time CubeSat Developers* (tech. rep.). (Cit. on p. 134).

[81] Marboe, I. (2016). *Small satellites: Regulatory challenges and chances.* Brill. (Cit. on p. 134).

[82] Air Force Space Command. (2021). *Range Safety User Requirements Manual Volume 3.* Retrieved February 8, 2021, from https://static.e-publishing.af.mil/production/1/afspc/publication/afspcman91-710v3/afspcman91-710v3.pdf. (Cit. on p. 135)

[83] NASA. (2021). *Outgassing Data for Selecting Spacecraft Materials Online.* Retrieved February 8, 2021, from https://outgassing.nasa.gov/. (Cit. on p. 135)

[84] ROBU.IN. (2020). *Arduino Nano Board: Pinout & Programming.* Retrieved February 17, 2021, from https://robu.in/arduino-nano-board-pinout-programming-guide-2020/. (Cit. on pp. 145, 148)

[85] Freeelectron. (2020). Installing ST-Link v2 to flash STM32 targets on Linux — Free Electron. Retrieved June 24, 2021, from https://freeelectron.ro/installing-st-link-v2-to-flash-stm32-targets-on-linux/. (Cit. on p. 150)

[86] Reginald Watson. (2019). How to Program the STM32 "Blue Pill" with Arduino IDE — Arduino — Maker Pro. Retrieved April 5, 2021, from https://maker.pro/arduino/tutorial/how-to-program-the-stm32-blue-pill-with-arduino-ide. (Cit. on p. 150)

[87] Adafruit. (2021b). *Adafruit Unified Sensor Driver.* Retrieved February 17, 2021, from https://github.com/adafruit/Adafruit_Sensor. (Cit. on pp. 153, 154, 224)

[88] Bosch. (2014). *BNO055 Intelligent 9-axis absolute orientation sensor* (tech. rep.). (Cit. on p. 153).

[89] Luis Llamas. (2018). Adaptar 3.3V a 5V (y viceversa) en Arduino con Level Shifter. Retrieved June 12, 2021, from https://www.luisllamas.es/arduino-level-shifter/. (Cit. on p. 155)

[90] Components101. (2018b). HC06 Bluetooth module pinout, features & datasheet. Retrieved June 10, 2021, from https://components101.com/wireless/hc-06-bluetooth-module-pinout-datasheet. (Cit. on pp. 157, 158)

[91] AranaCorp. (2018). Arduino and Bluetooth module HC-05. Retrieved June 10, 2021, from https://www.aranacorp.com/en/arduino-and-bluetooth-module-hc-05/. (Cit. on p. 158)

[92] Mabuchi Motors. (2021). Motor Designations and Their Meanings — Products Information — MABUCHI MOTOR CO., LTD. Retrieved June 14, 2021, from https://www.mabuchi-motor.com/product/knowledge/classification/designations.html. (Cit. on pp. 167, 168, 169, 170)

[93] Power Stream. (2016). American Wire Gauge table and AWG Electrical Current Load Limits with skin depth frequencies and wire breaking strength. Retrieved April 28, 2021, from http://www.powerstream.com/Wire_Size.htm. (Cit. on p. 176)

[94] Sparkfun. (2019c). Qwiic Connect System - SparkFun Electronics. Retrieved April 28, 2021, from https://www.sparkfun.com/qwiic. (Cit. on p. 176)

[95] Sparkfun. (2019d). Qwiic Cable - Breadboard Jumper (4-pin). Retrieved April 28, 2021, from https://www.sparkfun.com/products/14425. (Cit. on pp. 177, 180)

[96] Morales, Jordan. (2021). Desarrollo de una placa ADCS para un prototipo de CubeSat con pares magnéticos y ruedas de inercia. UPC. (Cit. on p. 191).

[97] Reurer, Miquel. (2021). Realización de una placa de control de actitud 2D basada en pares magnnéticos para un prototipo de CubeSat. UPC. (Cit. on pp. 193, 195).

[98] Mahooti, M. (2018). Jacchia-Bowman Atmospheric Density Model - MATLAB Central. Retrieved May 16, 2021, from https://www.mathworks.com/matlabcentral/fileexchange/56163-jacchia-bowman-atmospheric-density-model?s_tid=mwa_osa_a. (Cit. on p. 214)

[99] Adafruit. (2021c). *Adafruit Unified BNO055*. Retrieved February 18, 2021, from https://learn.adafruit.com/adafruit-bno055-absolute-orientation-sensor/arduino-code. (Cit. on p. 225)

[100] Pieter-Jan Van de Maele. (2021). *Reading a IMU Without Kalman: The Complementary Filter*. Retrieved February 20, 2021, from https://www.pieter-jan.com/node/11. (Cit. on p. 246)

Appendices

Appendix A

Requirements

A.1   Technical requirements

In this section, we aim to provide strict and measurable statements that will guide and limit the process, to minimise weak spots and vague actions that could lead to misunderstandings:

One of the most important limitations comes from the maximum weight permitted for a 1U Cubesat. The Cubesat standard specification [2] dictates the maximum dimensions and weights for a 1U Cubesat, which cannot exceed a 10×10 cm side cube with a maximum weight of $1,3$ kg. Thereby, the current Cubesat is categorized as a nanosatellite since its weight is higher than 1 kg and less than 10 kg.

According to Dr. Griffin and Dr. French [37], the ADCS subsystem weight is around $10 - 15\%$ of the total weight of the satellite. This means the upper boundary limitation is around 200 g out of the whole satellite's weight.

An additional requirement shall be to reduce the response time of the reaction wheel once a duty is set. The final aim is to pursue the optimal response time and performance while preserving energy and accuracy.

The Cubesat must meet The Belgian Law on the Activities of Launching, Flight Operation or Guidance of Space Objects 2005 regulation Belgian Air and Space Policy [69].

The CubeSat must also comply with technical constraints as specified by FCC regulations Federal Communications Commission [70].

The experimental-licensed Cubesat spacecraft that use frequency bands allocated to the Amateur-Satellite Service must coordinate with the International Amateur Radio Union (IARU) [71].

If the Cubesat includes an imager or a camera, for non-Government owned Cubesat with an imager must contact the National Oceanic and Atmospheric Administration (NOAA) for a remote sensing license if needed [72].

The Cubesat design must always be designed following the United States Government (USG) Orbital Debris Mitigation Standard Practices (ODMSP) to address the increase in orbital debris in the near-Earth space environment. The goal is to limit the generation of new, long-lived debris by the control of debris released during normal operations, minimizing debris generated by accidental explosions, the selection of safe flight profile and operational configuration to minimize accidental collisions, and post-mission disposal of space structure from NASA [73].

Small satellites should also fall under the scope of national space legislation United Nations Office for Outer Space Affairs (UNOOSA) [74].

Finally, all of the aforementioned standards have a lot in common because their purpose is the same: to propose an organized system for getting things done correctly. If the project is space-related and takes place in Europe, the ECSS standard should be followed. Nonetheless, if it is located in the United States, it must adhere to the NASA standard. To consider which standard to use depends on many factors: where is the mission located, what is its purpose and whether it is part of a larger project.

The following are some formal document-based and model-based systems engineering standards:

- European Cooperation for Space Standardization (ECSS): ECSS-E-ST-10C [75]
- The National Aeronautics and Space Administration (NASA): Systems Engineering Handbook [76].
- International Council on Systems Engineering (INCOSE): Guide to the Systems Engineering Body of Knowledge (SEBoK) [77].
- ISO/IEC/IEEE 15288 : Systems and software engineering [78].

No exact requirements for attitude control or response time are set. Nonetheless, the aim will always be to increase the performance and response time ensuring fast and reliable maneuvers while preserving the Cubesat Standard constraints [3] [79] [80] [81].

A.2 Structure Subsystem

Cubesat specifications can be found in California's University's specification [2]. The CubeSat Design Specification document presents a deep overview of all general, mechanical, electrical and operational requirements.

Below, general and mechanical requirements are listed and explained:

A.2.1 General requirements of the Cubesat

1. CubeSats which incorporate any deviation from the CDS will submit a DAR and adhere to the waiver process.

   This means that any changes to the standardized specifications shall be notified, that is, the mission shall fill out a form and send it to the launch provider.

2. All parts shall remain attached to the CubeSats during launch, ejection and operation. No additional space debris will be created.

   To ensure this requirement, the structure of the Cubesat must ensure to be able to support the forces it is subjected.

3. No pyrotechnics shall be permitted.

4. Any propulsion systems shall be designed, integrated, and tested in accordance with AFSPCMAN 91-710 Volume 3 [82].

5. Propulsion systems shall have at least 3 inhibits to activation.

   This translates that 3 different mechanisms or buttons must be engaged to inhibit activation.

6. Total stored chemical energy will not exceed 100 Watt-Hours.

   6.1. Note: Higher capacities may be permitted, but could potentially limit launch opportunities.

7. CubeSat hazardous materials shall conform to AFSPCMAN 91-710, Volume 3 [82].

8. CubeSat materials shall satisfy the following low out-gassing criterion to prevent contamination of other spacecraft during integration, testing, and launch. A list of NASA approved low out-gassing materials can be found at: http://outgassing.nasa.gov.

   8.1. Note: Higher capacities may be permitted, but could potentially limit launch opportunities.

   Some materials, most common plastics, are designed to operate a certain level of atmospheric pressure. When exposed to outer space conditions, i.e. vacuum and zero pressure levels, the material can release a cloud of mist or condensate which can end up damaging other electronic components including solar panels. For instance, outgassing can contribute to the degraded performance of charge-coupled-device (CCD) sensors of some missions. NASA maintains a list of materials with low-outgassing properties suitable for use in spacecraft [83].

9. The latest revision of the CubeSat Design Specification will be the official version to which all CubeSat developers will adhere to. The latest revision is available at http://www.cubesat.org.

10. The CubeSat shall be designed to accommodate ascent venting per ventable volume/area < 2000 inches

A.2.2 Mechanical requirements of the Cubesat

CubeSats are cube-shaped nanosatellites with dimensions and features outlined in the CubeSat Specification Drawing (see Figure A.1) for reference.
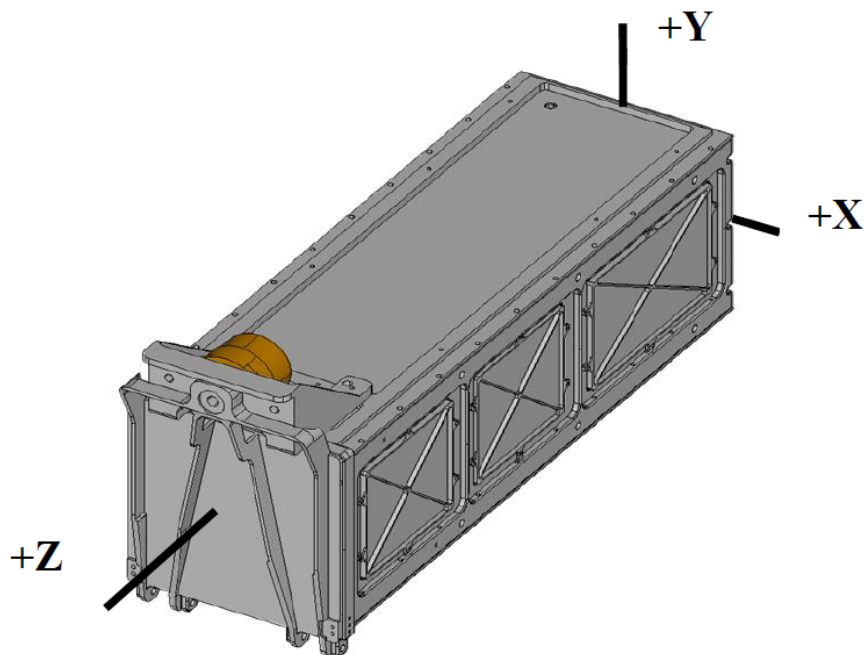
Figure A.1 P-POD Cubesat coordinate system. Source: California Polytechnic State University [2].

General features of all CubeSats include:

1. The CubeSat shall use the coordinate system as defined in Appendix B of [2] (see Figure A.2) for the appropriate size. The CubeSat coordinate system will match the P-POD coordinate system while integrated into the P-POD. The origin of the CubeSat coordinate system is located at the geometric centre of the CubeSat.

   1.1. The CubeSat configuration and physical dimensions shall be per the appropriate section of Appendix B [2].

   1.2. The extra volume available for 3U CubeSats is shown in Figure 6 of Appendix B [2].

2. The Z face of the CubeSat will be inserted first into the P-POD.

3. No components on the green and yellow shaded sides shall exceed 6.5 mm normal to the surface.

   3.1. When completing a CubeSat Acceptance Checklist (CAC), protrusions will be measured from the plane of the rails.

4. Deployables shall be constrained by the CubeSat, not the P-POD.

5. Rails shall have a minimum width of 8.5 mm.

6. Rails will have a surface roughness less than 1.6 μm.

7. The edges of the rails will be rounded to a radius of at least 1 mm.

8. The ends of the rails on the $+/-Z$ face shall have a minimum surface area of 6.5 mm $\times$ 6.5 mm contact area for neighbouring CubeSat rails (as per Figure 6).

9. At least 75% of the rail will be in contact with the P-POD rails. 25% of the rails may be recessed and no part of the rails will exceed the specification.

10. The maximum mass of a 1U CubeSat shall be 1.33 kg.

    10.1. Note: Larger masses may be evaluated on a mission to mission basis.

11. The maximum mass of a 1.5U CubeSat shall be 2.00 kg.

    11.1. Note: Larger masses may be evaluated on a mission to mission basis.

12. The maximum mass of a 2U CubeSat shall be 2.66 kg.

    12.1. Note: Larger masses may be evaluated on a mission to mission basis.

13. The maximum mass of a 3U CubeSat shall be 4.00 kg.

    13.1. Note: Larger masses may be evaluated on a mission to mission basis.

14. The CubeSat center of gravity shall be located within 2 cm from its geometric center in the X and Y direction.

    14.1. The 1U CubeSat center of gravity shall be located within 2 cm from its geometric center in the Z direction.

    14.2. The 1.5U CubeSat center of gravity shall be located within 3 cm from its geometric center in the Z direction.

    14.3. The 2U CubeSat center of gravity shall be located within 4.5 cm from its geometric center in the Z direction.

    14.4. 3U and 3U CubeSats' center of gravity shall be located within 7 cm from its geometric center in the Z direction.

15. Aluminum 7075, 6061, 5005, and/or 5052 will be used for both the main CubeSat structure and the rails.

    15.1. If other materials are used the developer will submit a DAR and adhere to the waiver process.

    These materials were selected considering their weight and their ability to support structural loads.

16. The CubeSat rails and standoff, which contact the P-POD rails and adjacent CubeSat standoffs, shall be hard anodized aluminium to prevent any cold welding within the P-POD.

17. The 1U, 1.5U and 2U CubeSats shall use separation springs to ensure adequate separation (see Table A.1 and Figure A.3).

    17.1. The 1U, 1.5U and 2U CubeSats shall use separation springs to ensure adequate separation.

    17.2. The compressed separation springs shall be at or below the level of the standoff.

    17.3. The 1U, 1.5U and 2U CubeSats separation spring will be centred on the end of the standoff on the CubeSat's $-Z$ face as per Figure 7 [2].

    17.4. Separation springs are not required for 3U CubeSats.
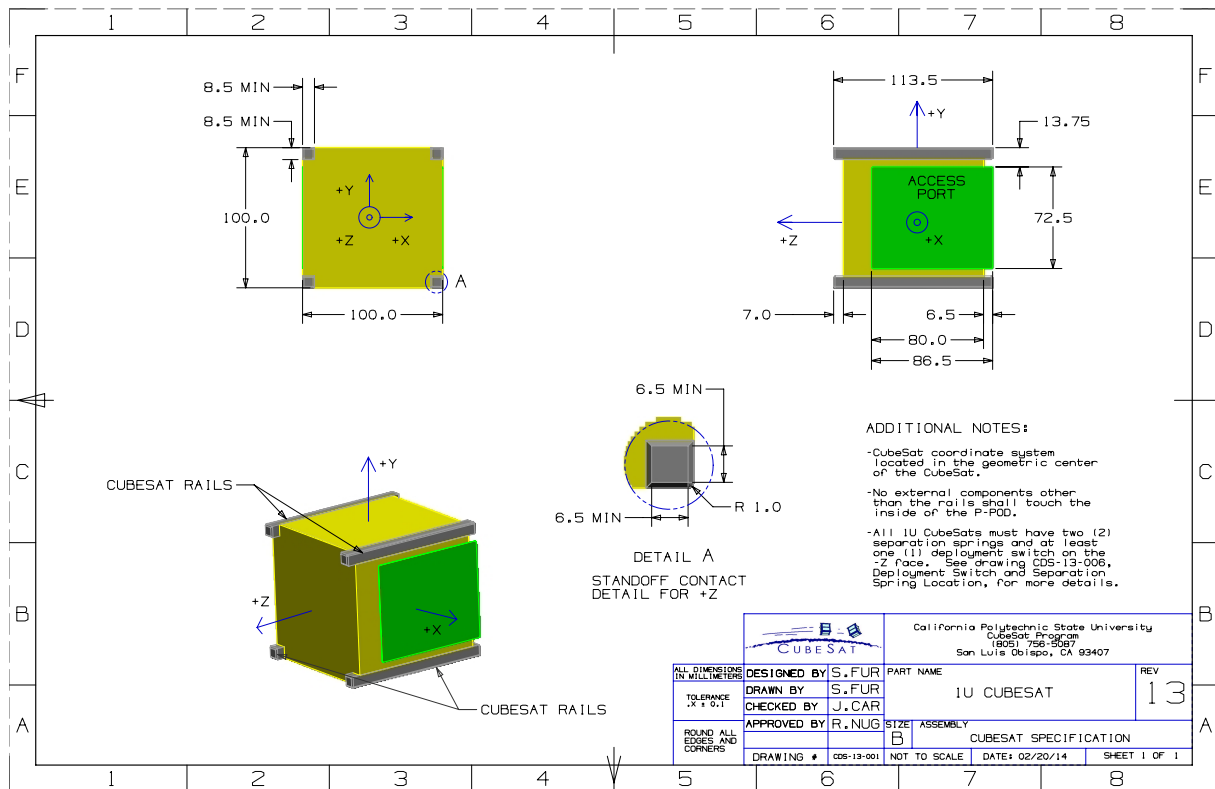
Figure A.2 1U CubeSat Design Specification Drawing. Source: California Polytechnic State University [2].

Characteristics	Value
Plunger Material	Stainless Steel
End Force Initial/Final	0.14 lbs / 0.9 lbs
Throw Length	0.16 in minimum above the standoff surface
Thread Pitch	8-36 UNF-2B

Table A.1 CubeSat Separation Spring Characteristics. Source: California Polytechnic State University [2].
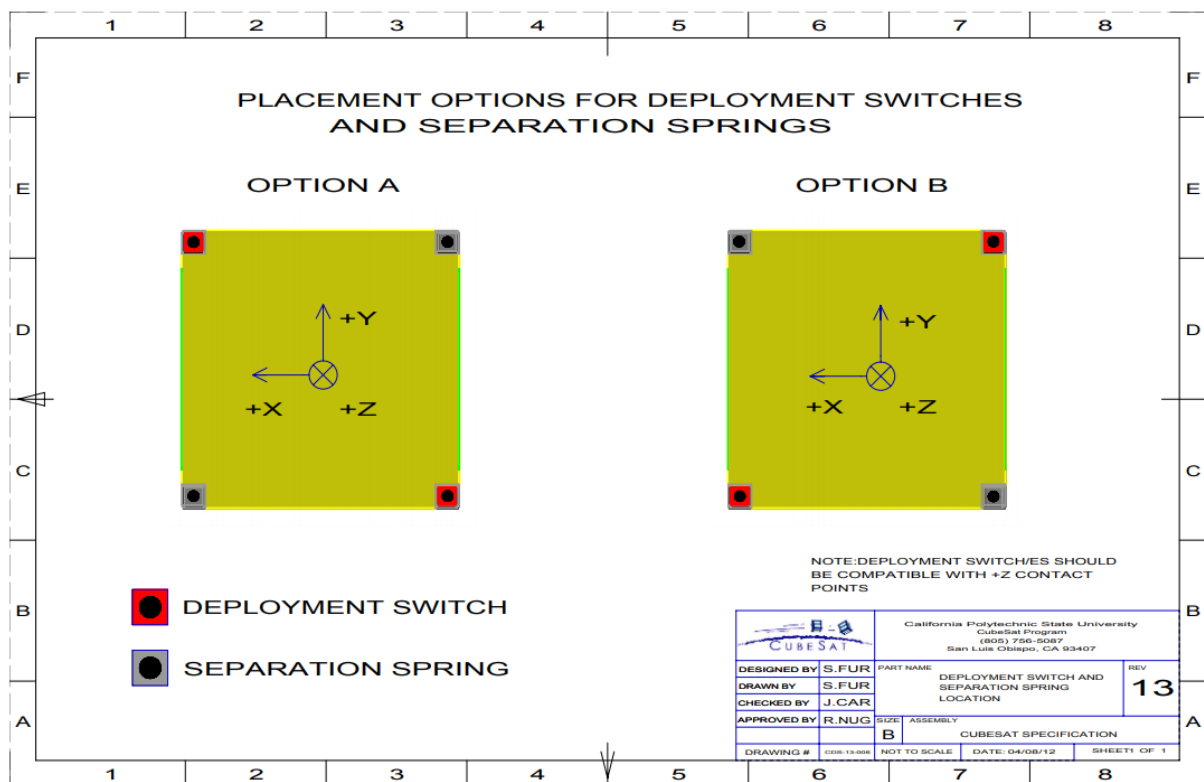
Figure A.3 Deployment switches and separation spring locations. Source: California Polytechnic State University [2].

A.2.3 Electrical Requirements

Electronic systems will be designed with the following safety features.

1. The CubeSat power system shall be at a power-off state to prevent CubeSat from activating any powered functions while integrated with the P-POD from the time of delivery to the Launch Vehicle (LV) through on-orbit deployment. CubeSat powered function includes a variety of subsystems such as Command and Data Handling (C&DH), RF Communication, Attitude Determine and Control (ADC), deployable mechanism actuation. CubeSat power systems include all battery assemblies, solar cells, and coin cell batteries.

2. The CubeSat shall have, at a minimum, one deployment switch on a rail standoff, per Figure A.3.

3. In the actuated state, the CubeSat deployment switch shall electrically disconnect the power system from the powered functions; this includes real-time clocks (RTC).

4. The deployment switch shall be in the actuated state at all times while integrated in the P-POD.

    4.1. In the actuated state, the CubeSat deployment switch will be at or below the level of the standoff.

5. If the CubeSat deployment switch toggles from the actuated state and back, the transmission and deployable timers shall reset to $t = 0$.

6. The Remove Before Flight (RBF) pin and all CubeSat umbilical connectors shall be within the designated Access Port locations, green shaded areas shown in Appendix B [2].

   6.1. Note: All diagnostics and battery charging within the P-POD will be done while the deployment switch is depressed.

7. The CubeSat shall include an RBF pin.

   7.1. The RBF pin shall cut all power to the satellite once it is inserted into the satellite.

   7.2. The RBF pin shall be removed from the CubeSat after integration into the P-POD.

   7.3. The RBF pin shall protrude no more than 6.5 mm from the rails when it is fully inserted into the satellite.

8. CubeSats shall incorporate battery circuit protection for charging/discharging to avoid unbalanced cell conditions.

9. The CubeSat shall be designed to meet at least one of the following requirements to prohibit inadvertent radio frequency (RF) transmission. The use of three independent inhibits is highly recommended and can reduce required documentation and analysis. An inhibit is a physical device between a power source and a hazard. A timer is not considered an independent inhibit.

   9.1. The CubeSat will have one RF inhibit and RF power output of no greater than 1.5 W at the transmitting antenna's RF input.

   9.2. The CubeSat will have two independent RF inhibits

A.3   Operational Requirements

CubeSats will meet certain requirements pertaining to integration and operation to meet legal obligations and ensure the safety of other CubeSats [2].

1. Operators will obtain and provide documentation of proper licenses for use of radio frequencies.

   1.1. For amateur frequency use, this requires proof of frequency coordination by the International Amateur Radio Union (IARU). Applications can be found at www.iaru.org.

2. CubeSats will comply with their country's radio license agreements and restrictions.

3. CubeSats mission design and hardware shall be in accordance with NPR 8715.6 to limit orbital debris.

   3.1. Any CubeSat component shall re-enter with energy less than 15 J.

   3.2. Developers will obtain and provide documentation of approval of an orbital debris mitigation plan from the Federal Communication Commission (FCC) (or NOAA if imager is present).

      3.2.1. Note: To view FCC amateur radio regulations, go to http://www.arrl.org/part-97-amateur-radio

      3.2.2. Note: Analysis can be conducted to satisfy the above with NASA DAS, available at http://orbitaldebris.jsc.nasa.gov/mitigate/das.html

4. All deployable such as booms, antennas, and solar panels shall wait to deploy a minimum of 30 minutes after the CubeSat's deployment switch(es) are activated from P-POD ejection.

5. No CubeSats shall generate or transmit any signal from the time of integration into the P-POD through 45 minutes after on-orbit deployment from the P-POD. However, the CubeSat can be powered on the following deployment from the P-POD.

6. Private entities (non-U.S. Government) under the jurisdiction or control of the United States who propose to operate a remote sensing space system (satellite) may need to have a license as required by U.S. law. For more information visit http://www.nesdis.noaa.gov/CRSRA/licenseHome.html. Click on the Application Process link under the Applying for a License drop-down section to begin the process.

7. Cal Poly will conduct a minimum of one fit check-in in which developer hardware will be inspected and integrated into the P-POD or TestPOD. A final fit check will be conducted before launch. The CubeSat Acceptance Checklist (CAC) will be used to verify compliance with the specification (Found in the appendix of this document or online at http://cubesat.org/index.php/documents/developers).

A.4 Testing requirements

Testing will be performed to meet all launch provider requirements as well as any additional testing requirements deemed necessary to ensure the safety of the CubeSats, P-POD, and the primary mission. The General Environmental Verification Standard (GEVS, GSFC-STD-7000) and MIL-STD-1540 can be used to derive testing requirements[2].

The general testings are listed below:

1. **Random Vibration**:

   Random vibration testing shall be performed as defined by the launch provider

2. **Thermal Vacuum Bakeout**:

   Thermal vacuum bakeout shall be performed to ensure proper outgassing of components. The test specification will be outlined by the launch provider.

3. **Shock testing**:

   Shock testing shall be performed as defined by the launch provider.

4. **Visual inspection**:

   Visual inspection of the CubeSat and measurement of critical areas will be performed per the appropriate CAC (Appendix C) [2].

5. **Cubesat testing Philosophy**:

   The CubeSat shall be subjected to either a qualification or protoflight testing as defined in the CubeSat Testing Flow Diagram, shown in Figure A.4. The test levels and durations will be supplied by the launch provider or P-POD integrator.

5.1. **Qualification**:

Qualification testing is performed on an engineering unit hardware that is identical to the flight model CubeSat. Qualification levels will be determined by the launch vehicle provider or P-POD integrator. Both MIL-STD-1540 and LSP-REQ-317.01 are used as guides in determining testing levels. The flight model will then be tested to Acceptance levels in a TestPOD then integrated into the flight P-POD for a final acceptance/workmanship random vibration test. **Additional testing may be required if modifications or changes are made to the CubeSats after qualification testing**.

5.2. **Protoflight**:

Protoflight testing is performed on the flight model CubeSat. Protoflight levels will be determined by the launch vehicle provider or P-POD integrator. Both MIL-STD-1540 and LSPREQ-317.01 are used as guides in determining testing levels. The flight model will be tested to Protoflight levels in a TestPOD then integrated into the flight P-POD for a final acceptance/workmanship random vibration test. The flight CubeSat SHALL NOT be disassembled or modified after protoflight testing. Disassembly of hardware after protoflight testing will require the developer to submit a DAR and adhere to the waiver process prior to disassembly. **Additional testing will be required if modifications or changes are made to the CubeSats after protoflight testing**.

5.3. **Acceptance**:

After delivery and integration of the CubeSat into the P-POD, additional testing will be performed with the integrated system. This test ensures proper integration of the CubeSat into the P-POD. Additionally, any unknown, harmful interactions between CubeSats may be discovered during acceptance testing. The P-POD Integrator will coordinate and perform acceptance testing. Acceptance levels will be determined by the launch vehicle provider or P-POD integrator. Both MIL-STD-1540 and LSP-REQ-317.01 are used as guides in determining testing levels. The PPOD **SHALL NOT** be deintegrated at this point. If a CubeSat failure is discovered, a decision to disintegrate the P-POD will be made by the developers, in that P-POD, and the P-POD Integrator based on safety concerns. The developer is responsible for any additional testing required due to corrective modifications to deintegrated P-PODs and CubeSats.
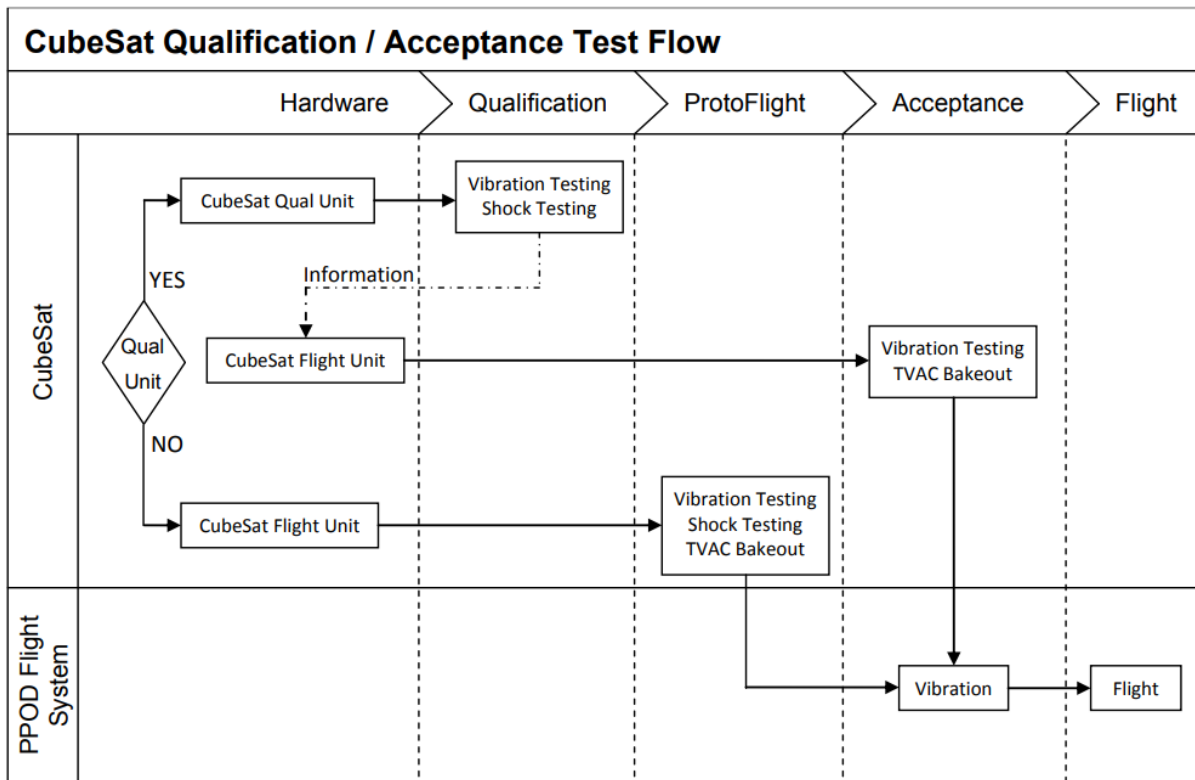
Figure A.4 CubeSat General Testing Flow Diagram. Source: California
Polytechnic State University [2].

Appendix B

Microcontroller

The following section provides the pin-outs of the microcontrollers discussed in the project.

B.1   Arduino Nano

The Arduino board is built in such a manner that newcomers may easily get started with microcontrollers. This board, in particular, is breadboard friendly, making it very simple to manage the connections. Let us begin by energizing the Board.

To power the Arduino Nano, there are three ways:

- USB Jack: Connect the mini USB jack to a phone charger or computer through a cable and it will draw the power required for the board to function

- Vin Pin: The Vin pin can be supplied with an unregulated 6-12V to power the board. The on-board voltage regulator regulates it to +5V

- +5V Pin: If you have a regulated +5V supply then you can directly provide this with the +5V pin of the Arduino.

For the input and outputs pins:

Arduino Nano board has 14 digital pins and 8 analog pins in total. The digital pins can be used to interconnect sensors (as input pins) or to control loads (as output pins). To regulate their functionality, basic functions such as `pinMode()` and `digitalWrite()` can be used. For digital pins, the operational voltage is 0 V and 5 V. Analog voltages ranging from 0V to 5 V may be measured using any of the 8 Analog pins and a simple function called `analogRead()`

Also some pins have special purposes:

- **Serial Pins 0 (Rx) and 1 (Tx)**: Rx and Tx pins are used to receive and transmit TTL serial data. They are connected with the corresponding ATmega328P USB to TTL serial chip.

- **External Interrupt Pins 2 and 3**: These pins can be configured to trigger an interrupt on a low value, a rising or falling edge, or a change in value.

- **PWM Pins 3, 5, 6, 9 and 11**: These pins provide an 8-bit PWM output by using analogWrite() function.

- **SPI Pins 10 (SS), 11 (MOSI), 12 (MISO) and 13 (SCK)**: These pins are used for SPI communication.

- **In-built LED Pin 13**: This pin is connected with an built-in LED, when pin 13 is HIGH – LED is on and when pin 13 is LOW, its off.

- **I2C A4 (SDA) and A5 (SCA)**: Used for I2C communication using Wire library.

- **AREF**: Used to provide reference voltage for analog inputs with `analogReference()` function.

- **Reset Pin**: Making this pin LOW, resets the microcontroller.

These special functions and their respective pins are illustrated in the Arduino Nano pin diagram shown in Figure B.1.

Source: Fritzing
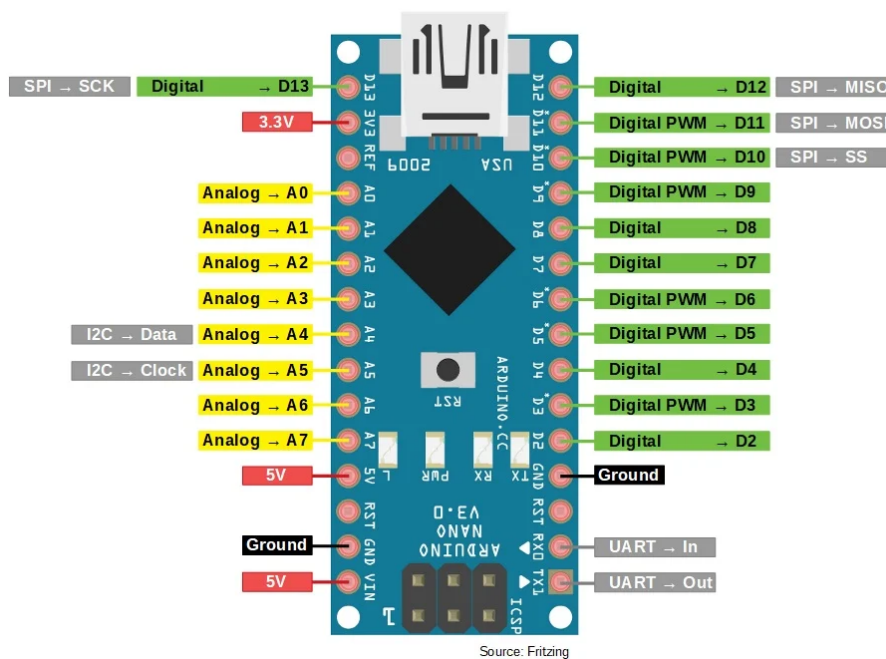
Figure B.1 Arduino Nano Pinout. Source: Robu.in Arduino [84]

USB

+5V REG

+5V AUTO SELECTOR

ICSP

B.2   STM32 Bluepill

STM- Microelectronics provides a diverse variety of general-purpose microcontrollers for both 8-bit MCU (STM8) and 32-bit ARM Cortex-M-based microcontrollers (STM32). STM32 Blue Pill is based on STM32F103C8T6, which contains a Cortex-M3 ARM CPU that operates at 72 MHz, 20 kB of RAM, and 64kB of flash memory. The microcontroller has one USB port (with no additional chip support), three UART ports, 16-bit PWM pins, and other features. It is a 3.3V microcontroller with several 5V tolerant pins [51].

To power the Bluepill, one can power it with several ways listed below:

- Using the built-in USB micro connector.
- Supplying 5 V to the 5 V pin as external supply.
- Supplying 3.3 V directly to the 3.3 V pin.

For the input and output pins: The Blue Pill features 37 GPIO pins distributed across four ports: A and B (16 pins), C (3 pins), and D (no pins) (2 pins). Each pin has a 6 mA current sink/source capability. On each of the pins, pull-up and pull-down resistors can be activated.

Most pins have extra functionality as well:

- Serial ports – receive and transmit data via the UART protocol
- I^2C ports – two-wire communication via the I^2C protocol
- SPI – serial communication
- PWM
- Pin 13 has a built-in LED

These special functions and their respective pins are illustrated in the Blue Pill pin diagram shown below:
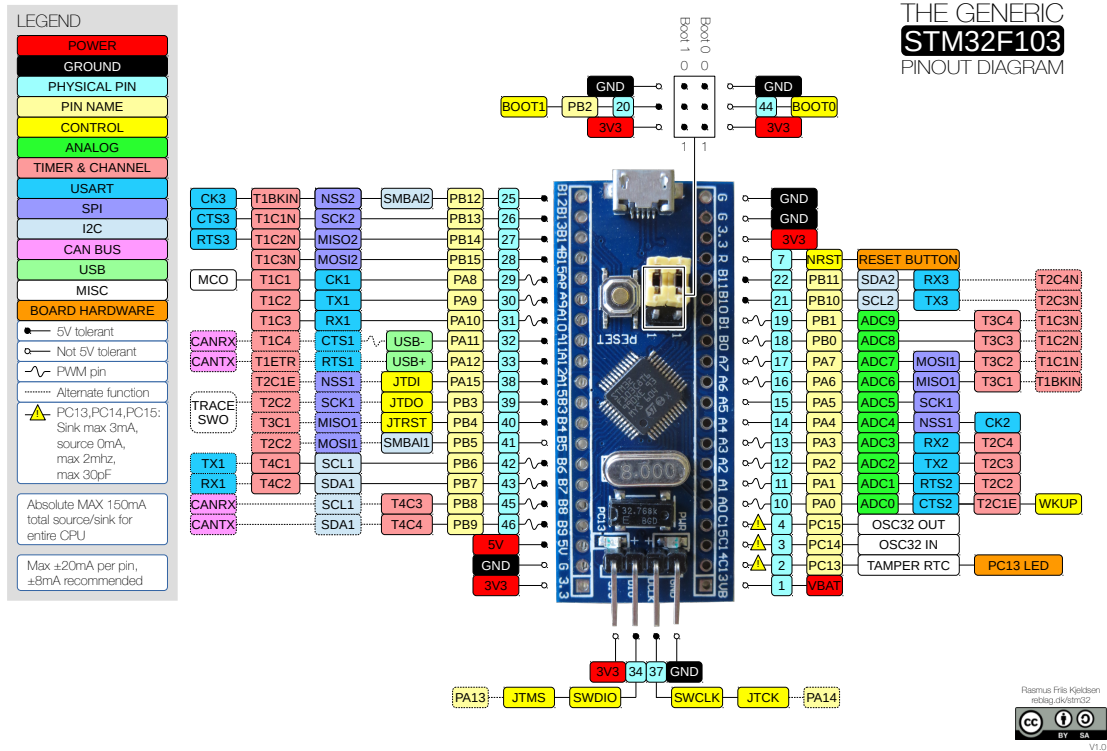
Figure B.2 Arduino Nano Pinout. Source: Robu.in Arduino [84]

复位电路

VCC3V3
R2
10K,0603
RESET
REST
C2
105,0603
GND

BOOTx设置

VCC3V3
P1
1 2
3 4
5 6
Header 3X2
BOOT0 R3 100K,0603
R4 BOOT1 100K,0603
GND

LED灯，D2可编程

VCC3V3 D1
LED
R1 510R,0603 GND
VCC3V3 D2
LED
R5 510R,0603 PC13

对外端子

VBAT PC13 PC14 PC15 PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 PB0 PB10 PB11 NRST VCC3V3 GND GND
P3
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
Header 20

PB12 PB13 PB14 PB15 PA8 PA9 PA10 PA11 PA12 PA15 PB3 PB4 PB5 PB6 PB7 PB8 PB9 5v GND VCC3V3
P4
20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1
Header 20

电源（5V-->3.3V）

5V
U1
1 VIN VOUT 5 VCC3v3
4 BP
3 EN
2 GND
RT9193-33
C5 105,0805 C7 C15 223,0603 C6 106,0805 C8 104,0603
C15 104,0603
GND GND GND

MCU

VCC3V3 GND PB9 PB8 BOOT0 PB7 PB6 PB5 PB4 PB3 PA15 SWCLK
48 47 46 45 44 43 42 41 40 39 38 37
VDD_3 VSS_3 PB9 PB8 BOOT0 PB7 PB6 PB5 PB4 PB3 PA15 SWCLK
U2
STM32F103C8T6

VBAT 1 VBAT   VDD_2 36 VCC3V3
PC13 2 PC13   VSS_2 35 GND
PC14 3 PC14   SWIO   34 SWIO
PC15 4 PC15   PA12   33 PA12
OSCIN 5 OSCIN   PA11   32 PA11
OSCOUT 6 OSCOUT   PA10   31 PA10
RESET 7 NRST   PA9   30 PA9
GND   8 VSSA   PA8   29 PA8
VCC3V3 9 VDDA   PB15   28 PB15
PA0   10 PA0   PB14   27 PB14
PA1   11 PA1   PB13   26 PB13
PA2   12 PA2   PB12   25 PB12

PA3 PA4 PA5 PA6 PA7 PB0 PB1 PB2/BOOT1 PB10 PB11 VSS_1 VDD_1
13 14 15 16 17 18 19 20 21 22 23 24
PA3 PA4 PA5 PA6 PA7 PB0 PB1 BOOT1 PB10 PB11 GND VCC3V3

USB电路

5V
U3
Vbus Vbus
D-   D-   R9 20R,0603
D+   D+   R11 20R,0603 PA11 PA12
ID   ID
G   G   R10 4.7k,0603
USB-mrico
GND 5V

J-Link OB接口

P2
1 VCC3V3
2 SWDIO
3 SWDCLK
4
SWD
C1 104,0603
SWD

VCC3V3 VCC3V3
C10 104,0603 C11 104,0603

VCC3V3 VCC3V3
C3 104,0603 C4 104,0603

C13 20p,0603 OSCIN
Y2 8MHz R6 1M
C14 20p,0603 OSCOUT

C9 20p,0603 PC14
Y3 32.768K
C12 20p,0603 PC15

源地工作室http://vcc-gnd.taobao.com/
论坛地址：http://www.vcc-gnd.com/

B.2.1 ST-Link

To upload and download code from the PC to the STM32 Bluepill an external device is needed called STLink.
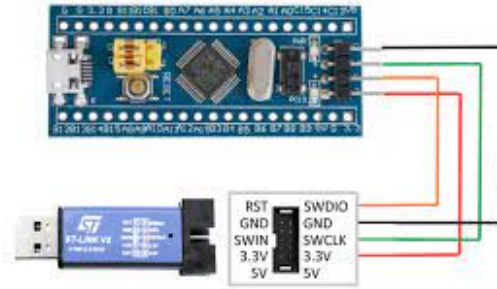
Figure B.3 STLink v2 pinout diagram. Source: Freeelectron [85].

Table B.1 ST-Link and STM32 pin connections. Source: Own.

ST-Link	STM32 Bluepill
GND	GND
SWCL	SCL
SWDIO	SWDIO
3,3 V	3.3V

B.2.2 Arduino IDE setup configuration for STM32

Having seen the advantages of the STM32 microcontroller explained in the precedent section. The next step encompasses the configuration of the Arduino IDE for developing on the STM32 microcontroller. The STM32 microcontroller is compatible with the Arduino IDE. However, some configurations need to be met in order to satisfy the compatibility.

The next paragraphs will summarize the STM32 configuration to meet the Arduino IDE requirements. The whole process can be found in [86].

- First, the Arduino IDE must be opened.
- Go to `File` then `Preferences`.
- In " Additional Boards Manager URL" field add: http://dan.drown.org/stm32duino/package_STM32duino_index.json
- Next click on `Tools`, `Board`, `Board Manager` and search for "STM32".
- Install "STM32F1xx/GD32F1xx" by stm32duino.
- Then select the corresponding board in `Tools` and make sure upload method is "ST-LINK"'.

Appendix C

Inertial Measurement Unit

The following chapter shows the two IMU analyzed and tested within the project. It has everything to check

C.1 MPU9025 module

This module is based on the MPU9250 sensor and includes several sensors to monitor 9-axis (9-DoF) motion. It is composed of a 3-axis gyroscope, a 3-axis accelerometer, and a 3-axis magnetometer. Furthermore, it has a DMP (Digital Motion Processor) that can run complex 9-axis motion capture algorithms [52].
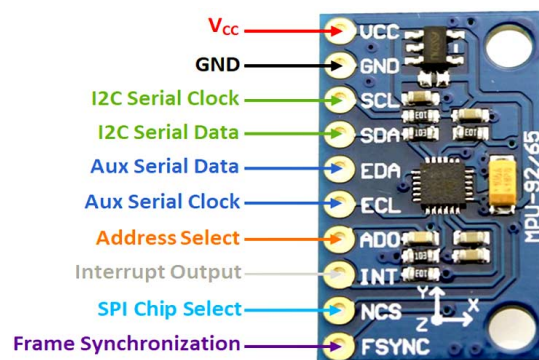
Figure C.1 MPU9250 Pin diagram. Source: Components101 [52].

It communicates with the microcontrollers through an I^{2}C interface and includes a widely used library for simple usage. This sensor has 9 degrees of freedom and has a 3.3V voltage regulator as well as pull-up resistors for direct I^{2}C application.

The accelerometer sensibility can be adjusted to work in different ranges from $\pm$2g, $\pm$4g, $\pm$8g, $\pm$16g. I the case of the magnetometer it has a sensibility of $\pm$4800 μT and $\pm$250°/s, $\pm$500°/s, $\pm$1000 °/s, $\pm$2000°/s gyroscope.

The technical specifications are summarized below:

- Operating voltage: 3 V, 3.3 V or 5 V

- Accelerometer Range: ±2g, ±4g, ±8g, ±16g

- Gyroscope Range: ±250°/s, ±500°/s, ±1000 °/s, ±2000°/s

- Magnetometer range: ±4800 μT

- Interface: I^{2}C and SPI

- AD converter: 16 Bits (digital output)

- Degrees of Freedom (DoF): 9

- Sensor: MPU9250

- Onboard voltage regulator

- Size: 2.0cm x 1.6cm x 0.3cm

Table C.1 MPU9250 Pin-outs. Source: Components101 [52].

Pin Number	Pin Name	Description
1	VCC	Power Supply
2	GND	Ground Reference
3	SCL	I^{2}C Serial Clock
4	SDA	I^{2}C Serial Data
5	EDA	Auxiliary Serial Data
6	ECL	Auxiliary Serial Clock
7	AD0	I^{2}C/SPI Address Select
8	INT	Interrupt
9	NCS	SPI Chip Select
10	FSYNC	Frame Synchronization
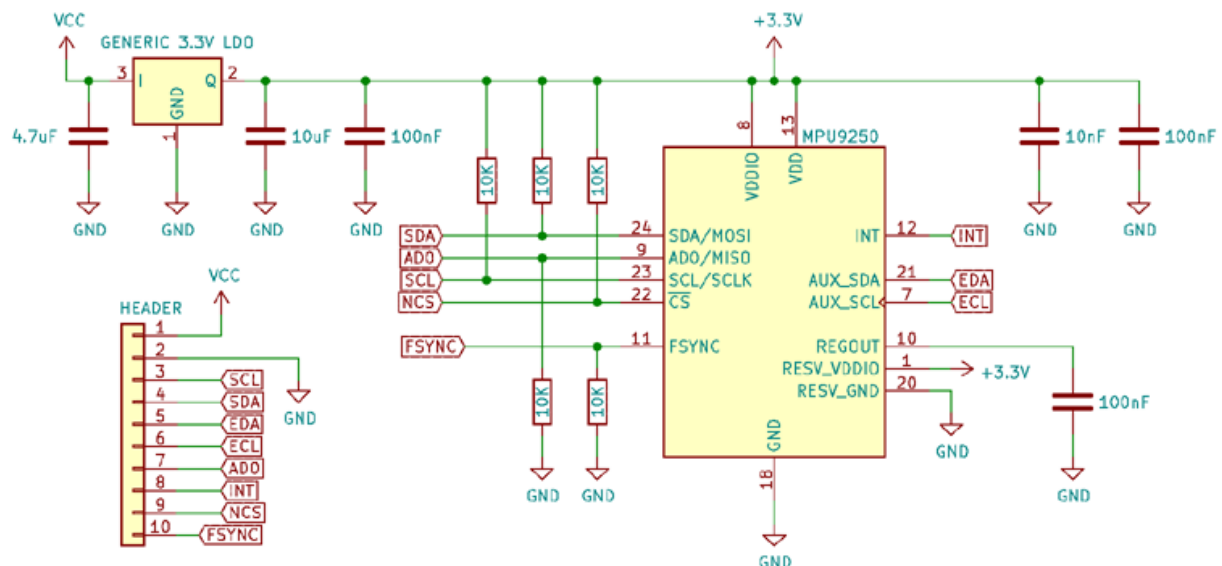
Figure C.2 MPU9250 Internal Circuit. Source: Components101 [52].

C.2   BNO055 module

The BNO055 is a System in Package (SiP) that combines in a single package a triaxial 14-bit accelerometer, a triaxial 16-bit gyroscope with a range of 2000 degrees per second, a triaxial geomagnetic sensor, and a 32-bit cortex M0+ microprocessor running Bosch Sensortec sensor fusion software.

The chip-sets are housed in a single 28-pin LGA chassis measuring 3.8mm x 5.2mm x 1.1mm. The BNO055 has digital bidirectional I2C and UART interfaces for optimal system integration. The BNO055 data output is versatile, it can provide both quaternions or Euler angles as well as providing directly the use of the RAW output as well (see items below).

The BNO055 can output the following sensor data [87]:

- Absolute Orientation (Euler Vector, 100 Hz)
- Three axis orientation data based on a 360° sphere
- Absolute Orientation (Quaternion, 100 Hz)
- Four point quaternion output for more accurate data manipulation
- Angular Velocity Vector (100 Hz)
- Three axis of 'rotation speed' in rad/s
- Acceleration Vector (100 Hz)
- Three axis of acceleration (gravity + linear motion) in m/s^2
- Magnetic Field Strength Vector (20 m/s^2)
- Three axis of magnetic field sensing in micro Tesla (μT)
- Linear Acceleration Vector (100 Hz)
- Three axis of linear acceleration data (acceleration minus gravity) in m/s^2
- Gravity Vector (100 Hz)
- Three axis of gravitational acceleration (minus any movement) in m/s^2
- Temperature (1 Hz)
- Ambient temperature in degrees Celsius

With respect to the pin-out of this sensor [88]:

Table C.2 BNO055 Pin-out connections. Source: [87].

Pins	Pin Name	Description
Power pins	VIN	3.3-5.0V power supply input
	3VO	3.3V output from the on-board linear voltage regulator, up to about 50mA as necessary
	GND	The common/GND pin for power and logic
I2C	SCL	I2C clock pin, connect to your microcontrollers I2C clock line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin
	SDA	I2C data pin, connect to your microcontrollers I2C data line. This pin can be used with 3V or 5V logic, and there's a 10K pullup on this pin.
Other pins	RST	Hardware reset pin. Set this pin low the high to perform a rest. This pin is 5V safe.
	ADR	Set this pin high to change the default I2C address for the BNO055 if you need to connect two ICs on the same I2C bus. The default address is 0x28. If this pin is connected to 3V, the address will be 0x29
	INT	The HW interrupt output pin, which can be configured to generate an interrupt signal when certain events occur like movement detected by the accelerometer, etc. (not currently supported in the Adafruit library, but the chip and HW is capable of generating this signal). The voltage level out is 3V
	PS0 and PS1	These pins can be used to change the mode of the device (it can also do HID-I2C and UART) and also are provided in case Bosch provides a firmware update at some point for the ARM Cortex M0 MCU inside the sensor. They should normally be left unconnected.

Appendix D

Logic Level Converter

The logic level converter is a fundamental piece when converting high voltage to low voltage and vice-versa. Processors, run at several voltages, the most popular of which are 5 V, 3.3 V, and, to a lesser extent, 2.8 and 1.8 V. 5 V. However, most current microcontrollers, though (Raspberry, Arduino, etc.), run at 3.3 V.

Nevertheless, the majority of the electrical components (sensors, controllers, and displays) operate at various nominal voltages, namely 5 V and 3.3 V. To link digital devices with various nominal voltages, the voltage levels must be adjusted. Otherwise, the assembly is unlikely to function well, and we may even damage a component.
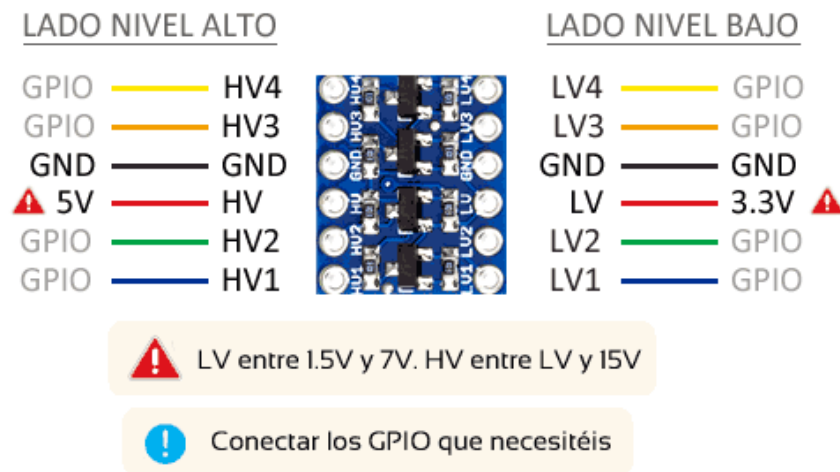
Figure D.1 Logic Level converter. Source: Luis Llamas [89].

To connect to a logic level shifter is a simple process. There are two sides, one for the device with the highest voltage (identified as HIGH) and one for the device with the lowest voltage (identified as LOW).

First, connect the GND of both devices to the GND of the board on the respective sides of the end adapter. GND has to be common for both devices, so the GND of both sides of the board are linked

internally. If not sure, check with a multimeter on both sides of the level adapter.

On the other hand, both devices to the level shifter must be connected to the two reference voltages of both devices to the level shifter. For this, there is the HV pin, for the Vcc voltage of the higher voltage device, and LV for Vcc of the lower voltage device.

Finally, GPIO pins (channels) can be used as many time as needed. The channels are bidirectional and work for both digital and analogue signals, or communication systems (UART, I^{2}C, SPI).

Appendix E

Bluetooth Module

The Bluetooth module selected in the present project is the HC-06 model from DSD Tech. The following section is a guide for setting up the Bluetooth module for both Arduino and STM32 microcontrollers.

As regards the pins on the HC-06 Bluetooth module [90],

- RXD: Serial Data Receive Pin. Used for serial input, 3.3V logic. Typically hooked up to transmission pin (TX) of the Arduino.
- TXD: Serial Data Transmit Pin. Used for serial output. 3.3V logic. Typically hooked up to reception pin (RX) of the Arduino
- GND: Ground. Typically hooked up to GND pin of the Arduino
- VCC: +5V. Power supply. Typically hooked up to 5V pin of the Arduino.

The connections are [90]:

$$
\begin{aligned}
\text{RXD} &\longrightarrow \text{TX} \\
\text{TXD} &\longrightarrow \text{RX} \\
\text{GND} &\longrightarrow \text{GND} \\
\text{VCC} &\longrightarrow \text{5V}
\end{aligned}
$$
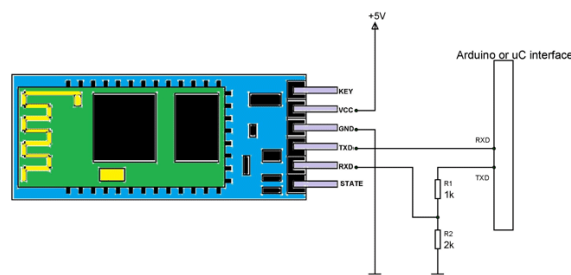
Figure E.1 HC-05 & HC-06 pin-out. Source: Components101 [90].

The characteristics of the module are [90]

- HC-06 Features and Electrical characteristics

- Bluetooth protocol: Bluetooth V2.0 protocol standard

- Power Level: Class2(+6dBm)

- Band: 2.40GHz—2.48GHz, ISM Band

- Receiver sensitivity: -85 dBm

- USB protocol: USB v1.1/2.0

- Modulation mode: Gauss frequency Shift Keying

- Safety feature: Authentication and encryption

- Operating voltage range:+3.3V to +6V

- Operating temperature range: -20ºC to +55ºC

- Operating Current: 40mA

On the other hand, the HC-05 shares similar pins [91]

Table E.1 HC-05 pins and its purpose. Source: AranaCorp [91].

Pin	Name	Function
1	Key	The pin state determines whether the module works in AT command mode or normal mode*
2	Vcc	+5V Positive supply needs to be given to this pin for powering the module
3	Gnd	Connect to ground
4	TXD	Serial data is transmitted by module through this pin (at 9600bps by default), 3.3V logic
5	RXD	Serial data is received by module through this pin (at 9600bps by default),3.3V logic
6	State	The pin is connected to the LED on the board to represent the state of the module

*High=AT commands receiving mode(Commands response mode), Low or NC= Bluetooth module normally working

When setting up the Bluetooth, the process is detailed in [90],

1. The user must set the default baud rate of UART serial communication to 9600 in the IDE. The value is the module's default setting and can be changed in the application.

2. Because the module is a slave, a master is required to create a successful wireless link. Another [arduino + module (with master feature)] setup is required for this, alternatively, the user may use a mobile phone as a master and search for HC-06 slave. The HC-06 module comes with a default password of '1234,' which may be altered.

3. The user may also use libraries to facilitate communication. Just need to download them and add them to the header. After it is set, then it is possible to command the Arduino to transmit or receive data. This data is sent to the master over wireless Bluetooth by the module. If the module gets data from the master, it will send it to the Arduino through UART serial connection.

4. To connect with the module, use the AT command on a serial terminal, which may be obtained here.

This section aims to develop a code that uses HC06 Bluetooth module for telemetry and telecommand.

E.1   Initial Bluetooth module code

Make sure to configure the Bluetooth using a Arduino device.

```
1  /* HC06 bluetooth module setup
2  */
3
4  // Libraries
5  #include <SoftwareSerial.h>   // This library enables bluetooth communcication
6
7  // HC06 parameters
8  const int Rx = 2;            // Connect Arduino Digital Pin 2 to HC06 Pin Tx
9  const int Tx = 3;            // Connect Arduino Digital Pin 3 to HC06 Pin Rx
10 SoftwareSerial hc06(Rx, Tx); // Set Receive and Transmission pins
11
12 // Configuration parameters
13 char BPS = '4';                    // 1=1200 , 2=2400, 3=4800, 4=9600, 5=19200, ...
        6=38400, 7=57600, 8=115200
14 char NAME_ID[20] = "HC06_Plathon"; // 30 characters maximum
15 char PASS[10] = "1234";            // 4 bit number
16
17 // Setup
18 void setup()
19 {
20   //Initialize Serial Monitor
21   Serial.begin(9600);
22
23   //Initialize Bluetooth Serial Port
24   hc06.begin(9600);
25
26   // AT commands for configuration
27   hc06.print("AT"); // Initiate sequence
28   delay(1000); // Wait 1 second
29
30   hc06.print("AT+NAME");
31   hc06.print(NAME_ID); // Configure bluetooth device name
32   delay(1000); // Wait 1 second
33
34   hc06.print("AT+BAUD");
35   hc06.print(BPS); // Transmission baud rate velocity
36   delay(1000); // Wait 1 second
```

```
37
38    hc06.print("AT+PIN");
39    hc06.print(PASS); // Set Password
40    delay(1000); // Wait 1 second
41
42    // Print setup message
43    Serial.println("Module set up");
44    Serial.print("PIN:");
45    Serial.println(PASS);
46    Serial.print("Name: ");
47    Serial.println(NAME_ID);
48
49    // Wait 1000 ms to ensure the sensor starts correctly
50    delay(1000);
51  }
52
53  // Send and receive data
54  void loop()
55  {
56    //Write data from HC06 to Serial Monitor
57    if (hc06.available())
58    {
59      Serial.write(hc06.read());
60    }
61
62    //Write from Serial Monitor to HC06
63    if (Serial.available())
64    {
65      hc06.write(Serial.read());
66    }
67
68  }
```

Code E.1 HC06 Bluetooth Module configuration. Source: Own.

E.2   Send and Receive data from module (Arduino Nano) code

```
1  /* This code reads all sensors data from the BNO055 IMU Sensor
2   *  and uses bluetooth communication
3   */
4
5  // Libraries
6  #include <Wire.h>             // This library allows to communicate with I2C / TWI ...
        devices.
7  #include <Adafruit_Sensor.h>  // Library with drivers that are based on the ...
        Adafruit Unified Sensor Driver
8  #include <Adafruit_BNO055.h>  // This is a library for the BNO055 orientation sensor
9  #include <utility/imumaths.h> // Inertial Measurement Unit Maths Library (it ...
        includes matrix.h, quaternions.h and vector.h)
10  #include <math.h>            // Math functions
11  #include <SoftwareSerial.h>  // This library enables bluetooth communcication
12
13  // Global parameters and objects
```

160

```
14  #define BNO055_SAMPLERATE_DELAY_MS (100) // Define how fast the sensor sample rate ...
        (sample every 100 ms)
15
16  Adafruit_BNO055 IMU = Adafruit_BNO055(); // Create IMU object and set what the ...
        object is
17
18  const int Rx = 2;              // Connect Arduino Digital Pin 2 to HC06 Pin Tx
19  const int Tx = 3;              // Connect Arduino Digital Pin 3 to HC06 Pin Rx
20  SoftwareSerial hc06(Rx, Tx); // Set Receive and Transmission pins
21
22  /* Function Prototypes*/
23  // Calibration function
24  void displayCalStatusSerial(void);
25  void displayCalStatusBluetooth(void);
26  void printSerial(void);
27  void prinBluetooth(void);
28
29  void setup()
30  {
31    //Initialize Serial Monitor
32    Serial.begin(9600);
33
34    // Start the IMU sensor
35    IMU.begin();
36
37    //Initialize Bluetooth Serial Port
38    hc06.begin(9600);
39
40    // Wait 1000 ms to ensure the sensor starts correctly
41    delay(1000);
42
43    // Change the time clock on the chip to the time clock on board of the IMU
44    IMU.setExtCrystalUse(true);
45  }
46
47  void loop()
48  {
49
50    // Print calibration data through serial
51    // displayCalStatusSerial();
52    // Print calibration data
53    displayCalStatusBluetooth();
54
55    // Print data over Bluetooth port
56    printBluetooth();
57    // Print data over Serial Port
58    // printSerial();
59
60    //Write data from HC06 to Serial Monitor
61    if (hc06.available())
62    {
63      Serial.write(hc06.read());
64    }
65
66    //Write from Serial Monitor to HC06
67    if (Serial.available())
68    {
69      hc06.write(Serial.read());
```

```
70    }
71
72    // Wait the specified delay before requesting next data
73    delay(BNO055_SAMPLERATE_DELAY_MS);
74  }
75
76  // Calibration function
77  void displayCalStatusSerial(void)
78  {
79    /*  Get the four calibration values (0..3)
80        Any sensor data reporting 0 should be ignored,
81        3 means 'fully calibrated" */
82    uint8_t system, gyros, accel, mg = 0;
83    IMU.getCalibration(&system, &gyros, &accel, &mg);
84
85    // The data should be ignored until the system calibration is > 0
86    // Display the individual values
87    Serial.print(system, DEC);
88    Serial.print(",");
89    Serial.print(gyros, DEC);
90    Serial.print(",");
91    Serial.print(accel, DEC);
92    Serial.print(",");
93    Serial.print(mg, DEC);
94  }
95
96  // Calibration function for bluetooth
97  void displayCalStatusBluetooth(void)
98  {
99    /*  Get the four calibration values (0..3)
100       Any sensor data reporting 0 should be ignored,
101       3 means 'fully calibrated" */
102   uint8_t system, gyros, accel, mg = 0;
103   IMU.getCalibration(&system, &gyros, &accel, &mg);
104
105   // The data should be ignored until the system calibration is > 0
106   // Display the individual values
107   hc06.print(system, DEC);
108   hc06.print(",");
109   hc06.print(gyros, DEC);
110   hc06.print(",");
111   hc06.print(accel, DEC);
112   hc06.print(",");
113   hc06.print(mg, DEC);
114 }
115
116 // Print serial
117 void printSerial(void)
118 {
119   // Work with the imu sensor from Adafruit library
120   // Go to BNO055's 'imu' and bring back a vector of 4 components into 'quat' ...
           (quaternion) for the specific object IMU
121   imu::Quaternion quat = IMU.getQuat();
122   // Go to BNO055's 'imu' and bring back a vector of 3 components into 'acc' ...
           (accelerometer) for the specific object IMU
123   imu::Vector<3> acc = IMU.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
124   // Go to BNO055's 'imu' and bring back a vector of 3 components into 'gyro' ...
           (gyroscope) for the specific object IMU
```

```
125    imu::Vector<3> gyro = IMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
126    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'magn' ...
           (magnetometer ) for the specific object IMU
127    imu::Vector<3> magn = IMU.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
128
129    // Print Quaternion data
130    Serial.print(quat.w()); // Real part
131    Serial.print(",");
132    Serial.print(quat.x()); // 'i'
133    Serial.print(",");
134    Serial.print(quat.y()); // 'j'
135    Serial.print(",");
136    Serial.print(quat.z()); // 'k'
137    Serial.print(",");
138
139    // Print Acceleration data
140    Serial.print(acc.x());
141    Serial.print(",");
142    Serial.print(acc.y());
143    Serial.print(",");
144    Serial.print(acc.z());
145    Serial.print(",");
146
147    Serial.print(gyro.x());
148    Serial.print(",");
149    Serial.print(gyro.x());
150    Serial.print(",");
151    Serial.print(gyro.x());
152    Serial.print(",");
153
154    Serial.print(magn.x());
155    Serial.print(",");
156    Serial.print(magn.y());
157    Serial.print(",");
158    Serial.println(magn.z());
159  }
160
161  // Print Bluetooth
162  void printBluetooth(void)
163  {
164    // Work with the imu sensor from Adafruit library
165    // Go to BNO055's 'imu' and bring back a vector of 4 components into 'quat' ...
           (quaternion) for the specific object IMU
166    imu::Quaternion quat = IMU.getQuat();
167    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'acc' ...
           (accelerometer) for the specific object IMU
168    imu::Vector<3> acc = IMU.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
169    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'gyro' ...
           (gyroscope) for the specific object IMU
170    imu::Vector<3> gyro = IMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
171    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'magn' ...
           (magnetometer ) for the specific object IMU
172    imu::Vector<3> magn = IMU.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
173
174    // Print Quaternion data
175    hc06.print(quat.w()); // Real part
176    hc06.print(",");
177    hc06.print(quat.x()); // 'i'
```

```
178     hc06.print(",");
179     hc06.print(quat.y()); // 'j'
180     hc06.print(",");
181     hc06.print(quat.z()); // 'k'
182     hc06.print(",");
183
184     // Print Acceleration data
185     hc06.print(acc.x());
186     hc06.print(",");
187     hc06.print(acc.y());
188     hc06.print(",");
189     hc06.print(acc.z());
190     hc06.print(",");
191
192     hc06.print(gyro.x());
193     hc06.print(",");
194     hc06.print(gyro.x());
195     hc06.print(",");
196     hc06.print(gyro.x());
197     hc06.print(",");
198
199     hc06.print(magn.x());
200     hc06.print(",");
201     hc06.print(magn.y());
202     hc06.print(",");
203     hc06.println(magn.z());
204  }
```

Code E.2 Send data through Bluetooth module with Arduino Nano. Source: Own.

E.3   Send and Receive data from module (STM32 Bluepill) code

```
1  /*
2   * UART Bluetooth Communication with STM32
3   */
4
5  // Libraries
6  #include <Arduino.h>
7  #include <time.h>
8
9  // Parameters
10 char inputData_serial1 = 0; // Initialize input data from Serial 1
11 char inputData_serial = 0;  // Initialize input data from Serial
12
13 // Main setup
14 void setup()
15 {
16   Serial1.begin(9600);                    // Begin UART serial port
17   Serial.begin(9600);                     // Begin Serial port
18   Serial1.println("Bluetooth initialized"); // Print initilize message
19 }
20
21 // Main loop
```

```
22  void loop()
23  {
24    // Check if serial is available
25    if (Serial1.available() > 0)
26    {
27      inputData_serial1 = Serial1.read();
28      Serial.write(inputData_serial1); // Display message throuhg Serial port
29    }
30    // If Serial
31    if (Serial.available() > 0)
32    {
33      inputData_serial = Serial.read();
34      Serial1.write(inputData_serial); // Display message throuhg Serial 1 port
35    }
36
37    // Delay 100 ms
38    delay(100);
39  }
```

Code E.3 Send and receive data through HC06 with STM32. Source: Own.

Let's see an example of Bluetooth and the Arduino. First, it is needed to configure the Bluetooth parameters, this can be done using the code provided in E.1

After that, it is possible to test data sending and receiving via the HC-06 module as shown in E.2.

In the code implementation, a baud rate of 9600 was set since otherwise the Bluetooth module does not show up on the phone.
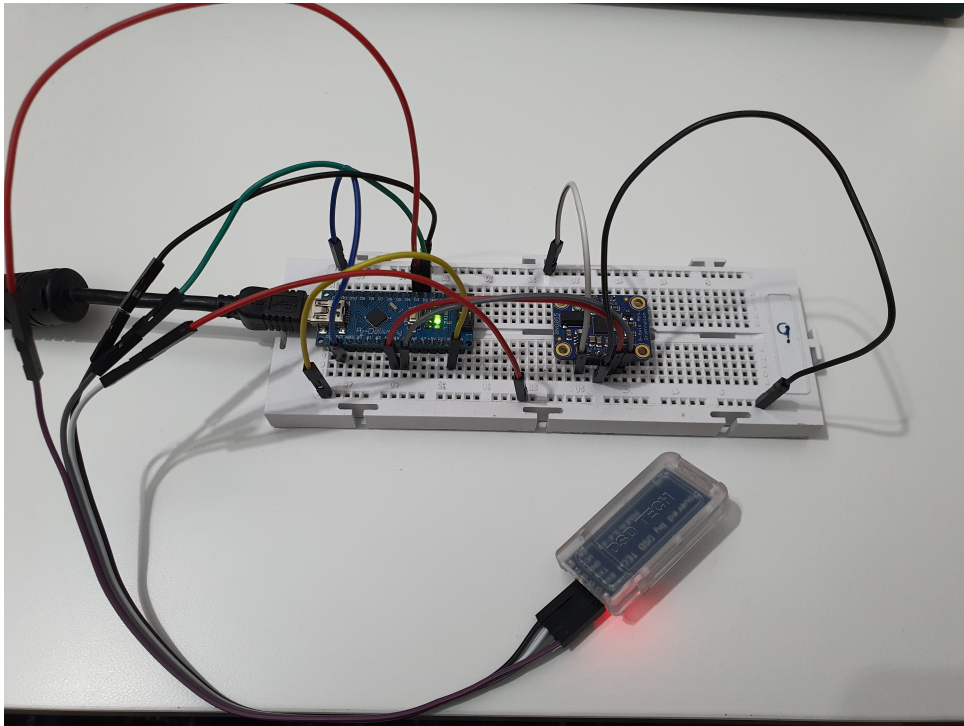
Figure E.2 Bluetooth setup schematic. Source: Own.
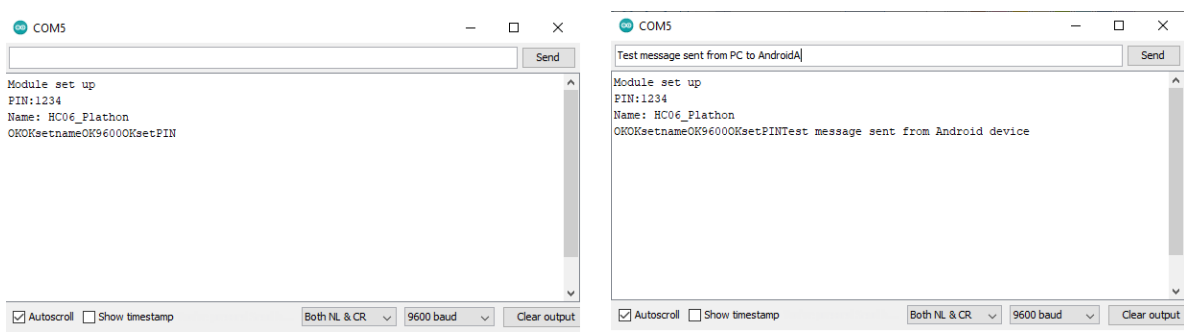
(a) Bluetooth setup schematic. Source: Own.

(b) Bluetooth setup schematic. Source: Own.

Figure E.3 Bluetooth Serial COM. Source: Own.

Appendix F

Motor

Here is presented the datasheet of each motor RF-300EA and RF-500TB from Mabuchi Motors [60] [59].

F.1 Motor Designations and their Meanings

Let's revise first the nomenclature of the motor. Each motor has a designation that collects the most important aspects of it.

- The types of motors are normally designated by a seven-digit code.
- The following 3 to 6 digits designate the armature winding specifications.
- The meaning of each character or figure is as shown in Figure F.1.
- However, since there are some exceptions, contact our sales office for further information.
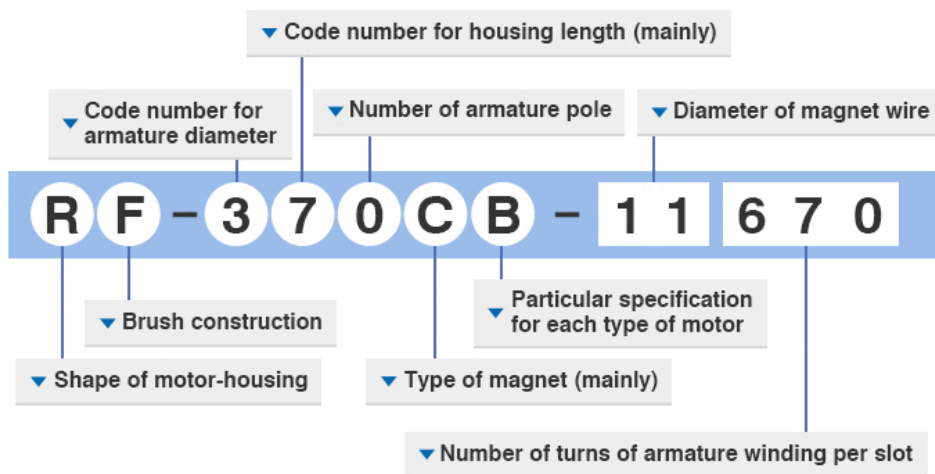
Figure F.1 Motor designation. Source: Mabuchi Motors [92].

Shape of motor-housing
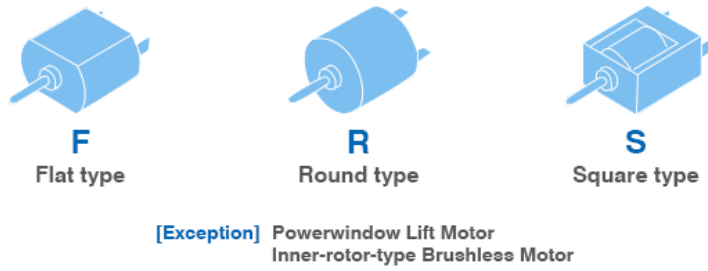
Figure F.2 Motor shape. Source: Mabuchi Motors [92].

Brush construction

Figure F.3 Motor brush construction. Source: Mabuchi Motors [92].

Code number for armature diameter
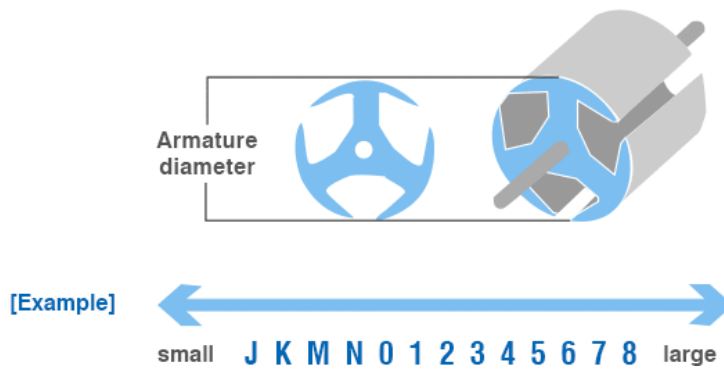
Figure F.4 Motor armature diameter code. Source: Mabuchi Motors [92].

Code number for housing length (mainly)
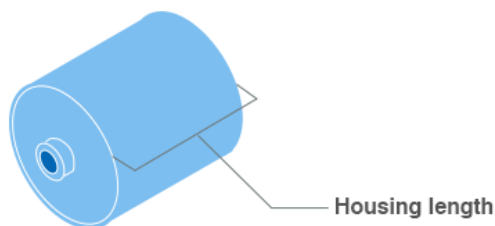
Figure F.5 Motor housing length code. Source: Mabuchi Motors [92].

Number of armature pole
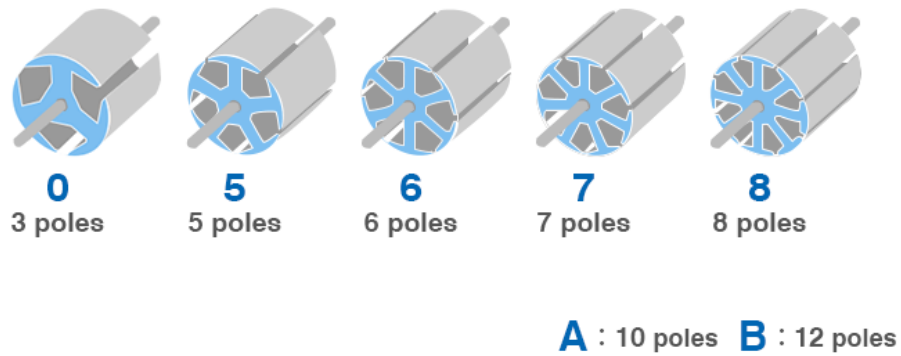
Figure F.6 Motor number of armature pole. Source: Mabuchi Motors [92].

Type of magnet (mainly)
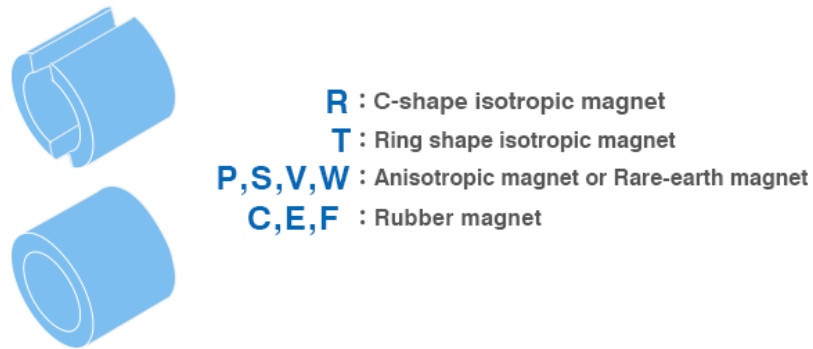
Figure F.7 Motor type of magnet. Source: Mabuchi Motors [92].

Diameter of magnet wire
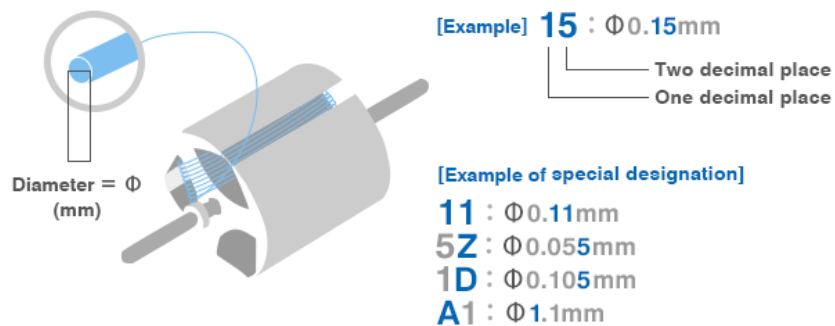
Figure F.8 Motor diameter of magnet wire. Source: Mabuchi Motors [92].

Number of turns of armature winding per slot
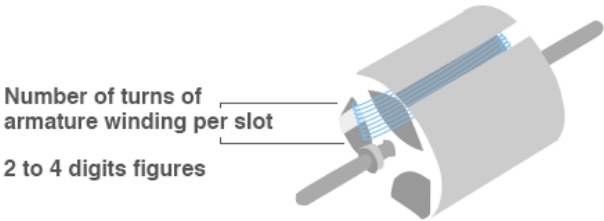
Figure F.9 Motor number of turns of armature winding per slot. Source: Mabuchi Motors [92].

Hereafter is presented the datasheets for each motor:
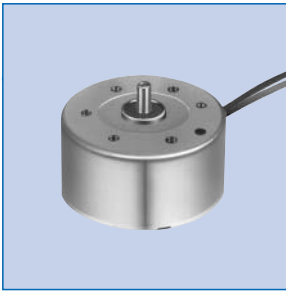
RF-300EA

MABUCHI MOTOR
Precious metal-brush motors

OUTPUT : 0.02W~1.8W (APPROX)

WEIGHT : 22g (APPROX)

Typical Applications   **Office Automation Equipment :** CD/DVD-ROM Drive
Audio and Visual Equipment : Car CD Player / DVD Player

MODEL	VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY					STALL		
	OPERATING RANGE	NOMINAL	SPEED	CURRENT	SPEED	CURRENT	TORQUE		OUTPUT	TORQUE		CURRENT
			r/min	A	r/min	A	mN·m	g·cm	W	mN·m	g·cm	A
RF-300EA-1D390	2.8~7.0	3.9V CONSTANT	4400	0.021	3520	0.084	0.47	4.8	0.17	2.35	24	0.34
RF-300EA-8Z485	1.6~11.0	8V CONSTANT	7100	0.021	5760	0.090	0.62	6.3	0.37	3.28	33	0.39

Also available with terminals (without leadwires).

DIRECTION OF ROTATION

60°
60°
φ16.5

JIS M1.7×0.35 TAPPED HOLE
6 PLACES
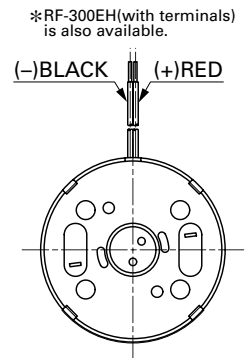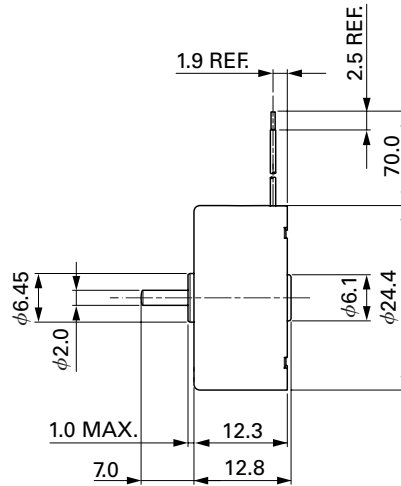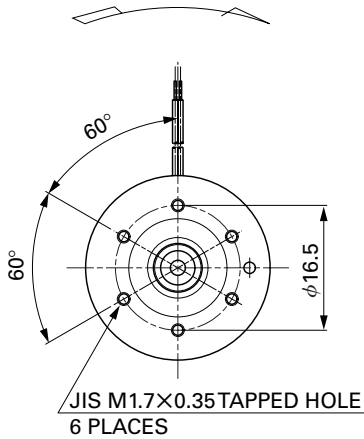
Usable machine screw length 1.1 max. from
motor mounting surface.

1.9 REF.
2.5 REF.
70.0
φ6.45
φ2.0
φ6.1
φ24.4
1.0 MAX.
12.3
7.0
12.8

SHAFT LENGTH 19.0

※RF-300EH(with terminals)
is also available.

(−)BLACK   (+)RED

UNIT: MILLIMETERS

RF-300EA-1D390   3.9V

η / I / N

η
N
I
Is 0.34
Ts 2.35

EFFICIENCY [%]
CURRENT [A]
SPEED [r/min]

1   [mN·m]   2
10   [g·cm]   20
TORQUE

RF-300EA-8Z485   8.0V

η / I / N

η
N
I
Is 0.39
Ts 3.28

EFFICIENCY [%]
CURRENT [A]
SPEED [r/min]

2.5   [mN·m]   5.0
25   [g·cm]   50
TORQUE
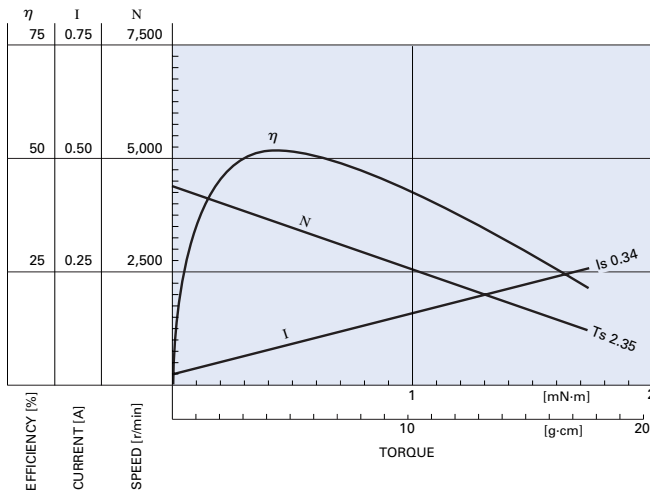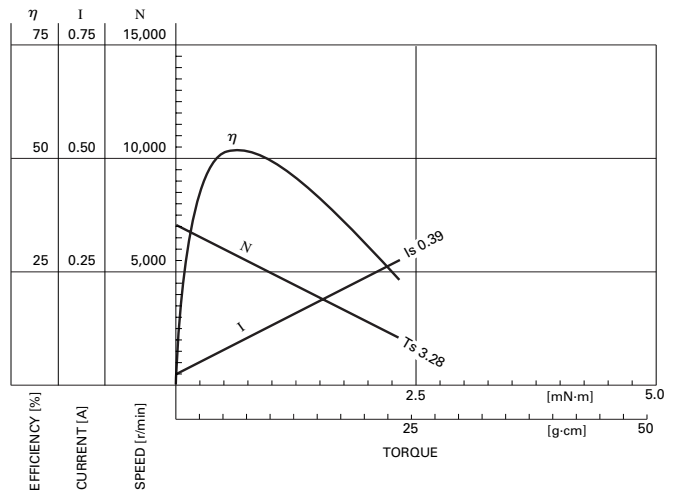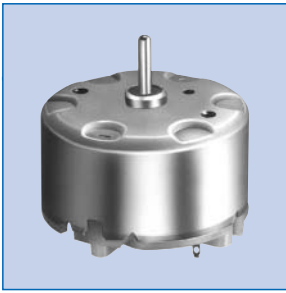
RF-500TB

MABUCHI MOTOR
Precious metal-brush motors

OUTPUT : 0.01W~2.0W (APPROX)　　　WEIGHT : 45g (APPROX)

Typical Applications　**Audio and Visual Equipment :** CD Player / DVD Player / VCR

MODEL	VOLTAGE		NO LOAD		AT MAXIMUM EFFICIENCY					STALL		
	OPERATING RANGE	NOMINAL	SPEED	CURRENT	SPEED	CURRENT	TORQUE		OUTPUT	TORQUE		CURRENT
			r/min	A	r/min	A	mN·m	g·cm	W	mN·m	g·cm	A
RF-500TB-14415	1.5~9.0	5V CONSTANT	3100	0.026	2540	0.12	1.23	12.6	0.33	6.86	70	0.54
RF-500TB-12560	1.5~12.0	6V CONSTANT	2700	0.020	2180	0.084	1.13	11.6	0.26	5.88	60	0.35

DIRECTION OF ROTATION

33.5 REF.
10.7　22.8
1.9　19.5
17.5
0.9　16.6

ϕ32.0
ϕ6.2
ϕ2.0
ϕ7.5
(+)
(−)

SHAFT LENGTH 32.0

3.2
ϕ1.1 HOLE
1.9
0.3
20.3 REF.
4.5 REF.

22.0
30°
ϕ
R8.0

JIS M2.6×0.45 TAPPED HOLE
2 PLACES
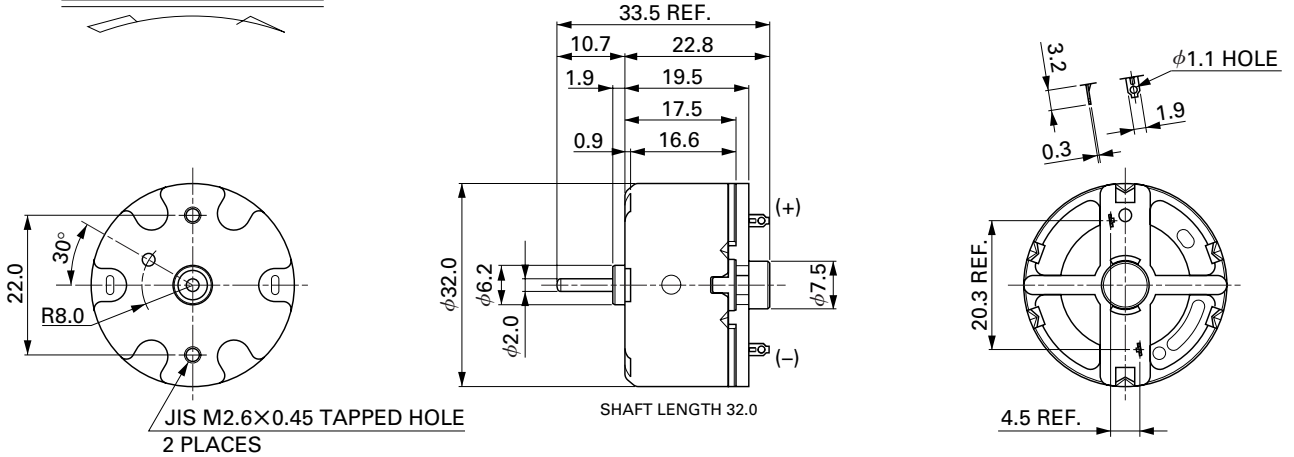
Usable machine screw length 1.5 max. from
motor mounting surface.

UNIT: MILLIMETERS

RF-500TB-14415　　　5.0V

η　I　N
75　0.75　7,500
50　0.50　5,000
25　0.25　2,500

Is 0.54
Ts 6.86

η
N
I

EFFICIENCY [%]　CURRENT [A]　SPEED [r/min]

2.5　[mN·m]　5.0
25　[g·cm]　50
TORQUE

RF-500TB-12560　　　6.0V

η　I　N
75　0.75　3,000
50　0.50　2,000
25　0.25　1,000

N
η
I
Is 0.35
Ts 5.88

EFFICIENCY [%]　CURRENT [A]　SPEED [r/min]

2.5　[mN·m]　5.0
25　[g·cm]　50
TORQUE

Ø 2,00
Ø 6,20
Ø 32,00

2,10
8,60
Ø 7,50

3,00
7,88

10,70
33,50
19,50
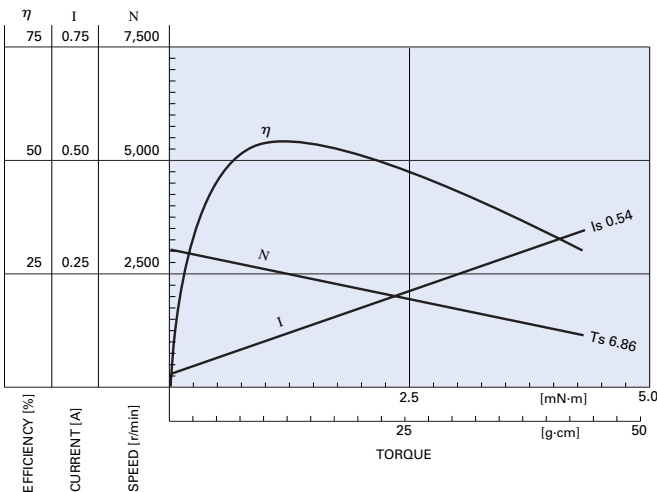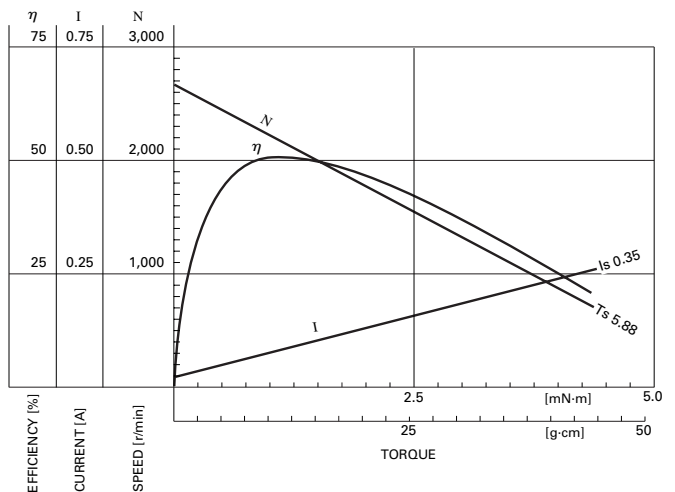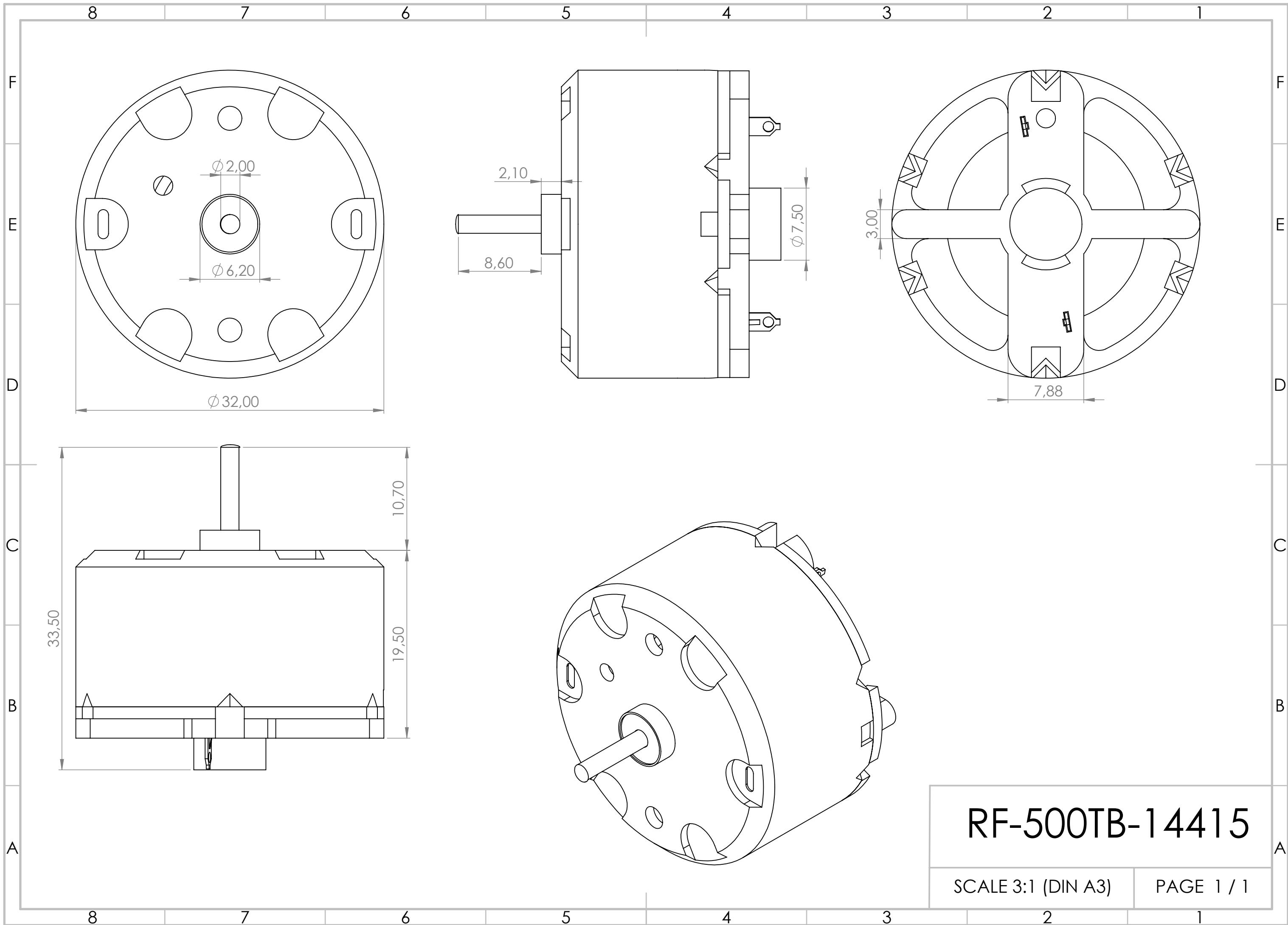
RF-500TB-14415

SCALE 3:1 (DIN A3) PAGE 1 / 1

Appendix G

SparkFun Qwiic Motor Driver

A motor driver is used to convert a low strength signal to high strength current signal to adequately control the power with a sufficient amount of power from an external source. This section aims to describe all the features of the SparkFun Qwiic Rob-15451 motor driver [56].

G.1 SparkFun Qwiic Rob-15451

As mentioned earlier, the SparkFun Qwiic Rob-15451 motor driver was chosen to control the motor.
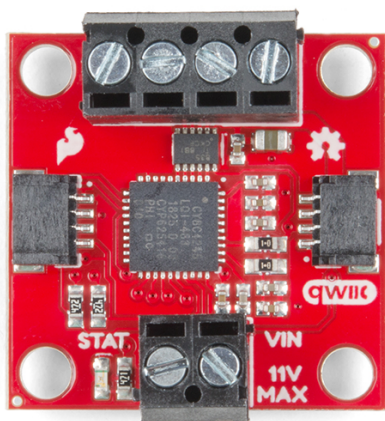
(a) SparkFun Qwiic Motor driver ROB 15451 top view. Source: Sparkfun [56].
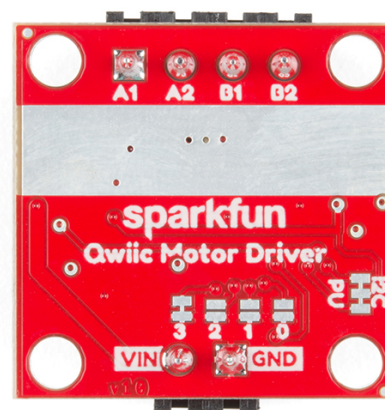
(b) SparkFun Qwiic Motor driver ROB 15451 bottom view. Source: Sparkfun [56].

Figure G.1 Braking configuration (left) and incorrect use-case configuration (right). Source: Core Electronics [56]

Table G.1 main shows the main characteristics and features of this motor.

SparkFun's Qwiic motor driver specifications and features
1.5 A peak drive per channel, 1.2 A steady state
Operates from 3 to 11 volts with 12 V absolute max
3.3 V default VCC and logic
127 levels of DC drive strength.
Controllable by I^{2}C or TTL UART signals
Direction inversion on a per motor basis
Global Drive enable
Exposed small heat sink shape
Several I^{2}C addresses, default UART bauds available

Table G.1 SparkFun's Qwiic motor driver specifications and features.
Source: SparkFun [56].

G.2   Power

There are two separate power circuits on this board. The power for the motors is supplied through the VIN Connectors and it supports voltage ranges from 3.3 V to 11 V with the MAX11V and GND connections. On the other hand, power for the PSoC (Programmable System on Chip), i.e. the STM32, and the logic circuits are provided by the 3.3 V input on the Qwiic connectors. All these two power sources are needed for proper functioning.
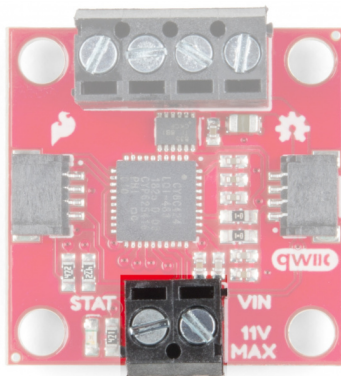
Figure G.2 Qwiic motor driver power port. Source: SparkFun [56].

G.3   Qwiic connectors

SparkFun's Qwiic Connect System uses 4-pin JST connectors to quickly interface development boards with sensors, LCDs, relays and more. There are two Qwiic connectors on the board such that two motors can be controlled by a single motor driver (see Figure G.3).
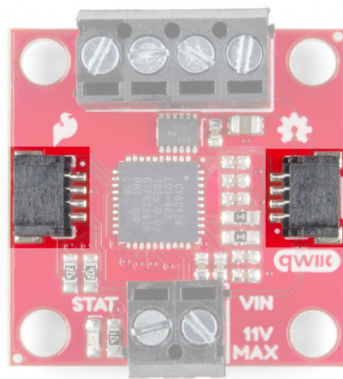
Figure G.3 Qwiic motor driver port. Source: SparkFun [2].

Unlike traditional connectors, this type of connector is thinner and lighter. The 28 AWG wire's length is 150 *milli*m and its properties are shown in Tables G.2 and G.3.

AWG gauge	Conductor Diameter Inches	Conductor Diameter mm	Conductor cross section in mm^2	Ohms per 1000 ft	Ohms per km
8	0.0126	0.32004	0.080	64.9	212.872

Table G.2 8 AWG gauge cable specifications. Source: Handbook of Electronic Tables and Formulas for American Wire Gauge [93].

Maximum A for chassis wiring	Maximum A for power transmission	Maximum frequency for 100% skin depth for solid conductor copper	Breaking force Soft Annealed Cu 37000 PSI
1.4	0.226	170 kHz	4.5 lbs

Table G.3 8 AWG gauge cable specifications. Source: Handbook of Electronic Tables and Formulas for American Wire Gauge [93].

The Qwiic adapter's wires have been colour coded to red, black, blue and yellow (see Table G.4).

Color scheme	Arrangement
Black	GND
Red	3.3V
Blue	SDA
Yellow	SCL

Table G.4 Color scheme. Source: SparkFun [94].
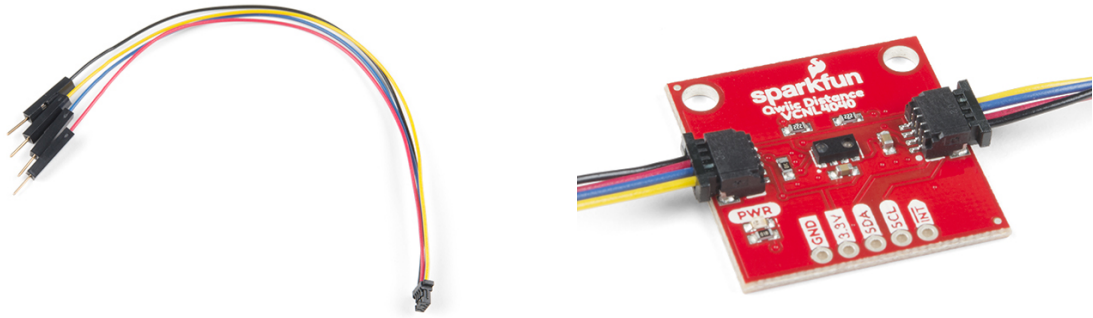
(a) Qwiic Cable to Breadboard Jumper (4-pin). Source: Sparkfun [95].

(b) Qwiic Cable connected to the board. Source: Sparkfun [95].

Figure G.4 Qwiic cable. Source: SparkFun [95]

As shown in the specification by the AWG standard the maximum current on a Qwiic cable is 226 mA. Despite it is possible to push it up to 1.4 A for chassis wiring in an isolated, unbundled wire in free air conditions as per Handbook of Electronic Tables and Formulas for American Wire Gauge, though. It is not recommended to stress the cable to that level but that cable can easily support a few hundreds of mA.

G.4   Motor ports

There are screw pin terminals at the top of the board that allows two motor connections. They are labelled on the backside of the board.
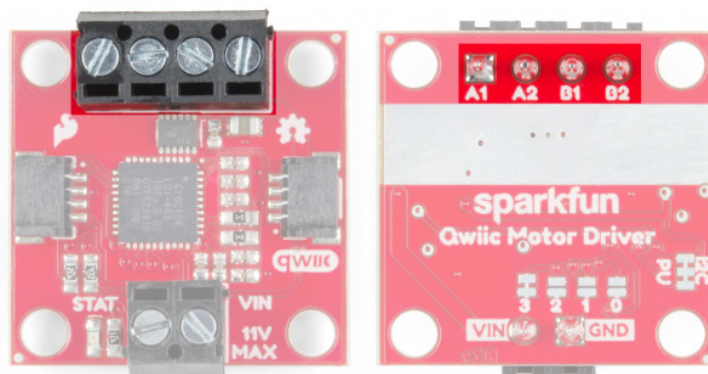
Figure G.5 Qwiic motor driver power port. Source: SparkFun [56].

Table G.5 summarizes the connection name and communication protocols.

Group	Name	Direction	Description	UART	I^{2}C
Motor Port	A1	O	Winding of first addressable location	Motor A winding	
	A2	O	Winding of first addressable location	Motor A winding	
	B1	O	Winding of second addressable location	Motor B winding	
	B2	O	Winding of second addressable location	Motor B winding	

Table G.5 Qwiic Motor driver motor ports. Source: Source: SparkFun [56]

G.5 Jumpers

There are 2 sets of jumpers to configure on this board. There are pull-up enables for I^{2}C and 4 config bits that select operational mode.

- **I^{2}C Jumpers**: I^{2}C pull-up enable. Opening these disables the I^{2}C pull-up resistors used for I^{2}C communication. If multiple I^{2}C devices are being used, these pull-ups should be disabled on all but one device. If UART is being used, the pull-up resistors should be disabled.

- **Address Jumpers**: Serial and function selection. The configuration bits are 4 bits that form a configuration nybble (half-byte). A closed jumper is a '1' and an open jumper is a '0'. See configuration Table G.6 for more information.
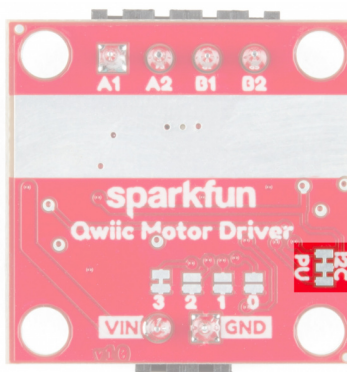
Figure G.6 Qwiic motor driver pull up jumpers. Source: SparkFun [56].

G.5.1 Address bits

The configuration is set by encoding a number into the 4 config bits on the bottom of the board. Close a jumper to indicate a 1, or leave it open to indicate a 0.
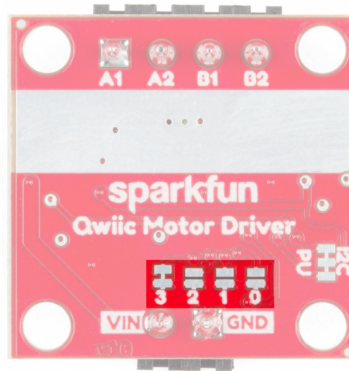
Figure G.7 Qwiic motor driver jumpers. Source: SparkFun [56].

Use this table as a reference to see what the user port, address, and expansion port will become in each configuration (the default configuration is shown in bold):

Pattern	Mode and User port	User Address	Expansion Port
0000	UART at 9600	N/A	Master
0011	I^{2}C	0x58	Master
0100	I^{2}C	0x59	Master
0101	I^{2}C	0x5A	Master
0110	I^{2}C	0x5B	Master
0111	I^{2}C	0x5C	Master
1000	I^{2}C	**0x5D**	**Master**
1001	I^{2}C	0x5E	Master
1010	I^{2}C	0x5F	Master
1011	I^{2}C	0x60	Master
1100	I^{2}C	0x61	Master
1101	UART at 57600	N/A	Master
1110	UART at 115200	N/A	Master
1111	N/A	N/A	N/A

Table G.6 Motor driver configurations. Source: SparkFun [56].

G.6 Thermal Conduction Area

The Qwiic Motor Driver is designed to operate small robot drive motors without a heatsink; we were able to run up to about 1.1 A continuous current without going above 100 °C. Nonetheless, if a heatsink is needed, Theragrip Thermal Tape can be used to attach three Small Heat Sinks across the thermal conduction area on the back of the board.
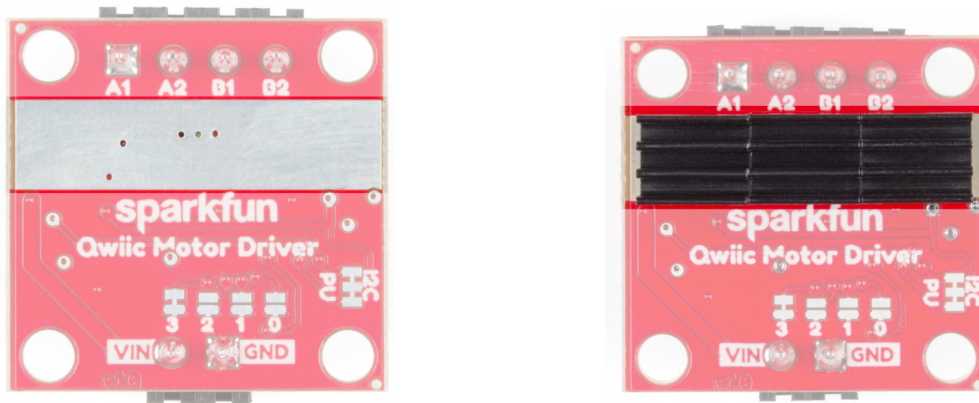
(a) Qwiic motor driver thermal conduction area. Source: Sparkfun [95].

(b) Qwiic motor driver heatsink. Source: Sparkfun [95].

Figure G.8 Thermal system of the motor driver. Source: SparkFun [95]

G.7   Board dimensions

The following blueprint (Figure G.9) depicts the dimensions of the board (please note that all dimensions are in inches).
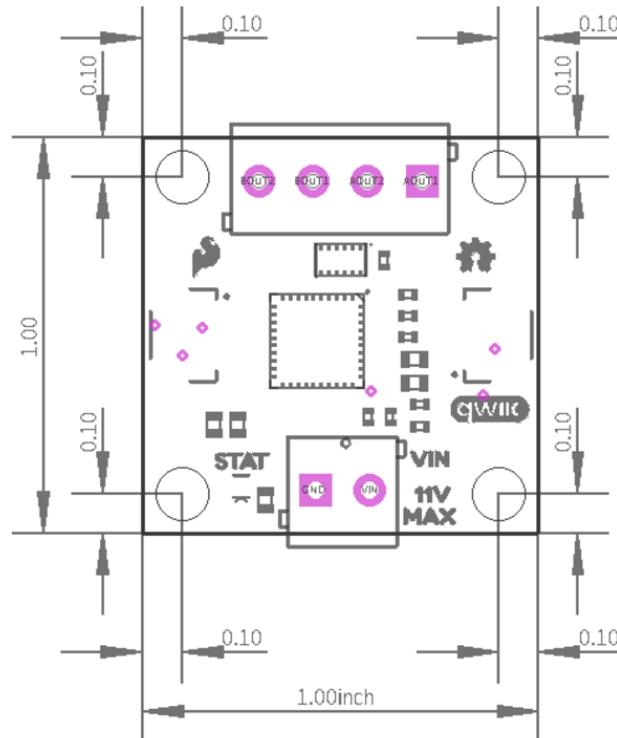
Figure G.9 Qwiic motor driver dimensions. Source: SparkFun [56].

Connectors

VIN: 11V MAX

J1 — Power In — VIN, GND, 1, 2

J3 — Qwiic — 3.3V — SCL 4, SDA 3, VCC 2, GND 1 — SCL, SDA, GND

J2 — Motor Connections — BOUT2 4, BOUT1 3, AOUT2 2, AOUT1 1

J4 — Qwiic — 3.3V — SCL 4, SDA 3, VCC 2, GND 1 — SCL, SDA, GND

Status LED

SLED — R1 470 — D1 BLUE — GND

H—Bridge Driver

DRV8835 — U1

VIN, 3.3V

VM 1, VCC 12, MODE 11, AIN1/PH 10, AIN2/EN 9, BIN1/PH 8, BIN2/EN 7, VSS 6*2

AOUT1 2, BOUT1 4, AOUT2 3, BOUT2 5

P2.0 10, P2.1 11, P2.2 9, P2.3 8, P2.4 7

C1 10uF, C2 0.1uF — GND, GND, GND

AOUT1, BOUT1, AOUT2, BOUT2
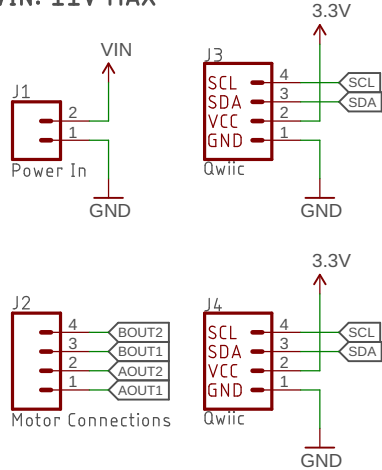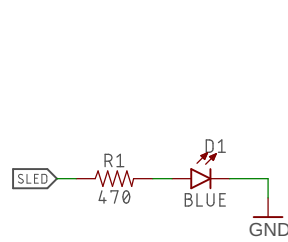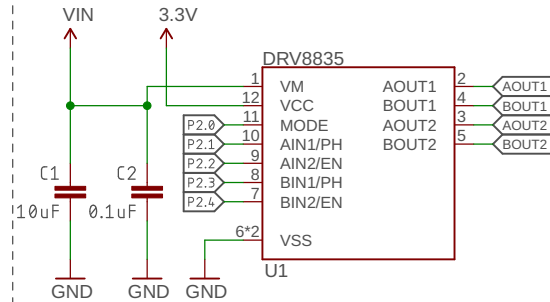
Config Jumpers

The default (all open) is the UART mode. A '1' indicates the jumper is bridged.

0000: UART
0001: SPI
0010: Slave device
      (User port unused)
0011: I2C addr 0x58
0100: I2C addr 0x59
0101: I2C addr 0x5A
0110: I2C addr 0x5B
      (...)
1100: I2C addr 0x61
1101: I2C addr 0x62
1110: I2C addr 0x63
1111: Reserved

MODEB3 — JP4 — GND
MODEB2 — JP3 — GND
MODEB1 — JP2 — GND
MODEB0 — JP1 — GND

PSoC I/O

U2G1 — CY8C4245LQI-483
MODEB0 22 P0.0
MODEB1 23 P0.1
MODEB2 24 P0.2
MODEB3 25 P0.3
SCL 26 P0.4
SDA 27 P0.5
28 P0.6
29 P0.7

U2G3 — CY8C4245LQI-483
P2.0 1 P2.0
P2.1 2 P2.1
P2.2 3 P2.2
P2.3 4 P2.3
P2.4 5 P2.4
6 P2.5
7 P2.6
8 P2.7

U2G4 — CY8C4245LQI-483
10 P3.0
11 P3.1
P3.2 12 P3.2
P3.3 13 P3.3
SLED 14 P3.4
15 P3.5
16 P3.6
17 P3.7

Test Points

3.3V

TP1
TP2 — P3.2
TP3 — P3.3
TP4 — RESET
TP5 — SDA
TP6 — SCL

P3.2 swd_io[0]
P3.3 swd_clk[0]

I2C Jumper

3.3V

Cut traces to remove pull up resistors

JP6

R2 4.7k, R3 4.7k

SDA
SCL

PSoC Power and Filtering

3.3V — R4 1
3.3V — R5 1

U2G6 — CY8C4245LQI-483
31 VCCD
32 VDDD
33 VDDA
40 VREF
30 XRES
34 VSSA
9*2 VSS

RESET

C3 1.0uF, C4 0.1uF, C5 1.0uF, C6 0.1uF, C7 1.0uF, C8 1.0uF, C9 0.1uF

GND GND GND GND GND GND GND GND
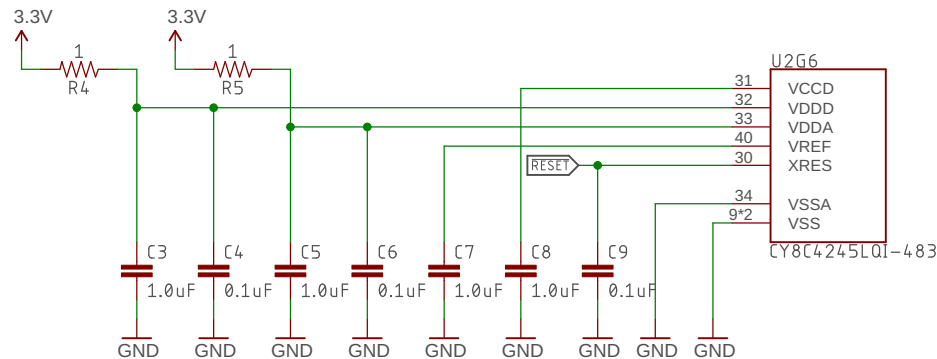
sparkfun qwiic — open hardware

TITLE: SparkFun_Qwiic_Motor_Driver

Design by: Alex Wende

REV: v10

Date: 8/26/2019 11:00 AM   Sheet: 1/1

G.8 Simple Impulse test code

```
1  /*
2   * This code sets an impulse to the motor at maximum speed.
3   *
4   * SCMD library by SparkFun used ...
        (https://github.com/sparkfun/SparkFun_Serial_Controlled_Motor_Driver_Arduino_Library)
5   * STM32 package from http://dan.drown.org/stm32duino/package_STM32duino_index.json
6   *
7  */
8
9  #include "SCMD.h"
10 #include "SCMD_config.h" // Contains #defines for common SCMD register names and ...
       values
11
12 SCMD DriverOne; // Driver object
13
14 void setup()
15 {
16   DriverOne.settings.commInterface = I2C_MODE; // Driver mode definition
17   DriverOne.settings.I2CAddress = 0x5D;        // Driver adress definition (0x5D ...
       by default)
18
19   while (DriverOne.begin() != 0xA9)
20   { // Wait for idle
21     Serial.println("ID Mismatch");
22     delay(200);
23   }
24   Serial.println("ID Match");
25
26   Serial.println("Waiting for enumeration"); // Wait for peripherals (enumeration)
27   while (DriverOne.ready() == false)
28     ;
29   Serial.println("Ready");
30
31   while (DriverOne.busy())
32     ; // Enables the driver
33   DriverOne.enable();
34 }
35
36 void loop()
37 {
38   DriverOne.setDrive(0, 0, 255); // Upload of values to motor
39   DriverOne.setDrive(1, 1, 0);
40 }
```

Code G.1 Motor driver Simple Impulse test. Source: Own.

Appendix H

CAD drawings

H.1   DC Motor

60°
10,00
Ø 1,00
Ø 6,40
Ø 5,40
Ø 2,25
60°

Ø 4,40
Ø 2,00
R0,10
Ø 2,25
0,50
1,00
2,00
5,00
SECTION A-A
9,00
10,00

A
6,20
1,00
R0,10
R1,00
9,04
R0,50
A
1,50

1,50

Ø 24,35
Ø 6,04
Ø 5,04

Polytechical University of Catalonia

DESIGNED BY	DATE:	TITLE
Yi Qiang Ji	03/04/2021	DC Motor
VERIF. Dr. David González Dr. Manel Lamich	MODEL: RF-300CA	
MANUF. MABUCHI	ADDITIONAL COMMENTS: Check manufacture's sheet for electrical specification. (see Appendices)	N.º DRAWING 01 - DC Motor A3

SCALE: 2:1   SHEET 1

H.2   Reaction Wheel

SECTION A-A

Ø60,00
Ø40,00
R2,50
Ø1,50
Ø1,50
Ø15,00
2,00
A
A
4,20
1,00
5,00
Ø1,00
2,50
Ø80,00
Ø15,00
Ø1,95

IF NOT INDICATED OTHERWISE: THE DIMENSIONS ARE EXPRESSED IN mm	This design is TIEG Research Group's property. It can't be reproduced or communicated without our written agreement.	DO NOT CHANGE THE SCALE		REVISION: Final
		Polytechical University of Catalonia		
DESIGNED BY Yi Qiang Ji	DATE: 20/05/2021	TITLE Reaction Wheel		
VERIF. Dr. David González Dr. Manel Lamich	MODEL: Own			
MANUF. TIEG & HP	ADDITIONAL COMMENTS:	N.º DRAWING 02 - Rection Wheel		A3
		SCALE: 1:1	SHEET 2	

H.3 Top support ring

15,68

$\emptyset$ 2,00

$\emptyset$ 27,35

$\emptyset$ 35,35

$\emptyset$ 9,00

5,04

3,04

A
A

$\emptyset$ 2,00

4,04

SECTION A-A

$\emptyset$ 2,00

All drill holes are M2x0.4

IF NOT INDICATED OTHERWISE: THE DIMENSIONS ARE EXPRESSED IN mm	This design is TIEG Research Group's property. It can't be reproduced or communicated without our written agreement.	DO NOT CHANGE THE SCALE		REVISION: Final
		Polytechical University of Catalonia		
DESIGNED BY Yi Qiang Ji	DATE: 20/05/2021	TITLE **Support Ring Top**		
VERIF. Dr. David González Dr. Manel Lamich	MODEL: Own			
MANUF. TIEG & HP	ADDITIONAL COMMENTS:	N.º DRAWING 03 - Support Ring Top		A3
		SCALE: 2:1	SHEET 3	

H.4   Bottom support ring

Ø 2,00

15,68

Ø 25,35

3,00

45°

A — A

4,04

SECTION A-A

3,40

Ø 12,00   Ø 35,35

All drill holes are M2x0.4

Polytechical University of Catalonia

DESIGNED BY Yi Qiang Ji	DATE: 20/05/2021	TITLE
VERIF. Dr. David González Dr. Manel Lamich	MODEL: Own	Support Ring Bottom
MANUF. TIEG & HP	ADDITIONAL COMMENTS:	N.º DRAWING 04 - Support Ring Bottom A3

SCALE: 2:1   SHEET 4

Appendix I

PCB connections drawings

Below is presented the schematic of the ADCS PCB board used in PLATHON's project.

I.1   ADCS PCB schematic

Designed by magnetorquer team [96].

BluePill1
BluePill_STM32F103C

21 PB12	GND 20
22 PB13_SCK2	GND 19
23 PB14_MISO2	VCC3V3 18
24 PB15_MOSI2	NRST 17
25 PA8	PB11_SDA2 16
26 PA9_TX1	PB10_SCL2 15
27 PA10_RX1	PB1 14
28 PA11_USB_D−	PB0 13
29 PA12_USBD+	PA7_MOSI1 12
30 PA15	PA6_MISO1 11
31 PB3	PA5_SCK1 10
32 PB4	PA4 9
33 PB5	PA3_RX2 8
34 PB6_SCL1	PA2_TX2 7
35 PB7_SDA1	PA1 6
36 PB8	PA0 5
37 PB9	PC15 4
38 5V_USB	PC14 3
39 GND	PC13_LED 2
40 VCC3V3	VBAT 1

PA13_SWDIO 42
PA14_SWCLK 41

TX RX
PB6 PB7

C1 1uF C2 100nF GND
3V3 GND

U2 GY−9250

1	VCC
2	GND
3	SCL/SCK
4	SDA/MOSI
×5	EDA
×6	ECL
7	AD0/MISO
×8	INT
9	NCS
10	FSYNC

C7 1uF C8 100nF
GND

Vin Drivers

J5 Conn_01x02 TP2 TestPoint
100nF C3 R1 0 TP1 TestPoint
C4 1uF 5V GND

J7 Conn_01x02 TP4 TestPoint
100nF C5 R2 0 TP3 TestPoint
C6 1uF 5V GND

Vout Drivers

Conn_01x02 Conn_01x02
MT X MT Y
Vout 2 J6
J2 Vout B2 B1 A2 A1

RI MT Z
Vout 1 J8
J4 Vout B2 B1 A2 A1

Comm.
12 NC 10 GND 8 MOSI 6 SS 4 RX 2 SDA
11 NC 9 GND 7 SCK 5 MISO 3 TX 1 SCL

Comm.1
Vcc 5 5V 3 GND 1 3V3
Vcc1 6 15V 4 GND 2 9,6V
5V GND

Módulo Bluetooth
HC−05 J1
1 STATE
2 RX TX
3 TX RX
4 GND
5 VCC
6 EN
100nF C9 C10 1uF GND 5V

Qwiic connector
Qwiic1 Qwiic
1 GND GND
2 3V3 3V3
3 PB7 SDA
4 PB6 SCL

I.2 PCB drawing

Designed by magnetorquer team [97].
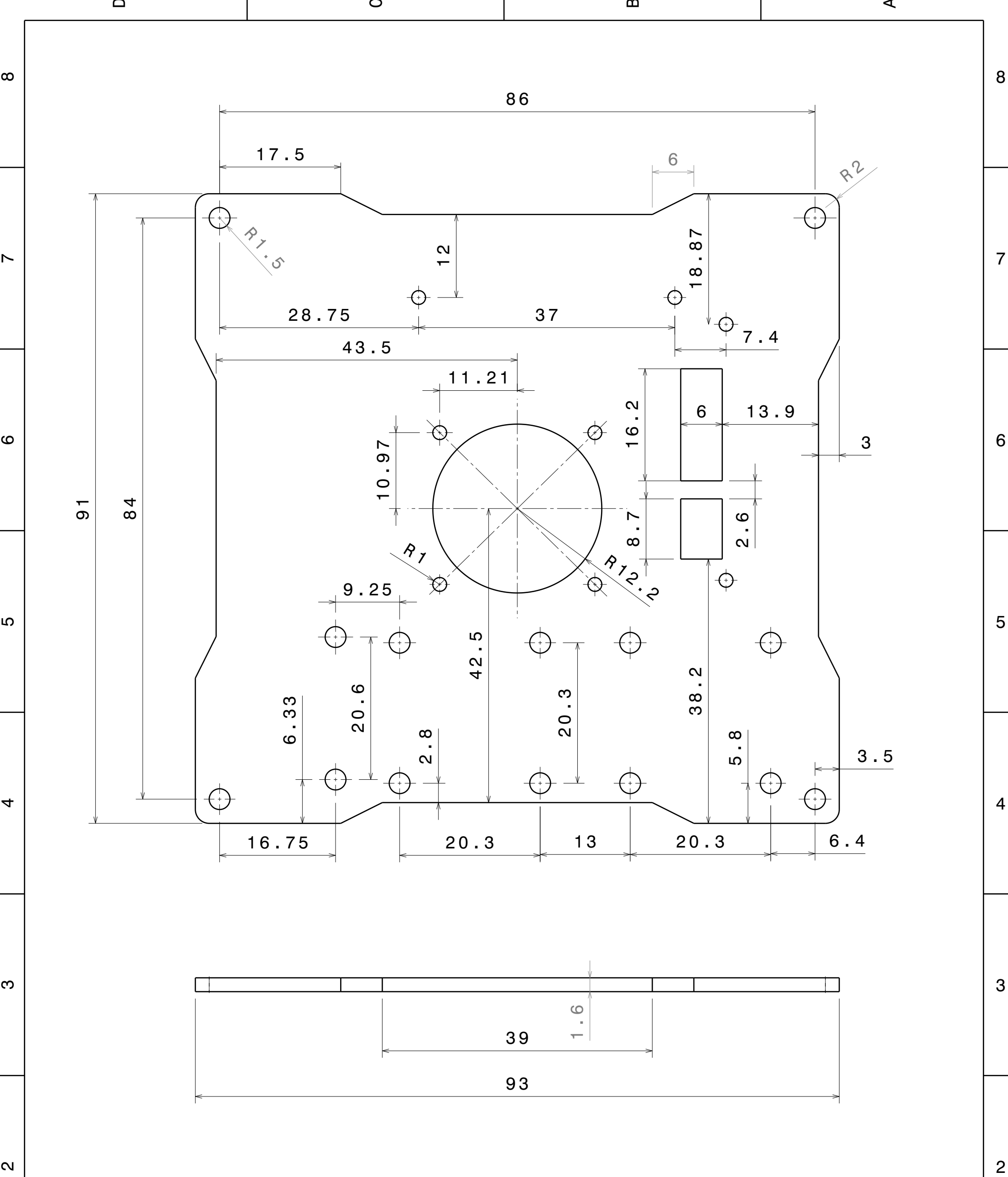
Dimensions

86
17.5
6
R2
R1.5
12
18.87
28.75
37
7.4
43.5
11.21
16.2
6
13.9
3
10.97
91
84
R1
R12.2
8.7
2.6
9.25
42.5
38.2
6.33
20.6
20.3
2.8
5.8
3.5
16.75
20.3
13
20.3
6.4
39
1.6
93

PLACA ADCS

DRAWING TITLE

PCB

DRAWN BY	DATE
Miguel Reurer	25/05/2021
CHECKED BY	DATE
Jordan Morales	31/05/2021
DESIGNED BY	DATE
Miguel Reurer	31/05/2021

SIZE A3

DRAWING NUMBER 1

Escala 2:1

SHEET 1/2

Taladros

Taladros	
8 agujeros	R1 mm
14 agujeros	R1.5 mm

I.3 Bottom support ring

Designed by magnetorquer team [97].
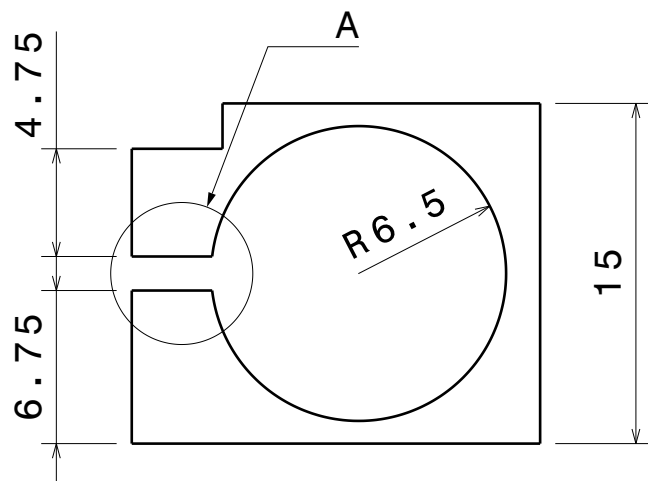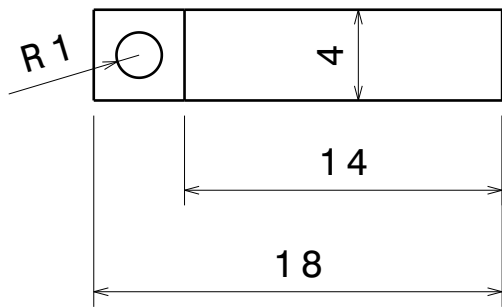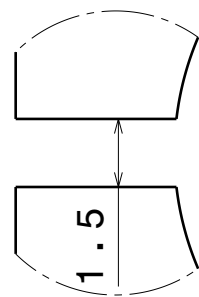
A (6:1)

PLACA ADCS

DRAWING TITLE
Soporte del magnetopar

DRAWN BY Miguel Reurer	DATE 25/05/2021		
CHECKED BY Jordan Morales	DATE 25/05/2021	SIZE A3	DRAWING NUMBER 2
DESIGNED BY Miguel Reurer	DATE 25/05/2021	Escala 3:1	SHEET 2/2

Appendix J

ADCS assembly diagrams

Diagram of the BNO055 and the Arduino Nano connections
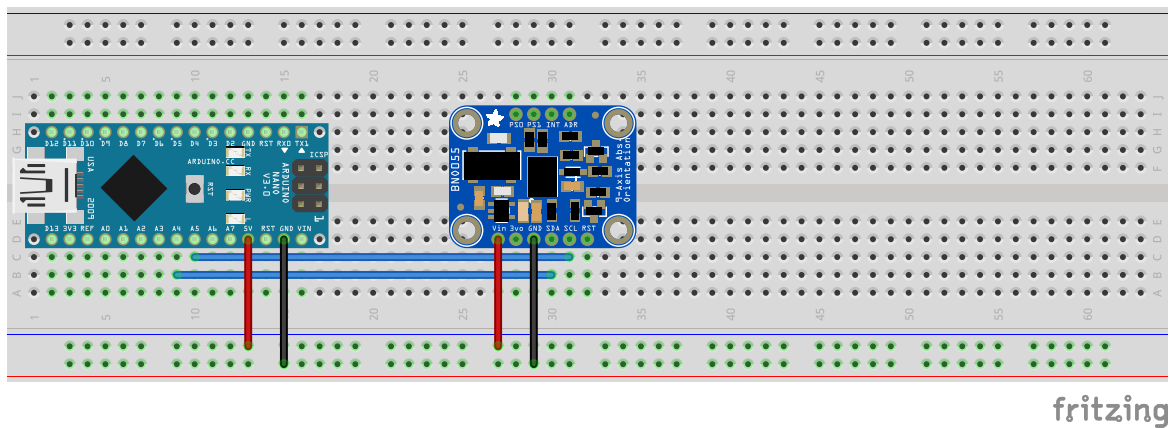
Figure J.1 Diagram of the BNO055 and the Arduino Nano connections.
Source: Own.

Diagram of the MPU9250 and the Arduino Nano connections
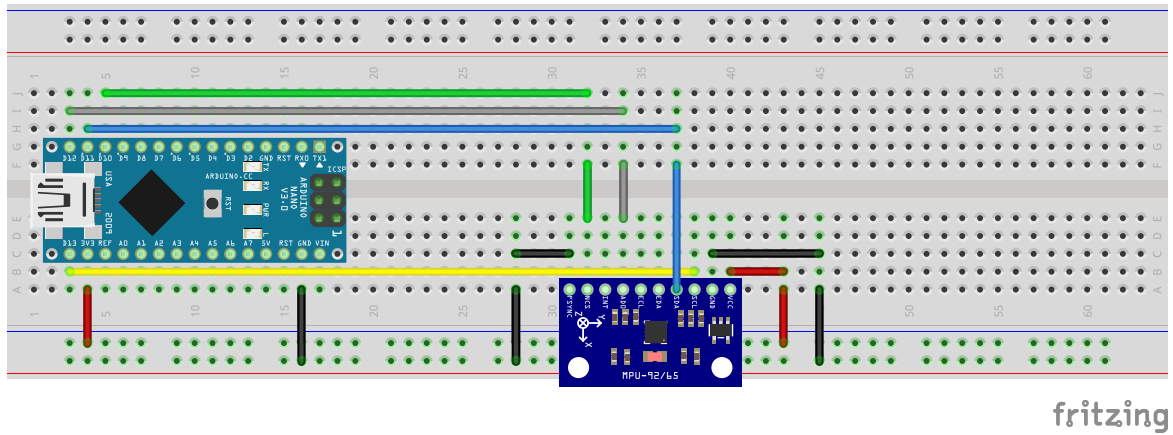
Figure J.2 Diagram of the MPU9250 and the Arduino Nano connections.
Source: Own.

Diagram of the MPU9250 and the STM3 Bluepill connections
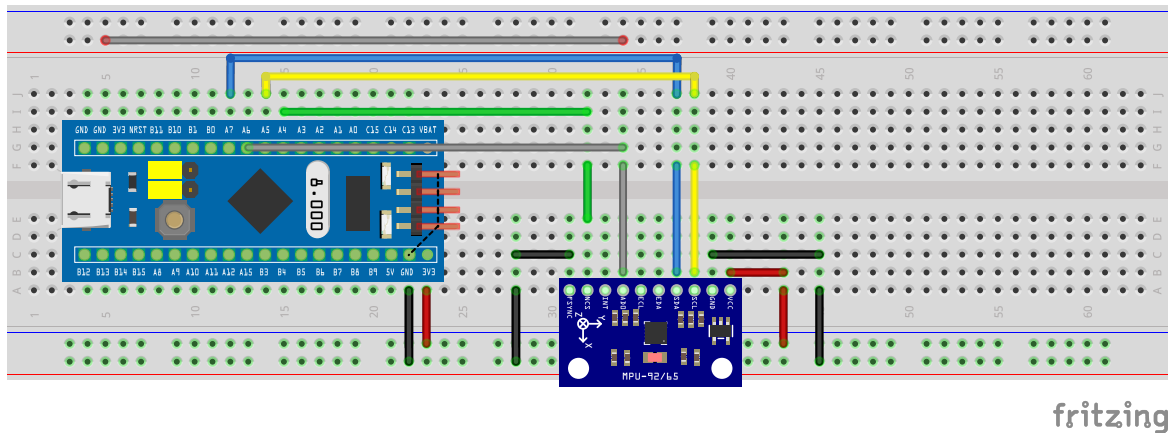
Figure J.3 Diagram of the MPU9250 and the STM3 Bluepill connections.
Source: Own.

Bluetooth setup diagram with STM32 Bluepill
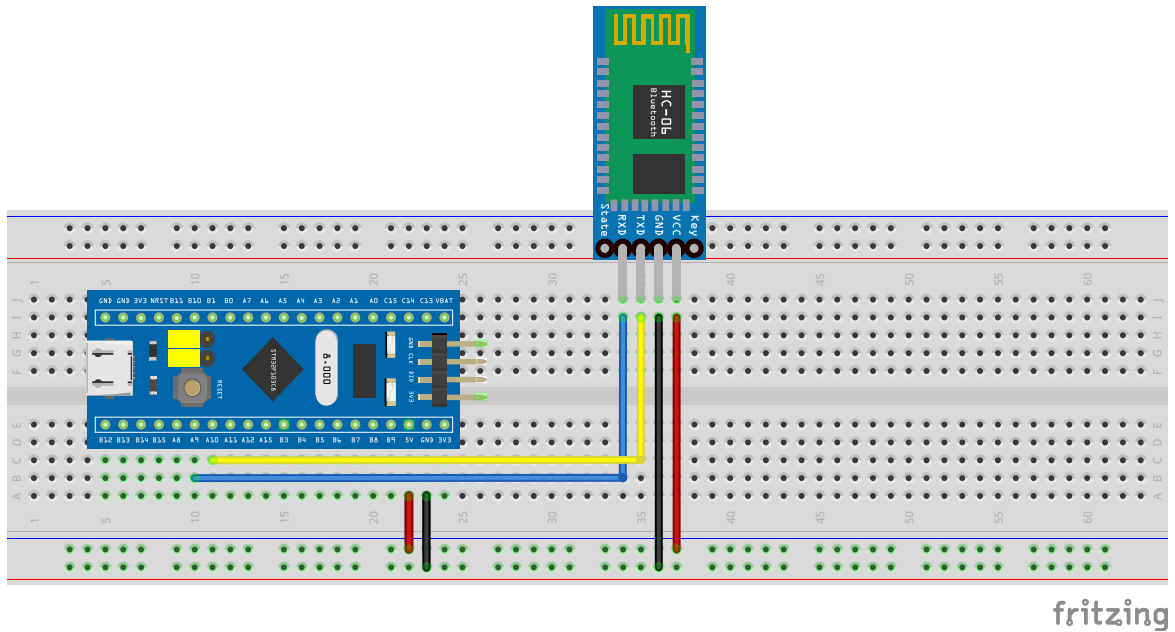
Figure J.4 Bluetooth setup diagram with STM32 Bluepill. Source: Own.

Diagram of the motor driver and motor and the STM32 Bluepill connections
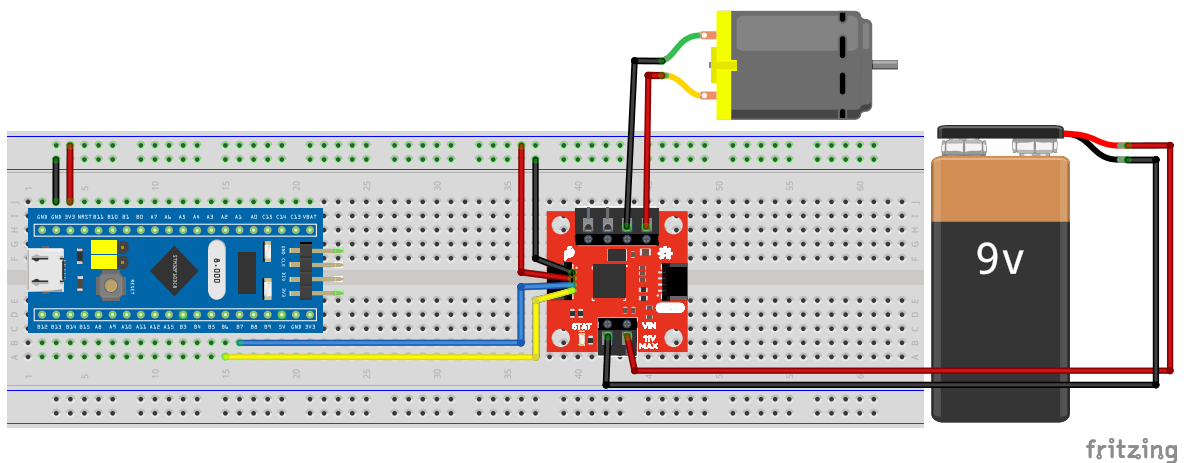
Figure J.5 Diagram of the motor driver and motor and the STM32 Bluepill connections. Source: Own.

Final assembly diagram
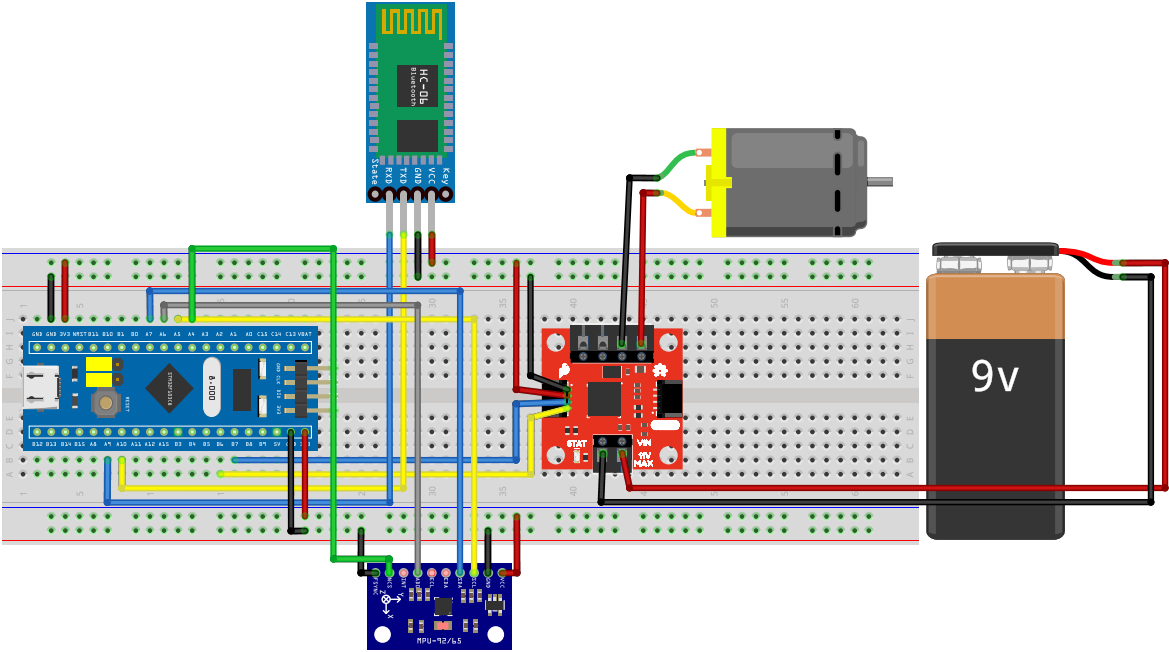
Figure J.6 Final assembly diagram. Source: Own.

Appendix K

Jacchia-Bowman JB2008 Atmospheric Density Model

The following code uses [28] to compute the values for density ρ and temperature T for different heights h. See Github [Github](#)

Jacchia-Bowman model uses several inputs such as EUV activity and geomagnetic indices, below is presented all the indices:

- MJD: Date and Time in Modified Julian Days and fraction $(\text{MJD} = \text{JD} - 2400000.5)$.

- Right Ascension of Sun [rad]

- Declination of Sun [rad]

- Right Ascension of Position [rad]

- Geocentric Latitude of Position [rad]

- Height of Position [km]

- F10: $F_{10.7}$ in $\left[10^{-22} \text{ W m}^{-2} \text{ Hz}^{-1}\right]$ (Tabular time 1.0 day earlier)

- F10B: $F_{10.7}$ solar flux average, 81 -day centered on the input time (Tabular time 1.0 day earlier)

- S10: EUV index $(26 - 34 \text{ nm})$ scaled to F10 (Tabular time 1.0 day earlier)

- S10B: EUV 81 -day ave. centered index (Tabular time 1.0 day earlier)

- XM10: MG2 index scaled to F10 (Tabular time 2.0 days earlier)

- XM10B: MG2 81 -day averaged centered index (Tabular time 2.0 days earlier)

- Y10: Solar X-Ray & Lya index scaled to F10 (Tabular time 5.0 days earlier)

- Y10B: Solar X-Ray & Lya 81-day average centered index (Tabular time 5.0 days earlier)

- dT_c : Temperature change computed from Dst geomagnetic storm index

```
1  %******************************************************************
2  %       Jacchia-Bowman 2008 Model Atmosphere
```

```
 3   %
 4   %       This is the CIRA "Integration Form" of a Jacchia Model.
 5   %       There are no tabular values of density.  Instead, the barometric
 6   %       equation and diffusion equation are integrated numerically using
 7   %       the Newton-Coates method to produce the density profile up to the
 8   %       input position.
 9   %
10   %       INPUT:
11   %
12   %            MJD    : Date and Time, in modified Julian Days
13   %                     and Fraction (MJD = JD-2400000.5)
14   %            SUN(1) : Right Ascension of Sun (radians)
15   %            SUN(2) : Declination of Sun (radians)
16   %            SAT(1) : Right Ascension of Position (radians)
17   %            SAT(2) : Geocentric Latitude of Position (radians)
18   %            SAT(3) : Height of Position (km)
19   %            F10    : 10.7-cm Solar Flux (1.0E-22*Watt/(M**2*Hertz))
20   %                     (Tabular time 1.0 day earlier)
21   %            F10B   : 10.7-cm Solar Flux, ave.
22   %                     81-day centered on the input time
23   %                     (Tabular time 1.0 day earlier)
24   %            S10    : EUV index (26-34 nm) scaled to F10
25   %                     (Tabular time 1.0 day earlier)
26   %            S10B   : EUV 81-day ave. centered index
27   %                     (Tabular time 1.0 day earlier)
28   %            XM10   : MG2 index scaled to F10
29   %                     (Tabular time 2.0 days earlier)
30   %            XM10B  : MG2 81-day ave. centered index
31   %                     (Tabular time 2.0 days earlier)
32   %            Y10    : Solar X-Ray & Lya index scaled to F10
33   %                     (Tabular time 5.0 days earlier)
34   %            Y10B   : Solar X-Ray & Lya 81-day ave. centered index
35   %                     (Tabular time 5.0 days earlier)
36   %            DSTDTC : Temperature change computed from Dst index
37   %
38   %       OUTPUT:
39   %
40   %            TEMP(1): Exospheric Temperature above Input Position (deg K)
41   %            TEMP(2): Temperature at Input Position (deg K)
42   %            RHO    : Total Mass-Desnity at Input Position (kg/m**3)
43   %
44   %
45   %       JB2008 Model Development: (Ref. 7)
46   %
47   %
48   %       A. Development of the JB2006 model:
49   %
50   %         1. Started with the CIRA72 model (Jacchia 71).
51   %
52   %         2. Converted to CIRA70 model replacing equations from Jacchia 70
53   %            model (Ref. 5)
54   %
55   %         3. Replaced Tc equation using new solar indices (Ref. 1 and 2)
56   %
57   %         4. Replaced semiannual equation with new global model based
58   %            on F10B (Ref. 1 and 3)
59   %
60   %         5. Added correction for local solar time and latitude errors
```

```
61  %           (Ref. 1)
62  %            Added smooth transition between altitude bands
63  %
64  %        6. Added high altitude ( z > 1500 km ) correction
65  %           (Ref. 1 and 4)
66  %
67  %        7. REV A of JB2006 - Oct 2006
68  %                 Smoothing of density corrections and scale height
69  %                 through different altitude bands in the latitude-
70  %                 local time correction subroutine DTSUB
71  %                 dTx correction replaced with dTc correction
72  %
73  %     B. Modification to develop JB2008 model:
74  %
75  %        1. Replaced Tc equation in JB2006 using new solar indices
76  %           (Ref. 7)
77  %
78  %        2. Replaced semiannual equation with new global model based
79  %            on F10B and S10B (Ref. 6)
80  %
81  %        3. Use dTc value based on Dst geomagnetic storm index
82  %           (This replaces ap use) (Ref. 7)
83  %
84  %
85  %         All equation references below refer to the original
86  %         Jacchia 1971 (CIRA 1972) model papers.
87  %
88  %
89  %     References:
90  %
91  %      1. Bowman, Bruce R., etc. : "A New Empirical Thermospheric
92  %         Density Model JB2006 Using New Solar Indices",
93  %         AIAA/AAS Astrodynamics Specialists Conference, Keystone, CO,
94  %         21-24 Aug 2006, (Paper AIAA 2006-6166).
95  %
96  %      2. Bowman, Bruce R., etc. : "Improvements in Modeling
97  %         Thermospheric Densities Using New EUV and FUV Solar Indices",
98  %         AAS/AIAA Space Flight Mechanics Meeting, Tampa, FL,
99  %         23-26 Jan 2006, (Paper AAS 06-237).
100 %
101 %      3. Bowman, Bruce R.: "The Semiannual Thermospheric Density
102 %         Variation From 1970 to 2002 Between 200-1100 km",
103 %         AAS/AIAA Space Flight Mechanics Meeting, Maui, HI,
104 %         8-12 Feb 2004, (Paper AAS 04-174).
105 %
106 %      4. Bowman, Bruce R.; "Atmospheric Density Variations at
107 %         1500 km to 4000 km Height Determined from Long Term
108 %         Orbit Perturbation Analysis", AAS/AIAA Space Flight
109 %         Mechanics Meeting, Santa Barbara, CA, 11-14 Feb 2001,
110 %         (Paper AAS 01-132).
111 %
112 %      5. Jacchia, Luigi G.; "New Static Models of the
113 %         Thermosphere and Exosphere with Empirical Temperature
114 %         Profiles", (Smithsonian Astrophysical Observatory
115 %         Special Report 313), 6 May 1970.
116 %
117 %      6. Bowman, Bruce R., etc. : "The Thermospheric Semiannual Density
118 %         Response to Solar EUV Heating," JASTP, 2008
```

203

```matlab
119  %
120  %       7. Bowman, Bruce R., etc. : "A New Empirical Thermospheric
121  %          Density Model JB2008 Using New Solar and Geomagnetic Indices",
122  %          AIAA/AAS 2008, COSPAR CIRA 2008 Model
123  %
124  % Last modified:   2018/01/27   M. Mahooti
125  %
126  %************************************************************************
127  function [TEMP,RHO] = JB2008(MJD,SUN,SAT,F10,F10B,S10,S10B,XM10,XM10B,Y10,Y10B,DSTDTC)
128
129  % The alpha are the thermal diffusion coefficients in Eq. (6)
130  ALPHA = [0.,0.,0.,0.,-0.38];
131
132  % AL10 is log(10.0)
133  AL10 = 2.3025851;
134
135  % The AMW are the molecular weights in order: N2, O2, O, Ar, He & H
136  AMW = [28.0134,31.9988,15.9994,39.9480,4.0026,1.00797];
137
138  % AVOGAD is Avogadro's number in mks units (molecules/kmol)
139  AVOGAD = 6.02257e26;
140
141  TWOPI  = 2*pi;
142  PIOV2  = 1.5707963;
143
144  % The FRAC are the assumed sea-level volume fractions in order:
145  % N2, O2, Ar, and He
146  FRAC = [0.78110,0.20955,9.3400e-3,1.2890e-5];
147
148  % RSTAR is the universal gas-constant in mks units (joules/K/kmol)
149  RSTAR = 8314.32;
150
151  % The R# are values used to establish height step sizes in
152  % the regimes 90km to 105km, 105km to 500km and 500km upward.
153  R1 = 0.010;
154  R2 = 0.025;
155  R3 = 0.075;
156
157  % The WT are weights for the Newton-Cotes Five-Point Quad. formula
158  WT =[0.311111111111111,1.422222222222222,0.533333333333333,...
159       1.422222222222222,0.311111111111111];
160
161  % The CHT are coefficients for high altitude density correction
162  CHT = [0.22,-0.20e-2,0.115e-2,-0.211e-5];
163  DEGRAD = pi/180.;
164
165  % Equation (14)
166  FN = (F10B/240)^(1/4);
167  if (FN>1)
168      FN = 1.0;
169  end
170  FSB = F10B*FN + S10B*(1-FN);
171  TSUBC = 392.4 + 3.227*FSB + 0.298*(F10-F10B) ...
172              + 2.259*(S10-S10B) + 0.312*(XM10-XM10B) ...
173              + 0.178*(Y10-Y10B);
174
175  % Equation (15)
176  ETA =   0.5 * abs(SAT(2) - SUN(2));
```

```matlab
177  THETA = 0.5 * abs(SAT(2) + SUN(2));
178
179  % Equation (16)
180  H = SAT(1) - SUN(1);
181  TAU = H - 0.64577182 + 0.10471976 * sin(H + 0.75049158);
182  GLAT  = SAT(2);
183  ZHT   = SAT(3);
184  GLST  = H + pi;
185  GLSTHR = (GLST/DEGRAD)*(24/360);
186
187  if (GLSTHR≥24)
188      GLSTHR = GLSTHR - 24;
189  end
190  if (GLSTHR< 0)
191      GLSTHR = GLSTHR + 24;
192  end
193
194  % Equation (17)
195  C = cos(ETA)^2.5;
196  S = sin(THETA)^2.5;
197
198  DF = S + (C - S) * abs(cos(0.5 * TAU))^3;
199  TSUBL = TSUBC * (1 + 0.31 * DF);
200
201  % Compute correction to dTc for local solar time and lat correction
202  DTCLST = DTSUB (F10,GLSTHR,GLAT,ZHT);
203
204  % Compute the local exospheric temperature.
205  % Add geomagnetic storm effect from input dTc value
206  TEMP(1) = TSUBL + DSTDTC;
207  TINF = TSUBL + DSTDTC + DTCLST;
208
209  % Equation (9)
210  TSUBX = 444.3807 + 0.02385 * TINF - 392.8292 * exp(-0.0021357 * TINF);
211
212  % Equation (11)
213  GSUBX = 0.054285714 * (TSUBX - 183);
214
215  % The TC array will be an argument in the call to
216  % XLOCAL, which evaluates Equation (10) or Equation (13)
217  TC(1) = TSUBX;
218  TC(2) = GSUBX;
219
220  % A AND GSUBX/A OF Equation (13)
221  TC(3) = (TINF - TSUBX)/PIOV2;
222  TC(4) = GSUBX/TC(3);
223
224  % Equation (5)
225  Z1 = 90;
226  Z2 = min(SAT(3),105);
227  AL = log(Z2/Z1);
228  N = floor(AL/R1) + 1;
229  ZR = exp(AL/N);
230  AMBAR1 = XAMBAR(Z1);
231  TLOC1 = XLOCAL(Z1,TC);
232  ZEND = Z1;
233  SUM2 = 0;
234  AIN = AMBAR1 * XGRAV(Z1)/TLOC1;
```

```
235
236  for I = 1:N
237      Z = ZEND;
238      ZEND = ZR * Z;
239      DZ = 0.25 * (ZEND-Z);
240      SUM1 = WT(1)*AIN;
241      for J = 2:5
242          Z = Z + DZ;
243          AMBAR2 = XAMBAR(Z);
244          TLOC2 = XLOCAL(Z,TC);
245          GRAVL = XGRAV(Z);
246          AIN = AMBAR2 * GRAVL/TLOC2;
247          SUM1 = SUM1 + WT(J) * AIN;
248      end
249      SUM2 = SUM2 + DZ * SUM1;
250  end
251
252  FACT1 = 1000/RSTAR;
253  RHO = 3.46e-6 * AMBAR2 * TLOC1 * exp(-FACT1*SUM2)/AMBAR1/TLOC2;
254
255  % Equation (2)
256  ANM = AVOGAD * RHO;
257  AN  = ANM/AMBAR2;
258
259  % Equation (3)
260  FACT2  = ANM/28.960;
261  ALN(1) = log(FRAC(1)*FACT2);
262  ALN(4) = log(FRAC(3)*FACT2);
263  ALN(5) = log(FRAC(4)*FACT2);
264
265  % Equation (4)
266  ALN(2) = log(FACT2 * (1 + FRAC(2)) - AN);
267  ALN(3) = log(2 * (AN - FACT2));
268
269  if (SAT(3) > 105)
270  else
271      TEMP(2) = TLOC2;
272      % Put in negligible hydrogen for use in DO-LOOP 13
273      ALN(6) = ALN(5) - 25;
274      % Equation (24)  - J70 Seasonal-Latitudinal Variation
275      TRASH = (MJD - 36204) / 365.2422;
276      CAPPHI = mod(TRASH,1);
277      DLRSL = 0.02 * (SAT(3) - 90) * exp(-0.045 * (SAT(3) - 90))...
278          * sign_(1,SAT(2)) * sin(TWOPI * CAPPHI+ 1.72)...
279          * sin(SAT(2))^2;
280      % Equation (23) - Computes the semiannual variation
281      DLRSA = 0;
282      if (Z<2000)
283          YRDAY = TMOUTD (MJD);
284          % Use new semiannual model
285          [FZZ,GTZ,DLRSA] = SEMIAN08 (YRDAY,ZHT,F10B,S10B,XM10B);
286          if (FZZ<0)
287              DLRSA = 0;
288          end
289      end
290
291      % Sum the ∆-log-rhos and apply to the number densities.
292      % In CIRA72 the following equation contains an actual sum,
```

```matlab
293      % namely DLR = AL10 * (DLRGM + DLRSA + DLRSL)
294      % However, for Jacchia 70, there is no DLRGM or DLRSA.
295      DLR = AL10 * (DLRSL + DLRSA);
296
297      for I = 1:6
298          ALN(I) = ALN(I) + DLR;
299      end
300
301      % Compute mass-density and mean-molecular-weight and
302      % convert number density logs from natural to common.
303      SUMN = 0;
304      SUMNM = 0;
305
306      for I = 1:6
307          AN = exp(ALN(I));
308          SUMN = SUMN + AN;
309          SUMNM = SUMNM + AN*AMW(I);
310          AL10N(I) = ALN(I)/AL10;
311      end
312
313      RHO = SUMNM/AVOGAD;
314
315      % Compute the high altitude exospheric density correction factor
316      FEX = 1;
317
318      if ((ZHT≥1000)&&(ZHT<1500))
319          ZETA   = (ZHT - 1000) * 0.002;
320          ZETA2  =  ZETA * ZETA;
321          ZETA3  =  ZETA * ZETA2;
322          F15C   = CHT(1) + CHT(2)*F10B + CHT(3)*1500 + CHT(4)*F10B*1500;
323          F15C_ZETA = (CHT(3) + CHT(4)*F10B) * 500;
324          FEX2   = 3 * F15C - F15C_ZETA - 3;
325          FEX3   = F15C_ZETA - 2 * F15C + 2;
326          FEX    = 1 + FEX2 * ZETA2 + FEX3 * ZETA3;
327      end
328
329      if (ZHT ≥ 1500)
330          FEX = CHT(1) + CHT(2)*F10B + CHT(3)*ZHT + CHT(4)*F10B*ZHT;
331      end
332
333      % Apply the exospheric density correction factor.
334      RHO = FEX * RHO;
335      return
336  end
337
338  % Equation (6)
339  Z3 = min(SAT(3),500);
340  AL = log(Z3/Z);
341  N = floor(AL/R2) + 1;
342  ZR = exp(AL/N);
343  SUM2 = 0;
344  AIN = GRAVL/TLOC2;
345
346  for I = 1:N
347      Z = ZEND;
348      ZEND = ZR * Z;
349      DZ = 0.25 * (ZEND - Z);
350      SUM1 = WT(1) * AIN;
```

```
351     for J = 2:5
352         Z = Z + DZ;
353         TLOC3 = XLOCAL(Z,TC);
354         GRAVL = XGRAV(Z);
355         AIN = GRAVL/TLOC3;
356         SUM1 = SUM1 + WT(J) * AIN;
357     end
358     SUM2 = SUM2 + DZ * SUM1;
359 end
360
361 Z4 = max(SAT(3),500);
362 AL = log(Z4/Z);
363 R = R2;
364
365 if (SAT(3) > 500)
366     R = R3;
367 end
368
369 N = floor(AL/R) + 1;
370 ZR = exp(AL/N);
371 SUM3 = 0;
372
373 for I=1:N
374     Z = ZEND;
375     ZEND = ZR * Z;
376     DZ = 0.25 * (ZEND - Z);
377     SUM1 = WT(1) * AIN;
378     for J = 2:5
379         Z = Z + DZ;
380         TLOC4 = XLOCAL(Z,TC);
381         GRAVL = XGRAV(Z);
382         AIN = GRAVL/TLOC4;
383         SUM1 = SUM1 + WT(J) * AIN;
384     end
385     SUM3 = SUM3 + DZ * SUM1;
386 end
387
388 if (SAT(3) > 500)
389     T500 = TLOC3;
390     TEMP(2) = TLOC4;
391     ALTR = log(TLOC4/TLOC2);
392     FACT2 = FACT1 * (SUM2 + SUM3);
393     HSIGN = -1;
394 else
395     T500 = TLOC4;
396     TEMP(2) = TLOC3;
397     ALTR = log(TLOC3/TLOC2);
398     FACT2 = FACT1 * SUM2;
399     HSIGN = 1;
400 end
401
402 for I = 1:5
403     ALN(I) = ALN(I) - (1 + ALPHA(I)) * ALTR - FACT2 * AMW(I);
404 end
405
406 % Equation (7) - Note that in CIRA72, AL10T5 = log10(T500)
407 AL10T5 = log10(TINF);
408 ALNH5 = (5.5 * AL10T5 - 39.40) * AL10T5 + 73.13;
```

```matlab
409   ALN(6) = AL10 * (ALNH5 + 6) + HSIGN * (log(TLOC4/TLOC3) + FACT1 * SUM3 * AMW(6));
410
411   % Equation (24)  - J70 Seasonal-Latitudinal Variation
412   TRASH = (MJD - 36204) / 365.2422;
413   CAPPHI = mod(TRASH,1);
414   DLRSL = 0.02 * (SAT(3) - 90) * exp(-0.045 * (SAT(3) - 90))...
415         * sign_(1,SAT(2)) * sin(TWOPI * CAPPHI+ 1.72)...
416         * sin(SAT(2))^2;
417
418   % Equation (23) - Computes the semiannual variation
419   DLRSA = 0;
420   if (Z<2000)
421       YRDAY = TMOUTD (MJD);
422       % Use new semiannual model
423       [FZZ,GTZ,DLRSA] = SEMIAN08 (YRDAY,ZHT,F10B,S10B,XM10B);
424       if (FZZ<0)
425           DLRSA = 0;
426       end
427   end
428
429   % Sum the Δ-log-rhos and apply to the number densities.
430   % In CIRA72 the following equation contains an actual sum,
431   % namely DLR = AL10 * (DLRGM + DLRSA + DLRSL)
432   % However, for Jacchia 70, there is no DLRGM or DLRSA.
433   DLR = AL10 * (DLRSL + DLRSA);
434
435   for I = 1:6
436       ALN(I) = ALN(I) + DLR;
437   end
438
439   % Compute mass-density and mean-molecular-weight and
440   % convert number density logs from natural to common.
441   SUMN = 0.;
442   SUMNM = 0.;
443
444   for I = 1:6
445       AN = exp(ALN(I));
446       SUMN = SUMN + AN;
447       SUMNM = SUMNM + AN*AMW(I);
448       AL10N(I) = ALN(I)/AL10;
449   end
450
451   RHO = SUMNM/AVOGAD;
452
453   % Compute the high altitude exospheric density correction factor
454   FEX = 1;
455
456   if ((ZHT≥1000)&&(ZHT<1500))
457       ZETA   = (ZHT - 1000) * 0.002;
458       ZETA2  =  ZETA * ZETA;
459       ZETA3  =  ZETA * ZETA2;
460       F15C   = CHT(1) + CHT(2)*F10B + CHT(3)*1500 + CHT(4)*F10B*1500;
461       F15C_ZETA = (CHT(3) + CHT(4)*F10B) * 500;
462       FEX2   = 3 * F15C - F15C_ZETA - 3;
463       FEX3   = F15C_ZETA - 2 * F15C + 2;
464       FEX    = 1 + FEX2 * ZETA2 + FEX3 * ZETA3;
465   end
466
```

```matlab
467   if (ZHT ≥ 1500)
468       FEX = CHT(1) + CHT(2)*F10B + CHT(3)*ZHT + CHT(4)*F10B*ZHT;
469   end
470
471   % Apply the exospheric density correction factor.
472   RHO = FEX * RHO;
473
474   end
475
476   %************************************************************************
477   function XAMBAR2 = XAMBAR(Z)
478   % Evaluates Equation (1)
479
480   C = [28.15204,-8.5586e-2,+1.2840e-4,-1.0056e-5,-1.0210e-5,+1.5044e-6,+9.9826e-8];
481   DZ = Z - 100;
482   AMB = C(7);
483
484   for I = 1:6
485       J = 7-I;
486       AMB = DZ * AMB + C(J);
487   end
488
489   XAMBAR2 = AMB;
490
491   end
492
493   %************************************************************************
494   function XGRAV2 = XGRAV(Z)
495   % Evaluates Equation (8)
496   XGRAV2 = 9.80665/(1 + Z/6356.766)^2;
497
498   end
499   %************************************************************************
500   function XLOCAL2 = XLOCAL(Z,TC)
501   % Evaluates Equation (10) or Equation (13), depending on Z
502   DZ = Z - 125;
503   if (DZ > 0)
504       XLOCAL2 = TC(1) + TC(3) * atan(TC(4)*DZ*(1 + 4.5e-6*DZ^2.5));
505       return
506   end
507       XLOCAL2 = ((-9.8204695e-6 * DZ - 7.3039742e-4) * DZ^2 + 1) * DZ * TC(2) + TC(1);
508   end
509   %************************************************************************
510   function DTC = DTSUB (F10,XLST,XLAT,ZHT)
511   %
512   % COMPUTE dTc correction for Jacchia-Bowman model
513   %
514   %    Calling Args:
515   %    ------------
516   %    F10       = (I)   F10 FLUX
517   %    XLST      = (I)   LOCAL SOLAR TIME (HOURS 0-23.999)
518   %    XLAT      = (I)   XLAT = SAT LAT (RAD)
519   %    ZHT       = (I)   ZHT = HEIGHT (KM)
520   %    DTC       = (O)   dTc correction
521   %
522   B = [-0.457512297e1, -0.512114909e1, -0.693003609e2,...
523        0.203716701e3,  0.703316291e3, -0.194349234e4,...
524        0.110651308e4, -0.174378996e3,  0.188594601e4,...
```

```
525      -0.709371517e4,  0.922454523e4, -0.384508073e4,...
526      -0.645841789e1,  0.409703319e2, -0.482006560e3,...
527       0.181870931e4, -0.237389204e4,  0.996703815e3,...
528       0.361416936e2];
529
530  C = [-0.155986211e2, -0.512114909e1, -0.693003609e2,...
531       0.203716701e3,  0.703316291e3, -0.194349234e4,...
532       0.110651308e4, -0.220835117e3,  0.143256989e4,...
533      -0.318481844e4,  0.328981513e4, -0.135332119e4,...
534       0.199956489e2, -0.127093998e2,  0.212825156e2,...
535      -0.275555432e1,  0.110234982e2,  0.148881951e3,...
536      -0.751640284e3,  0.637876542e3,  0.127093998e2,...
537      -0.212825156e2,  0.275555432e1];
538
539  DTC = 0;
540  tx  = XLST/24;
541  ycs = cos(XLAT);
542  F   = (F10 - 100)/100;
543
544  % calculates dTc
545  if (ZHT≥120 && ZHT≤200)
546      H = (ZHT - 200)/50;
547      DTC200 = C(17)            + C(18)*tx*ycs      + C(19)*tx^2*ycs...
548          + C(20)*tx^3*ycs   + C(21)*F*ycs        + C(22)*tx*F*ycs...
549          + C(23)*tx^2*F*ycs;
550      sum = C(1) + B(2)*F + C(3)*tx*F    + C(4)*tx^2*F...
551          + C(5)*tx^3*F    + C(6)*tx^4*F    + C(7)*tx^5*F...
552          + C(8)*tx*ycs      + C(9)*tx^2*ycs  + C(10)*tx^3*ycs...
553          + C(11)*tx^4*ycs + C(12)*tx^5*ycs + C(13)*ycs...
554          + C(14)*F*ycs      + C(15)*tx*F*ycs  + C(16)*tx^2*F*ycs;
555      DTC200DZ = sum;
556      CC  = 3*DTC200 - DTC200DZ;
557      DD  = DTC200 - CC;
558      ZP  = (ZHT-120)/80;
559      DTC = CC*ZP*ZP + DD*ZP*ZP*ZP;
560  end
561
562  if (ZHT>200 && ZHT≤240)
563      H = (ZHT - 200)/50;
564      sum = C(1)*H + B(2)*F*H + C(3)*tx*F*H     + C(4)*tx^2*F*H...
565          + C(5)*tx^3*F*H    + C(6)*tx^4*F*H    + C(7)*tx^5*F*H...
566          + C(8)*tx*ycs*H     + C(9)*tx^2*ycs*H  + C(10)*tx^3*ycs*H...
567          + C(11)*tx^4*ycs*H + C(12)*tx^5*ycs*H + C(13)*ycs*H...
568          + C(14)*F*ycs*H     + C(15)*tx*F*ycs*H  + C(16)*tx^2*F*ycs*H...
569          + C(17)            + C(18)*tx*ycs      + C(19)*tx^2*ycs...
570          + C(20)*tx^3*ycs   + C(21)*F*ycs       + C(22)*tx*F*ycs...
571          + C(23)*tx^2*F*ycs;
572      DTC = sum;
573  end
574
575  if (ZHT>240 && ZHT≤300)
576      H = 40/50;
577      sum = C(1)*H + B(2)*F*H + C(3)*tx*F*H     + C(4)*tx^2*F*H...
578          + C(5)*tx^3*F*H    + C(6)*tx^4*F*H    + C(7)*tx^5*F*H...
579          + C(8)*tx*ycs*H     + C(9)*tx^2*ycs*H  + C(10)*tx^3*ycs*H...
580          + C(11)*tx^4*ycs*H + C(12)*tx^5*ycs*H + C(13)*ycs*H...
581          + C(14)*F*ycs*H     + C(15)*tx*F*ycs*H  + C(16)*tx^2*F*ycs*H...
582          + C(17)            + C(18)*tx*ycs      + C(19)*tx^2*ycs...
```

```
583          + C(20)*tx^3*ycs   + C(21)*F*ycs      + C(22)*tx*F*ycs...
584          + C(23)*tx^2*F*ycs;
585     AA = sum;
586     BB = C(1)  + B(2)*F   + C(3)*tx*F       + C(4)*tx^2*F...
587          + C(5)*tx^3*F   + C(6)*tx^4*F    + C(7)*tx^5*F...
588          + C(8)*tx*ycs      + C(9)*tx^2*ycs  + C(10)*tx^3*ycs...
589          + C(11)*tx^4*ycs + C(12)*tx^5*ycs + C(13)*ycs...
590          + C(14)*F*ycs      + C(15)*tx*F*ycs  + C(16)*tx^2*F*ycs;
591     H   = 300/100;
592     sum = B(1)     + B(2)*F  + B(3)*tx*F        + B(4)*tx^2*F...
593          + B(5)*tx^3*F      + B(6)*tx^4*F       + B(7)*tx^5*F...
594          + B(8)*tx*ycs       + B(9)*tx^2*ycs    + B(10)*tx^3*ycs...
595          + B(11)*tx^4*ycs  + B(12)*tx^5*ycs   + B(13)*H*ycs...
596          + B(14)*tx*H*ycs    + B(15)*tx^2*H*ycs + B(16)*tx^3*H*ycs...
597          + B(17)*tx^4*H*ycs + B(18)*tx^5*H*ycs + B(19)*ycs;
598     DTC300 = sum;
599     sum = B(13)*ycs...
600          + B(14)*tx*ycs    + B(15)*tx^2*ycs + B(16)*tx^3*ycs...
601          + B(17)*tx^4*ycs + B(18)*tx^5*ycs;
602     DTC300DZ = sum;
603     CC = 3.*DTC300 - DTC300DZ - 3.*AA - 2.*BB;
604     DD = DTC300 - AA - BB - CC;
605     ZP  = (ZHT-240)/60;
606     DTC = AA + BB*ZP + CC*ZP*ZP + DD*ZP*ZP*ZP;
607 end
608
609 if (ZHT>300 && ZHT≤600)
610     H   = ZHT/100;
611     sum = B(1)     + B(2)*F  + B(3)*tx*F        + B(4)*tx^2*F...
612          + B(5)*tx^3*F      + B(6)*tx^4*F       + B(7)*tx^5*F...
613          + B(8)*tx*ycs       + B(9)*tx^2*ycs    + B(10)*tx^3*ycs...
614          + B(11)*tx^4*ycs  + B(12)*tx^5*ycs   + B(13)*H*ycs...
615          + B(14)*tx*H*ycs    + B(15)*tx^2*H*ycs + B(16)*tx^3*H*ycs...
616          + B(17)*tx^4*H*ycs + B(18)*tx^5*H*ycs + B(19)*ycs;
617         DTC = sum;
618 end
619
620 if (ZHT>600 && ZHT≤800)
621     ZP = (ZHT - 600)/100;
622     HP = 600./100.;
623     AA  = B(1)     + B(2)*F  + B(3)*tx*F        + B(4)*tx^2*F...
624          + B(5)*tx^3*F      + B(6)*tx^4*F       + B(7)*tx^5*F...
625          + B(8)*tx*ycs       + B(9)*tx^2*ycs    + B(10)*tx^3*ycs...
626          + B(11)*tx^4*ycs  + B(12)*tx^5*ycs   + B(13)*HP*ycs...
627          + B(14)*tx*HP*ycs   + B(15)*tx^2*HP*ycs+ B(16)*tx^3*HP*ycs...
628          + B(17)*tx^4*HP*ycs + B(18)*tx^5*HP*ycs + B(19)*ycs;
629     BB  = B(13)*ycs...
630          + B(14)*tx*ycs    + B(15)*tx^2*ycs + B(16)*tx^3*ycs...
631          + B(17)*tx^4*ycs + B(18)*tx^5*ycs;
632     CC  = -(3*AA+4*BB)/4;
633     DD  = (AA+BB)/4;
634     DTC = AA + BB*ZP + CC*ZP*ZP + DD*ZP*ZP*ZP;
635 end
636
637 end
638
639 %************************************************************
640 function [FZZ,GTZ,DRLOG] = SEMIAN08(DAY,HT,F10B,S10B,XM10B)
```

```
641  %
642  % COMPUTE SEMIANNUAL VARIATION (DELTA LOG RHO)
643  % INPUT DAY, HEIGHT, F10BAR
644  %      025. 650.  150.
645  % OUTPUT FUNCTIONS FZ, GT, AND DEL LOG RHO VALUE
646  %
647  % DAY     (I)   DAY OF YEAR
648  % HT      (I)   HEIGHT (KM)
649  % F10BAR  (I)   AVE 81-DAY CENTERED F10
650  % FZZ     (O)   SEMIANNUAL AMPLITUDE
651  % GTZ     (O)   SEMIANNUAL PHASE FUNCTION
652  % DRLOG   (O)   DELTA LOG RHO
653
654  TWOPI = 2*pi;
655
656  % FZ GLOBAL MODEL VALUES
657  % 1997-2006 FIT:
658  FZM = [0.2689,-0.1176e-1, 0.2782e-1, ...
659        -0.2782e-1, 0.3470e-3];
660
661  % GT GLOBAL MODEL VALUES
662  % 1997-2006 FIT:
663  GTM = [-0.3633, 0.8506e-1, 0.2401,-0.1897, ...
664        -0.2554,-0.1790e-1, 0.5650e-3,-0.6407e-3, ...
665        -0.3418e-2,-0.1252e-2];
666
667  % COMPUTE NEW 81-DAY CENTERED SOLAR INDEX FOR FZ
668  FSMB = F10B - 0.70*S10B - 0.04*XM10B;
669  HTZ = HT/1000;
670
671  FZZ = FZM(1) + FZM(2)*FSMB  + FZM(3)*FSMB*HTZ ...
672      + FZM(4)*FSMB*HTZ^2     + FZM(5)*FSMB^2*HTZ;
673
674  % COMPUTE DAILY 81-DAY CENTERED SOLAR INDEX FOR GT
675  FSMB = F10B - 0.75*S10B - 0.37*XM10B;
676
677  TAU   = (DAY-1)/365;
678  SIN1P = sin(TWOPI*TAU);
679  COS1P = cos(TWOPI*TAU);
680  SIN2P = sin(2*TWOPI*TAU);
681  COS2P = cos(2*TWOPI*TAU);
682
683  GTZ = GTM(1) + GTM(2)*SIN1P + GTM(3)*COS1P ...
684      + GTM(4)*SIN2P + GTM(5)*COS2P ...
685      + GTM(6)*FSMB ...
686      + GTM(7)*FSMB*SIN1P + GTM( 8)*FSMB*COS1P ...
687      + GTM(9)*FSMB*SIN2P + GTM(10)*FSMB*COS2P;
688
689  if (FZZ<1e-6)
690      FZZ = 1e-6;
691  end
692
693  DRLOG = FZZ*GTZ;
694
695  end
696
697  %*****************************************************************
698  function doy = TMOUTD(MJD)
```

```
699
700  [year,month,day,hr,min,sec] = invjday(MJD+2400000.5);
701  doy = finddays(year,month,day,hr,min,sec);
702
703  end
```

Code K.1 Jacchia-Bowman JB2008 Atmospheric Density Model. Source: Matlab and Jacchia-Bowman [98] [28].

```
1   %-------------------------------------------------------------------------%
2   % Temperature and density plots using Jacchia-Bowman 2008 Model Atmosphere
3
4   % Date: 17/04/2021
5   % Author/s: Yi Qiang Ji Zhang
6
7   %-------------------------------------------------------------------------%
8
9   clc;
10  close all;
11  clear all;
12  format long g
13
14  % Set interpreter to latex
15  set(groot,'defaultAxesTickLabelInterpreter','latex');
16  set(groot,'defaulttextinterpreter','latex');
17  set(groot,'defaultLegendInterpreter','latex');
18
19  global const PC
20
21  SAT_Const
22  constants
23  load DE430Coeff.mat
24  PC = DE430Coeff;
25
26  % read Earth orientation parameters
27  fid = fopen('eop19620101.txt','r');
28  %  ...
       ----------------------------------------------------------------------------------------------
29  % | Date    MJD      x          y        UT1-UTC     LOD      dPsi     dEpsilon    dX ...
            dY     DAT
30  % |(0h UTC)           "          "         s          s        "         "           " ...
             "      s
31  %  ...
       ----------------------------------------------------------------------------------------------
32  eopdata = fscanf(fid,'%i %d %d %i %f %f %f %f %f %f %f %f %i',[13 inf]);
33  fclose(fid);
34
35  % read space weather data
36  fid = fopen('SOLFSMY.txt','r');
37  %  ----------------------------------------------------------------------
38  % | YYYY DDD   JulianDay  F10   F81c   S10   S81c   M10   M81c  Y10   Y81c
39  %  ----------------------------------------------------------------------
40  SOLdata = fscanf(fid,'%d %d %f %f %f %f %f %f %f %f %f',[11 inf]);
41  fclose(fid);
42
43
```

```matlab
44  %% Density at h=400km for different years
45
46  % Dates vector
47  dates = 1999:1:2015;
48
49  % Temperature vector
50  temp_dates = zeros(1,length(dates));
51
52  % Density vector
53  rho_dates = zeros(1,length(dates));
54
55  % Height vector
56  height = [400 500 600];
57
58  % Strings (for automatic plotting)
59  str = strings([1,length(height)]);
60
61  % Loop for each height
62  for k=1:1:length(height)
63      % Loop for each date
64      for j=1:1:length(dates)
65          year = dates(j);
66          doy = 200;
67          [month,day,hour,minute,sec] = days2mdh(year,doy);
68          MJD = Mjday(year,month,day,hour,minute,sec);
69
70          % READ SOLAR INDICES
71          % USE 1 DAY LAG FOR F10 AND S10 FOR JB2008
72          JD = floor(MJD-1+2400000.5);
73          i = find(JD==SOLdata(3,:),1,'first');
74          SOL = SOLdata(:,i);
75          F10 = SOL(4);
76          F10B = SOL(5);
77          S10 = SOL(6);
78          S10B = SOL(7);
79
80          % USE 2 DAY LAG FOR M10 FOR JB2008
81          SOL = SOLdata(:,i-1);
82          XM10 = SOL(8);
83          XM10B = SOL(9);
84
85          % USE 5 DAY LAG FOR Y10 FOR JB2008
86          SOL = SOLdata(:,i-4);
87          Y10 = SOL(10);
88          Y10B = SOL(11);
89
90          % READ GEOMAGNETIC STORM DTC VALUE
91          fid = fopen('DTCFILE.txt','r');
92          %  ------------------------------------------------------------------
93          % | YYYY DDD   DTC1 to DTC24
94          %  ------------------------------------------------------------------
95          DTCdata = fscanf(fid,'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d ...
     %d %d %d %d %d %d %d',[26 inf]);
96          fclose(fid);
97
98          doy = finddays(year,month,day,hour,minute,sec);
99          i = find(year==DTCdata(1,:) & floor(doy)==DTCdata(2,:),1,'first');
100         DTC = DTCdata(:,i);
```

```matlab
101            ii = floor(hour)+3;
102            DSTDTC = DTC(ii);
103
104            % CONVERT POINT OF INTEREST LOCATION (RADIANS AND KM)
105            % CONVERT LONGITUDE TO RA
106            [x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC] = ...
       IERS(eopdata,MJD,'l');
107            [UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC] = timediff(UT1_UTC,TAI_UTC);
108            [DJMJD0, DATE] = iauCal2jd(year, month, day);
109            TIME = (60*(60*hour+minute)+sec)/86400;
110            UTC = DATE+TIME;
111            TT = UTC+TT_UTC/86400;
112            TUT = TIME+UT1_UTC/86400;
113            UT1 = DATE+TUT;
114            GWRAS = iauGmst06(DJMJD0, UT1, DJMJD0, TT);
115            XLON = 60*const.Rad;
116            SAT(1) = mod(GWRAS + XLON, 2*pi);
117            SAT(2) = -70*const.Rad;
118            SAT(3) = height(k);
119
120            % SET Sun's right ascension and declination (RADIANS)
121            % Difference between ephemeris time and universal time
122            % JD = MJD_UTC+2400000.5;
123            % [year, month, day, hour, minute, sec] = invjday(JD);
124            % days = finddays(year, month, day, hour, minute, sec);
125            % ET_UT = ETminUT(year+days/365.25);
126            % MJD_ET = MJD_UTC+ET_UT/86400;
127            % [r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus, ...
128            %  r_Neptune,r_Pluto,r_Moon,r_Sun,r_SunSSB] = JPL_Eph_DE430(MJD_ET);
129
130            MJD_TDB = Mjday_TDB(TT);
131            [r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus, ...
132             r_Neptune,r_Pluto,r_Moon,r_Sun,r_SunSSB] = JPL_Eph_DE430(MJD_TDB);
133            ra_Sun  = atan2(r_Sun(2), r_Sun(1));
134            dec_Sun = atan2(r_Sun(3), sqrt(r_Sun(1)^2+r_Sun(2)^2));
135            SUN(1)  = ra_Sun;
136            SUN(2)  = dec_Sun;
137
138            % COMPUTE DENSITY KG/M3 RHO
139            [TEMP,RHO] = JB2008(MJD,SUN,SAT,F10,F10B,S10,S10B,XM10,XM10B,Y10,Y10B,DSTDTC);
140
141            % Save temperature
142            temp_dates(k,j) = TEMP(2);
143
144            % Save density in density vector
145            rho_dates(k,j) = RHO;
146
147        end
148        % Concatenate strings (for automatic plotting)
149        str(k) = {strcat('$h$ = ' , '\,', num2str(SAT(3)), '\,', '$\mathrm{km}$')};
150  end
151
152
153
154  % Plots
155
156  title1 = strcat('\textbf{Temperature $T$ }');
157  title2 = strcat('\textbf{Density $\rho$ }');
```

```matlab
158
159  plot_pdf1 = figure;
160  plot(dates,temp_dates);
161  grid on;
162  grid minor;
163  title(title1);
164  legend(str(:));
165  xlabel('Date [year]');
166  ylabel('Temperature [$^\circ K$]');
167
168  plot_pdf2 = figure;
169  plot(dates,rho_dates);
170  grid on;
171  grid minor;
172  title(title2);
173  legend(str(:));
174  xlabel('Date [year]');
175  ylabel('Density [$\mathrm{kg}/\mathrm{m^3}$]');
176
177
178  % Save plots in .pdf and .png
179
180  % % Figure 1
181  % set(plot_pdf1, 'Units', 'Centimeters');
182  % pos = get(plot_pdf1, 'Position');
183  % set(plot_pdf1, 'PaperPositionMode', 'Auto', 'PaperUnits', 'Centimeters', ...
184  %          'PaperSize', [pos(3), pos(4)]);
185  % print(plot_pdf1, 'JB2008_temperature_vs_height.pdf', '-dpdf', '-r0');
186  %
187  % % Save png
188  % print(plot_pdf1,'JB2008_temperature_vs_height.png','-dpng','-r1000');
189  %
190  % % Figure 2
191  % set(plot_pdf2, 'Units', 'Centimeters');
192  % pos = get(plot_pdf2, 'Position');
193  % set(plot_pdf2, 'PaperPositionMode', 'Auto', 'PaperUnits', 'Centimeters', ...
194  %          'PaperSize', [pos(3), pos(4)]);
195  % print(plot_pdf2, 'JB2008_density_vs_height.pdf', '-dpdf', '-r0');
196  %
197  % % Save png
198  % print(plot_pdf2,'JB2008_density_vs_height.png','-dpng','-r1000');
```

Code K.2 Jacchia-Bowman JB2008 Atmospheric Density Model. Source: Own.

```matlab
1   %-----------------------------------------------------------------------%
2   % Aerodynamic torque as a function of height, CD and Solar Acticity
3
4   % Date: 17/04/2021
5   % Author/s: Yi Qiang Ji Zhang
6
7   %-----------------------------------------------------------------------%
8
9   clc;
10  close all;
11  clear all;
12  format long g
```

```matlab
13
14  % Set interpreter to latex
15  set(groot,'defaultAxesTickLabelInterpreter','latex');
16  set(groot,'defaulttextinterpreter','latex');
17  set(groot,'defaultLegendInterpreter','latex');
18
19  global const PC
20
21  SAT_Const
22  constants
23  load DE430Coeff.mat
24  PC = DE430Coeff;
25
26  % read Earth orientation parameters
27  fid = fopen('eop19620101.txt','r');
28  %  ...
        ----------------------------------------------------------------------------------------------
29  % |  Date    MJD       x           y         UT1-UTC     LOD       dPsi      dEpsilon    dX  ...
            dY      DAT
30  % |(0h UTC)            "           "           s          s         "         "          "  ...
            "      s
31  %  ...
        ----------------------------------------------------------------------------------------------
32  eopdata = fscanf(fid,'%i %d %d %i %f %f %f %f %f %f %f %f %i',[13 inf]);
33  fclose(fid);
34
35  % read space weather data
36  fid = fopen('SOLFSMY.txt','r');
37  %  -----------------------------------------------------------------------
38  % | YYYY DDD   JulianDay  F10   F81c  S10   S81c  M10   M81c  Y10   Y81c
39  %  -----------------------------------------------------------------------
40  SOLdata = fscanf(fid,'%d %d %f %f %f %f %f %f %f %f %f',[11 inf]);
41  fclose(fid);
42
43
44  %% Aerodynamic torque as a function of height, CD and Solar Acticity
45
46  % Dates vector
47  dates = [2005];
48
49  % Temperature vector
50  temp_dates = zeros(1,length(dates));
51
52  % Density vector
53  rho_dates = zeros(1,length(dates));
54
55  % Height vector
56  height = 400:50:700; % [km]
57
58  % Strings (for automatic plotting)
59  str = strings([1,length(height)]);
60
61  % Drag coefficient
62  CD = 1:0.1:4;
63
64  % F10_7 Solar activity
65  F10_7 = 150; % Use [65 150 250 300]
66
```

```matlab
67  % Cubesat Wetted Area
68  Area = 0.01; % [m^2]
69
70  % Cubesat velocity
71  G = 6.67408e-11;
72  M_E = 5.9742e24;
73  R_E = 6371e3;
74  velocity = zeros(1,length(height));
75
76  % Location of the center of the atmospheric force
77  r_dA = 0.03; % [m]
78
79  % Exchange coeff
80  sigma_n = 0.8;
81  sigma_t = 0.8;
82
83  % Density data
84  rho_data = zeros(length(height), length(CD));
85
86
87  % Loop for each height
88  for k=1:1:length(height)
89      % Loop for every CD
90      for drag_counter=1:1:length(CD)
91          % Loop for each date
92          for j=1:1:length(dates)
93              year = dates(j);
94              doy = 200;
95              [month,day,hour,minute,sec] = days2mdh(year,doy);
96              MJD = Mjday(year,month,day,hour,minute,sec);
97
98              % READ SOLAR INDICES
99              % USE 1 DAY LAG FOR F10 AND S10 FOR JB2008
100             JD = floor(MJD-1+2400000.5);
101             i = find(JD==SOLdata(3,:),1,'first');
102             SOL = SOLdata(:,i);
103             F10 = F10_7;
104             F10B = 0;
105             S10 = SOL(6);
106             S10B = SOL(7);
107
108             % USE 2 DAY LAG FOR M10 FOR JB2008
109             SOL = SOLdata(:,i-1);
110             XM10 = SOL(8);
111             XM10B = SOL(9);
112
113             % USE 5 DAY LAG FOR Y10 FOR JB2008
114             SOL = SOLdata(:,i-4);
115             Y10 = SOL(10);
116             Y10B = SOL(11);
117
118             % READ GEOMAGNETIC STORM DTC VALUE
119             fid = fopen('DTCFILE.txt','r');
120             % -------------------------------------------------------------------
121             % | YYYY DDD   DTC1 to DTC24
122             % -------------------------------------------------------------------
123             DTCdata = fscanf(fid,'%d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d %d ...
        %d %d %d %d %d %d %d',[26 inf]);
```

```matlab
124                 fclose(fid);
125
126             doy = finddays(year,month,day,hour,minute,sec);
127             i = find(year==DTCdata(1,:) & floor(doy)==DTCdata(2,:),1,'first');
128             DTC = DTCdata(:,i);
129             ii = floor(hour)+3;
130             DSTDTC = DTC(ii);
131
132             % CONVERT POINT OF INTEREST LOCATION (RADIANS AND KM)
133             % CONVERT LONGITUDE TO RA
134             [x_pole,y_pole,UT1_UTC,LOD,dpsi,deps,dx_pole,dy_pole,TAI_UTC] = ...
        IERS(eopdata,MJD,'l');
135             [UT1_TAI,UTC_GPS,UT1_GPS,TT_UTC,GPS_UTC] = timediff(UT1_UTC,TAI_UTC);
136             [DJMJD0, DATE] = iauCal2jd(year, month, day);
137             TIME = (60*(60*hour+minute)+sec)/86400;
138             UTC = DATE+TIME;
139             TT = UTC+TT_UTC/86400;
140             TUT = TIME+UT1_UTC/86400;
141             UT1 = DATE+TUT;
142             GWRAS = iauGmst06(DJMJD0, UT1, DJMJD0, TT);
143             XLON = 60*const.Rad;
144             SAT(1) = mod(GWRAS + XLON, 2*pi);
145             SAT(2) = -70*const.Rad;
146             SAT(3) = height(k);
147
148             % SET Sun's right ascension and declination (RADIANS)
149             % Difference between ephemeris time and universal time
150             % JD = MJD_UTC+2400000.5;
151             % [year, month, day, hour, minute, sec] = invjday(JD);
152             % days = finddays(year, month, day, hour, minute, sec);
153             % ET_UT = ETminUT(year+days/365.25);
154             % MJD_ET = MJD_UTC+ET_UT/86400;
155             % [r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus, ...
156             %  r_Neptune,r_Pluto,r_Moon,r_Sun,r_SunSSB] = JPL_Eph_DE430(MJD_ET);
157
158             MJD_TDB = Mjday_TDB(TT);
159             [r_Mercury,r_Venus,r_Earth,r_Mars,r_Jupiter,r_Saturn,r_Uranus, ...
160              r_Neptune,r_Pluto,r_Moon,r_Sun,r_SunSSB] = JPL_Eph_DE430(MJD_TDB);
161             ra_Sun  = atan2(r_Sun(2), r_Sun(1));
162             dec_Sun = atan2(r_Sun(3), sqrt(r_Sun(1)^2+r_Sun(2)^2));
163             SUN(1)  = ra_Sun;
164             SUN(2)  = dec_Sun;
165
166             % COMPUTE DENSITY KG/M3 RHO
167             [TEMP,RHO] = JB2008(MJD,SUN,SAT,F10,F10B,S10,S10B,XM10,XM10B,Y10,Y10B,DSTDTC);
168
169             % Density for each height and drag
170             rho_data(k,drag_counter) = RHO;
171             velocity(k) = sqrt((G*M_E)/(R_E+height(k))); % [m/s]
172         end
173     end
174 end
175
176 % Estimate a mean orbital velocity
177 mean_velocity = mean(velocity);
178
179 plot_pdf1 = figure;
180 [X_CD, Y_height] = meshgrid(CD,height);
```

```matlab
181  % Calculate aero_torque
182  aero_torque = 0.5.*X_CD.*rho_data*Area*mean_velocity^2*r_dA*3/sqrt(2);
183  % Countourf to plot the data
184  contourf(X_CD,Y_height,aero_torque,'ShowText','on');
185  % shading interp (optional)
186  title('\textbf{Moderate Solar activity: F10.7 = 150  $\mathbf{s.f.u.}$}');
187  xlabel('Drag coefficient $C_D$ [adim]');
188  ylabel('Orbital Altitude $h$ [$\mathrm{km}$]');
189  colorbar_label = colorbar;
190  colorbar_label.Label.Interpreter = 'latex';
191  set(colorbar_label,'FontSize',11);
192  colorbar_label.Label.String = 'Atmospheric drag torque [$\mathrm{N \cdot m}$]';
193  grid on;
194
195  % Save plots in .pdf and .png
196
197  % % Figure 1
198  % set(plot_pdf1, 'Units', 'Centimeters');
199  % pos = get(plot_pdf1, 'Position');
200  % set(plot_pdf1, 'PaperPositionMode', 'Auto', 'PaperUnits', 'Centimeters', ...
201  %          'PaperSize', [pos(3), pos(4)]);
202  % print(plot_pdf1, 'JB2008_F107_150.pdf', '-dpdf', '-r0');
203  %
204  % % Save png
205  % print(plot_pdf1,'JB2008_F107_150.png','-dpng','-r1000');
```

Code K.3 Jacchia-Bowman JB2008 Evolution of the atmosphere torque in terms of h C_D and solar activity. Source: Own.

Appendix L

IMU Data Acquisition code

L.1 BNO055 IMU

The following code collects the data from the BNO055 IMU sensor [53].

First, the IMU must be calibrated:

```
1  /* This code reads calibrates the BNO055 IMU Sensor
2  */
3
4  // Libraries
5  #include <Wire.h> // This library allows to communicate with I2C / TWI devices.
6  #include <Adafruit_Sensor.h> // Library with drivers that are based on the ...
       Adafruit Unified Sensor Driver
7  #include <Adafruit_BNO055.h> // This is a library for the BNO055 orientation sensor
8  #include <utility/imumaths.h> // Inertial Measurement Unit Maths Library (it ...
       includes matrix.h, quaternions.h and vector.h)
9
10 // Global parameters and objects
11 #define BNO055_SAMPLERATE_DELAY_MS (100) // Define how fast the sensor sample rate ...
       (sample every 100 ms)
12
13 Adafruit_BNO055 IMU = Adafruit_BNO055(); // Create IMU object and set what the ...
       object is
14
15 // Calibration function
16 void displayCalStatus(void)
17 {
18    /*  Get the four calibration values (0..3)
19        Any sensor data reporting 0 should be ignored,
20        3 means 'fully calibrated" */
21    uint8_t system, gyros, accel, mg =0;
22    IMU.getCalibration(&system, &gyros, &accel, &mg);
23
24    // The data should be ignored until the system calibration is > 0
```

222

```
25    Serial.print("\t");
26    if (!system)
27    {
28      Serial.print("! ");
29    }
30
31    // Display the individual values
32    Serial.print("Sys:");
33    Serial.print(system, DEC);
34    Serial.print(" Gyros:");
35    Serial.print(gyros, DEC);
36    Serial.print(" Accel:");
37    Serial.print(accel, DEC);
38    Serial.print(" Magne:");
39    Serial.println(mg, DEC);
40  }
41
42  void setup() {
43    // Set the baud rate speed. This is how fast the data is to be sent through the ...
          USB connection
44    Serial.begin(115200);
45
46    // Print initial message
47    Serial.println("BNO055 IMU Sensor Raw Data Initialized");
48    Serial.println("");
49
50    // Start the IMU sensor
51    IMU.begin();
52
53    // Wait 1000 ms to ensure the sensor starts correctly
54    delay(1000);
55
56    // Change the time clock on the chip to the time clock on board of the IMU
57    IMU.setExtCrystalUse(true);
58  }
59
60  void loop() {
61
62    displayCalStatus();
63
64    // Wait the specified delay before requesting next data
65    delay(BNO055_SAMPLERATE_DELAY_MS);
66
67  }
```

Code L.1 Arduino IMU calibration code. Source: Own.
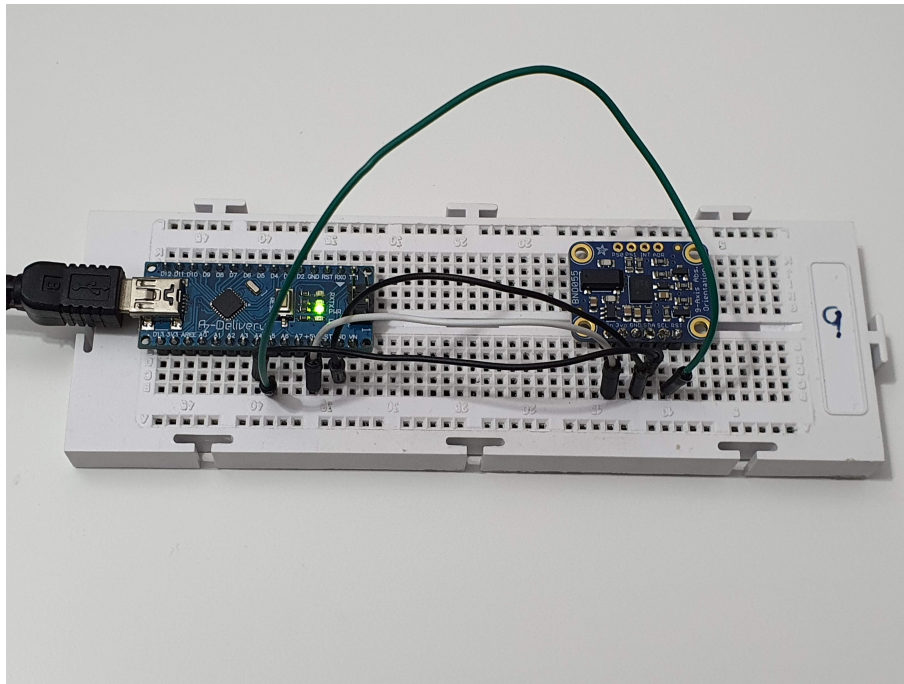
Figure L.1 Configuration of the BNO555 and the Arduino Nano connections. Source: Own.

The following code collects the data from the BNO055 IMU sensor [53].

To begin to use the code, two libraries must be installed:

```
1       Adafruit Unified Sensor Driver
2       Adafruit Unified BNO055 Driver (AHRS/Orientation)
```

Code L.2 Adafruit Libraries for Arduino IDE. Source: Own

The first library incorporates all Adafruit Sensor Drivers, from accelerometers, gyroscopes, magnetometers, barometric pressure, etc.. Besides, the Adafruit Unified Sensor Library provides a common interface and data type for any supported sensor. It defines some basic information about the sensor (sensor limits, etc.), and returns standard SI units of a specific type and scale for each supported sensor type. [87].

The second library is the driver for the Adafruit BNO055 Breakout, and is based on Adafruit's Unified Sensor Library (Adafruit_Sensor) [53].

First, the IMU must be calibrated using L calibration function. The calibration process is simple, 3 steps are needed to calibrate the IMU:

- To calibrate the **gyroscope**, just let the IMU rest in a flat surface.
- To calibrate the **accelerometer**, tilt the IMU 45 ° in all 3 axis x, y, z.
- To calibrate the **magnetometer**, swing the IMU and rotate it in all axis.

Once all of the instruments are correctly calibrated (calibration are ranged from $0 - 3$, when the output

is 3, the sensor is correctly calibrated), the system parameter should be calibrated as well.

The code implemented to extract the Acceleration, Gyroscope and Magnetometer's data is shown below [99]:

```
1  /* This code reads [ACCELEROMETER, GYROSCOPE and MAGNETOMETER'S] raw data from the ...
      BNO055 IMU Sensor
2
3     Connections
4     ===========
5     Connect SCL to SCL pin (analog 5 on Arduino UNO)
6     Connect SDA to SDA pin (analog 4 on Arduino UNO)
7     Connect Vin to 3-5V DC (depending on your board's logic level)
8     Connect GROUND to common ground
9
10 */
11
12 /* Units
13
14  - VECTOR_ACCELEROMETER - m/s^2
15  - VECTOR_MAGNETOMETER  - uT
16  - VECTOR_GYROSCOPE     - rad/s
17  - VECTOR_EULER         - degrees
18  - VECTOR_LINEARACCEL   - m/s^2
19  - VECTOR_GRAVITY       - m/s^2
20 */
21
22 // Libraries
23 #include <Wire.h> // This library allows to communicate with I2C / TWI devices.
24 #include <Adafruit_Sensor.h> // Library with drivers that are based on the ...
      Adafruit Unified Sensor Driver
25 #include <Adafruit_BNO055.h> // This is a library for the BNO055 orientation sensor
26 #include <utility/imumaths.h> // Inertial Measurement Unit Maths Library (it ...
      includes matrix.h, quaternions.h and vector.h)
27
28 // Global parameters and objects
29 #define BNO055_SAMPLERATE_DELAY_MS (100) // Define how fast the sensor sample rate ...
      (sample every 100 ms)
30
31 Adafruit_BNO055 IMU = Adafruit_BNO055(); // Create IMU object and set what the ...
      object is
32
33 // Calibration function
34 void displayCalStatus(void)
35 {
36   /*  Get the four calibration values (0..3)
37       Any sensor data reporting 0 should be ignored,
38       3 means 'fully calibrated" */
39   uint8_t system, gyros, accel, mg =0;
40   IMU.getCalibration(&system, &gyros, &accel, &mg);
41
42   // The data should be ignored until the system calibration is > 0
43   Serial.print("\t");
44   if (!system)
45   {
```

```
46        Serial.print("! ");
47    }
48
49    // Display the individual values
50    Serial.print("Sys:");
51    Serial.print(system, DEC);
52    Serial.print(" Gyros:");
53    Serial.print(gyros, DEC);
54    Serial.print(" Accel:");
55    Serial.print(accel, DEC);
56    Serial.print(" Magne:");
57    Serial.print(mg, DEC);
58 }
59
60 void setup() {
61    // Set the baud rate speed. This is how fast the data is to be sent through the ...
          USB connection
62    Serial.begin(115200);
63
64    // Print initial message
65    Serial.println("BNO055 IMU Sensor Raw Data Initialized");
66    Serial.println("");
67
68    // Start the IMU sensor
69    IMU.begin();
70
71    // Wait 1000 ms to ensure the sensor starts correctly
72    delay(1000);
73
74    // Get temperature from the IMU and save it into a 8-bit int variable named 'temp'
75    int8_t temp = IMU.getTemp();
76
77    // Print the temperature through the Serial Monitor
78    Serial.println(temp);
79
80    // Change the time clock on the chip to the time clock on board of the IMU
81    IMU.setExtCrystalUse(true);
82 }
83
84 void loop() {
85
86    // Display Calibration status (This line is optional)
87    // displayCalStatus();
88
89    // Work with the imu sensor from Adafruit library
90    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'acc' ...
          (accelerometer) for the specific object IMU
91    imu::Vector<3> acc = IMU.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
92    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'gyro' ...
          (gyroscope) for the specific object IMU
93    imu::Vector<3> gyro = IMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
94    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'mag' ...
          (magnetometer) for the specific object IMU
95    imu::Vector<3> mag = IMU.getVector(Adafruit_BNO055::VECTOR_MAGNETOMETER);
96
97    // Print Acceleration data
98    Serial.print(acc.x());
99    Serial.print(",");
```

```
100    Serial.print(acc.y());
101    Serial.print(",");
102    Serial.print(acc.z());
103    Serial.print(",");
104
105    // Print Gyroscope data
106    Serial.print(gyro.x());
107    Serial.print(",");
108    Serial.print(gyro.y());
109    Serial.print(",");
110    Serial.print(gyro.z());
111    Serial.print(",");
112
113    // Print Magnetometer data
114    Serial.print(mag.x());
115    Serial.print(",");
116    Serial.print(mag.y());
117    Serial.print(",");
118    Serial.println(mag.z());
119
120    // Wait the specified delay before requesting next data
121    delay(BNO055_SAMPLERATE_DELAY_MS);
122
123  }
```

Code L.3 Arduino and BNO055 IMU Raw data acquisition code. Source: Own.

The IMU has an on-board algorithm for continuous real-time generating quaternions.

L.2 Pitch and roll from accelerometers code

```
1  /* This code reads [ACCELEROMETER] from the BNO055 IMU Sensor and calculates pitch ...
       and roll angles
2  */
3
4  // Libraries
5  #include <Wire.h> // This library allows to communicate with I2C / TWI devices.
6  #include <Adafruit_Sensor.h> // Library with drivers that are based on the ...
       Adafruit Unified Sensor Driver
7  #include <Adafruit_BNO055.h> // This is a library for the BNO055 orientation sensor
8  #include <utility/imumaths.h> // Inertial Measurement Unit Maths Library (it ...
       includes matrix.h, quaternions.h and vector.h)
9  #include <math.h> // Math functions
10
11  // Global parameters and objects
12  #define BNO055_SAMPLERATE_DELAY_MS (100) // Define how fast the sensor sample rate ...
       (sample every 100 ms)
13
14  Adafruit_BNO055 IMU = Adafruit_BNO055(); // Create IMU object and set what the ...
       object is
15
16  const float pi = 3.1415926535; // Number Pi
```

```
17
18   //Variables
19   float theta, phi;
20
21   void setup() {
22     // Set the baud rate speed. This is how fast the data is to be sent through the ...
           USB connection
23     Serial.begin(115200);
24
25     // Print initial message
26     Serial.println("BNO055 IMU Sensor Raw Data Initialized");
27     Serial.println("");
28
29     // Start the IMU sensor
30     IMU.begin();
31
32     // Wait 1000 ms to ensure the sensor starts correctly
33     delay(1000);
34
35     // Change the time clock on the chip to the time clock on board of the IMU
36     IMU.setExtCrystalUse(true);
37   }
38
39   void loop() {
40
41     // Work with the imu sensor from Adafruit library
42     // Go to BNO055's 'imu' and bring back a vector of 3 components into 'acc' ...
           (accelerometer) for the specific object IMU
43     imu::Vector<3> acc = IMU.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
44
45     // Parameters
46     // Tilt approximation and normalize it to '1g' and convert into degrees
47     theta = - atan2(acc.x()/9.81,acc.z()/9.81)/(2*pi)*360;
48     phi = - atan2(acc.y()/9.81,acc.z()/9.81)/(2*pi)*360;
49
50     // Print Acceleration data
51     Serial.print(acc.x()/9.81);
52     Serial.print(",");
53     Serial.print(acc.y()/9.81);
54     Serial.print(",");
55     Serial.print(acc.z()/9.81);
56     Serial.print(",");
57     Serial.print(theta);
58     Serial.print(",");
59     Serial.println(phi);
60
61     // Wait the specified delay before requesting next data
62     delay(BNO055_SAMPLERATE_DELAY_MS);
63
64   }
```

Code L.4 Arduino BNO055 IMU pitch and roll code. Source: Own.

L.3 Complementary filter code

```
1  /* This code reads [ACCELEROMETER] and [GYROSCOPE] data from the BNO055 IMU Sensor
2   * with a Complementary Filter for the output signal
3   */
4
5  // Libraries
6  #include <Wire.h> // This library allows to communicate with I2C / TWI devices.
7  #include <Adafruit_Sensor.h> // Library with drivers that are based on the ...
       Adafruit Unified Sensor Driver
8  #include <Adafruit_BNO055.h> // This is a library for the BNO055 orientation sensor
9  #include <utility/imumaths.h> // Inertial Measurement Unit Maths Library (it ...
       includes matrix.h, quaternions.h and vector.h)
10 #include <math.h> // Math functions
11
12 // Global parameters and objects
13 #define BNO055_SAMPLERATE_DELAY_MS (100) // Define how fast the sensor sample rate ...
       (sample every 100 ms)
14
15 Adafruit_BNO055 IMU = Adafruit_BNO055(); // Create IMU object and set what the ...
       object is
16
17 const float pi = 3.1415926535; // Number Pi
18
19 //Variables
20 float theta, phi;
21 float theta_meas, phi_meas;
22 float theta_filt_old = 0, phi_filt_old = 0;
23 float theta_filt_new, phi_filt_new;
24
25 float theta_gyro = 0, phi_gyro = 0;
26 float Dt; // Change in time
27 unsigned long millisOld; // Marker of t[n-1]
28
29 void setup() {
30   // Set the baud rate speed. This is how fast the data is to be sent through the ...
       USB connection
31   Serial.begin(115200);
32
33   // Print initial message
34   Serial.println("BNO055 IMU Sensor Raw Data Initialized");
35   Serial.println("");
36
37   // Start the IMU sensor
38   IMU.begin();
39
40   // Wait 1000 ms to ensure the sensor starts correctly
41   delay(1000);
42
43   // Change the time clock on the chip to the time clock on board of the IMU
44   IMU.setExtCrystalUse(true);
45
46   // Start millis counter
47   millisOld = millis();
48 }
```

```
49
50  void loop() {
51
52    // Work with the imu sensor from Adafruit library
53    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'acc' ...
          (accelerometer) for the specific object IMU
54    imu::Vector<3> acc = IMU.getVector(Adafruit_BNO055::VECTOR_ACCELEROMETER);
55    // Go to BNO055's 'imu' and bring back a vector of 3 components into 'gyro ...
          (gyroscope) for the specific object IMU
56    imu::Vector<3> gyro = IMU.getVector(Adafruit_BNO055::VECTOR_GYROSCOPE);
57
58    // Parameters
59    // Tilt approximation and normalize it to '1g' and convert into degrees
60    theta_meas = - atan2(acc.x()/9.81,acc.z()/9.81)/(2*pi)*360;
61    phi_meas = - atan2(acc.y()/9.81,acc.z()/9.81)/(2*pi)*360;
62
63    theta_filt_new = theta_filt_old*0.9 + theta_meas*0.1;
64    phi_filt_new = phi_filt_old*0.9 + phi_meas*0.1;
65
66    // Change in time (seconds)
67    Dt = (millis()-millisOld)/1000.;
68    millisOld = millis();
69
70    // Complementary filter
71    theta = (theta + gyro.y()*Dt)*0.95 + theta_meas*0.05;
72    phi = (phi - gyro.x()*Dt)*0.95 + phi_meas*0.05; // Minus sign is for the ...
          particular orientation
73
74    theta_gyro = theta_gyro + gyro.y()*Dt;
75    phi_gyro = phi_gyro - gyro.x()*Dt; // Negative sign due to body axis
76
77    // Print Acceleration data
78    Serial.print(acc.x()/9.81);
79    Serial.print(",");
80    Serial.print(acc.y()/9.81);
81    Serial.print(",");
82    Serial.print(acc.z()/9.81);
83    Serial.print(",");
84    // Raw data
85    Serial.print(theta_meas);
86    Serial.print(",");
87    Serial.print(phi_meas);
88    Serial.print(",");
89    // Filtered data
90    Serial.print(theta_filt_new);
91    Serial.print(",");
92    Serial.print(phi_filt_new);
93
94    // Gryoscope data
95    Serial.print(",");
96    Serial.print(gyro.y()); // Pitch Rotational velocity
97    Serial.print(",");
98    Serial.print(gyro.x()); // Roll Rotational velocity
99
100   Serial.print(",");
101   Serial.print(theta_gyro); // Pitch
102   Serial.print(",");
103   Serial.print(phi_gyro); // Roll
```

```
104
105    // Complementary filter data
106    Serial.print(",");
107    Serial.print(theta); // Pitch
108    Serial.print(",");
109    Serial.println(phi); // Roll
110
111    // Next iteration
112    theta_filt_old = theta_filt_new;
113    phi_filt_old = phi_filt_new;
114
115    // Wait the specified delay before requesting next data
116    delay(BNO055_SAMPLERATE_DELAY_MS);
117
118  }
```

Code L.5 Arduino BNO055 IMU Complementary Filter. Source: Own.

L.4   Quaternion

```
1   /* This code reads [QUATERNIONS] data from the BNO055 IMU Sensor
2   */
3
4   // Libraries
5   #include <Wire.h> // This library allows to communicate with I2C / TWI devices.
6   #include <Adafruit_Sensor.h> // Library with drivers that are based on the ...
         Adafruit Unified Sensor Driver
7   #include <Adafruit_BNO055.h> // This is a library for the BNO055 orientation sensor
8   #include <utility/imumaths.h> // Inertial Measurement Unit Maths Library (it ...
         includes matrix.h, quaternions.h and vector.h)
9   #include <math.h> // Math functions
10
11  // Global parameters and objects
12  #define BNO055_SAMPLERATE_DELAY_MS (100) // Define how fast the sensor sample rate ...
         (sample every 100 ms)
13
14  Adafruit_BNO055 IMU = Adafruit_BNO055(); // Create IMU object and set what the ...
         object is
15
16
17  void setup() {
18    // Set the baud rate speed. This is how fast the data is to be sent through the ...
         USB connection
19    Serial.begin(115200);
20
21    // Start the IMU sensor
22    IMU.begin();
23
24    // Wait 1000 ms to ensure the sensor starts correctly
25    delay(1000);
26
27    // Change the time clock on the chip to the time clock on board of the IMU
28    IMU.setExtCrystalUse(true);
```

```
29
30  }
31
32  void loop() {
33
34    // Work with the imu sensor from Adafruit library
35    // Go to BNO055's 'imu' and bring back a vector of 4 components into 'quat' ...
         (quaternion) for the specific object IMU
36    imu::Quaternion quat=IMU.getQuat();
37
38    // Print Quaternion data
39    Serial.print(quat.w()); // Real part
40    Serial.print(",");
41    Serial.print(quat.x()); // 'i'
42    Serial.print(",");
43    Serial.print(quat.y()); // 'j'
44    Serial.print(",");
45    Serial.println(quat.z()); // 'k'
46
47    // Wait the specified delay before requesting next data
48    delay(BNO055_SAMPLERATE_DELAY_MS);
49
50  }
```

Code L.6 Arduino and BNO055 IMU Quaternion code. Source: Own.

L.5   MPU9250 IMU

To setup the IMU for I^2C communication protocol, first, it is needed to setup the address for the communication:

```
1   /*
2    * This code configures the MPU9250 IMU for I2C communication with the STM32 Bluepill
3    */
4
5   #include <Wire.h>
6
7   const int MPU = 0x68;                    // I2C address of the MPU9250
8   int16_t AcX, AcY, AcZ, Tmp, GyX, GyY, GyZ; // MPU values
9
10  void setup()
11  {
12    Wire.begin();              // Initiate the library and join the I2C bus
13    Wire.beginTransmission(MPU); // Begin a transmission with the given address
14    Wire.write(0x6B);          // Send value in bytes
15    Wire.write(0);
16    Wire.endTransmission(true); // End the transmission
17  }
18
19  void loop()
20  {
21    Wire.beginTransmission(MPU); // Begin a transmission with the given address
```

```
22    Wire.write(0x3B);
23    Wire.endTransmission(false);
24    Wire.requestFrom(MPU, 12, true);
25
26    AcX = Wire.read() << 8 | Wire.read();
27    AcY = Wire.read() << 8 | Wire.read();
28    AcZ = Wire.read() << 8 | Wire.read();
29    GyX = Wire.read() << 8 | Wire.read();
30    GyY = Wire.read() << 8 | Wire.read();
31    GyZ = Wire.read() << 8 | Wire.read();
32
33    Serial.print("IMU values: ");
34    Serial.print(AcX);
35    Serial.print(",");
36
37    // Repeat for each value
38
39    Serial.print(GyY);
40    Serial.print(",");
41    Serial.print(GyZ);
42  }
```

Code L.7 STM32 I^{2}C configuration of the MPU9250. Source: Own.

```
1  /*
2   * This code configures the MPU9250 IMU for SPI communication with the STM32 ...
        Bluepill and prints data.
3   *
4   * MPU9250 library by Brian Chen used (https://github.com/brianc118/MPU9250)
5   */
6
7  #include "Wire.h"
8  #include <SPI.h>
9  #include <MPU9250.h>
10
11 #define SPI_CLOCK 1000000 // 1MHz clock works.
12 #define CS1 PA4           // STM32 NCS pin
13 #define WAITFORINPUT() {  // Function to read serial imputs
14 while (!Serial.available())
15 {
16 };
17 while (Serial.available())
18 {
19   Serial.read();
20 };
21 }
22
23 MPU9250 mpu(SPI_CLOCK, CS1); // MPU object
24
25 void setup()
26 {
27   pinMode(CS1, OUTPUT); // Pin definition
28   SPI.begin();          // Initiate the library and join the SPI bus
29
30   Serial.println("Press any key to continue");
```

```
31    WAITFORINPUT();
32
33    mpu.init(true); // Initiate the IMU
34
35    uint8_t wai = mpu.whoami(); // Connection status with mpu
36    if (wai == 0x71)
37    {
38      Serial.println("Successful connection");
39    }
40    else
41    {
42      Serial.print("Failed connection: ");
43      Serial.println(wai, HEX);
44    }
45
46    uint8_t wai_AK8963 = mpu.AK8963_whoami(); // Connection status with mag
47    if (wai_AK8963 == 0x48)
48    {
49      Serial.println("Successful connection to mag");
50    }
51    else
52    {
53      Serial.print("Failed connection to mag: ");
54      Serial.println(wai_AK8963, HEX);
55    }
56
57    mpu.calib_acc(); // Calibrations
58    mpu.calib_mag();
59
60    Serial.println("Send any char to begin main loop.");
61    WAITFORINPUT();
62  }
63
64  void loop()
65  {
66    mpu.read_all();
67    // Data is stored in variables from the library
68    Serial.print(mpu.gyro_data[0]);
69    Serial.print('\t');
70    Serial.print(mpu.gyro_data[1]);
71    Serial.print('\t');
72    Serial.print(mpu.gyro_data[2]);
73    Serial.print('\t');
74    Serial.print(mpu.accel_data[0]);
75    Serial.print('\t');
76    Serial.print(mpu.accel_data[1]);
77    Serial.print('\t');
78    Serial.print(mpu.accel_data[2]);
79    Serial.print('\t');
80    Serial.print(mpu.mag_data[0]);
81    Serial.print('\t');
82    Serial.print(mpu.mag_data[1]);
83    Serial.print('\t');
84    Serial.print(mpu.mag_data[2]);
85    Serial.print('\t');
86    Serial.print(mpu.temperature);
87  }
```

Code L.8 STM32 SPI configuration of the MPU9250. Source: Own.

```
1   /*
2    * SCMD library by SparkFun used ...
        (https://github.com/sparkfun/SparkFun_Serial_Controlled_Motor_Driver_Arduino_Library)
3    * Rest of libraries from Arduino
4    * STM32 package from http://dan.drown.org/stm32duino/package_STM32duino_index.json
5    *
6    * ---------- TEST SMT32 1 MOTOR + IMU, SAME I2C PORT (MOTOR IN I2C, IMU IN I2C2)
7    *
8    */
9
10  #include <Arduino.h>
11  #include <stdint.h>
12  #include "SCMD.h"
13  #include "SCMD_config.h" //Contains #defines for common SCMD register names and values
14  #include<Wire.h>
15
16  #define LEDPIN PC13 //STM32 LED pin
17
18  const int MPU=0x68; //I2C adress of the MPU9250
19  int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ; //MPU values
20
21  SCMD DriverOne; //Driver object
22
23  void setup() {
24
25    Wire.begin(); //Initiate the library and join the I2C bus
26    Serial.begin(9600); //Data rate in bytes per second  //Serial Monitor initiation
27    pinMode(LEDPIN, OUTPUT); //LED definition
28    Serial.println("Start");
29
30    //IMU
31    Wire.beginTransmission(MPU); //Begin a transmission with the given adress
32    Wire.write(0x6B); //Send value in bytes
33    Wire.write(0);
34    Wire.endTransmission(true); //End the transmission
35
36    //MOTOR
37    DriverOne.settings.commInterface = I2C_MODE; //Driver mode definition
38    DriverOne.settings.I2CAddress = 0x5D; //Driver adress definition (0x5D by default)
39
40    while(DriverOne.begin() != 0xA9){ //Wait for idle
41      Serial.println("ID Mismatch");
42      delay(200);
43    }
44    Serial.println("ID Match");
45
46    Serial.println("Waiting for enumeration"); //Wait for peripherals (enumeration)
47    while(DriverOne.ready() == false);
48    Serial.println("Ready");
49
50    while(DriverOne.busy()); //Enables the driver
51    DriverOne.enable();
```

```
52  }
53
54  void loop() {
55
56     int vel=255;
57     int steps=20;
58
59     for(int i= 0; i<=vel; i++){
60
61        DriverOne.setDrive(0,0,i);
62        Wire.beginTransmission(MPU); //Begin a transmission with the given adress
63        Wire.write(0x3B);
64        Wire.endTransmission(false);
65        Wire.requestFrom(MPU,12,true);
66
67        AcX=Wire.read()<<8|Wire.read();
68        AcY=Wire.read()<<8|Wire.read();
69        AcZ=Wire.read()<<8|Wire.read();
70        GyX=Wire.read()<<8|Wire.read();
71        GyY=Wire.read()<<8|Wire.read();
72        GyZ=Wire.read()<<8|Wire.read();
73
74        Serial.print("IMU values: ");
75        Serial.print(AcX);
76        Serial.print(",");
77        Serial.print(AcY);
78        Serial.print(",");
79        Serial.print(AcZ);
80        Serial.print(",");
81        Serial.print(GyX);
82        Serial.print(",");
83        Serial.print(GyY);
84        Serial.print(",");
85        Serial.print(GyZ);
86        Serial.print(",   Speed: ");
87        Serial.println(i);
88        delay(steps);
89     }
90     delay(5000);
91
92     for(int i= vel; i>=0; i--){
93
94        DriverOne.setDrive(0,0,i);
95        Wire.beginTransmission(MPU); //Begin a transmission with the given adress
96        Wire.write(0x3B);
97        Wire.endTransmission(false);
98        Wire.requestFrom(MPU,12,true);
99
100       AcX=Wire.read()<<8|Wire.read();
101       AcY=Wire.read()<<8|Wire.read();
102       AcZ=Wire.read()<<8|Wire.read();
103       GyX=Wire.read()<<8|Wire.read();
104       GyY=Wire.read()<<8|Wire.read();
105       GyZ=Wire.read()<<8|Wire.read();
106
107       Serial.print("IMU values: ");
108       Serial.print(AcX);
109       Serial.print(",");
```

```
110       Serial.print(AcY);
111       Serial.print(",");
112       Serial.print(AcZ);
113       Serial.print(",");
114       Serial.print(GyX);
115       Serial.print(",");
116       Serial.print(GyY);
117       Serial.print(",");
118       Serial.print(GyZ);
119       Serial.print(",   Speed: ");
120       Serial.println(i);
121       delay(steps);
122     }
123   delay(5000);
124
125   }
```

Code L.9 Motor driver test with IMU using I^{2}C. Source: Own.

```
1   /*
2    * SCMD library by SparkFun used ...
         (https://github.com/sparkfun/SparkFun_Serial_Controlled_Motor_Driver_Arduino_Library)
3    * MPU9250 library by Brian Chen used (https://github.com/brianc118/MPU9250)
4    * Rest of libraries from Arduino
5    * STM32 package from http://dan.drown.org/stm32duino/package_STM32duino_index.json
6    *
7    * ---------- TEST IMU IN SPI + MOTOR IN I2C + millis
8    *
9    * Definitions stablished made to adapt to STM32 ports instead of Arduino ...
         (Library is designed for Arduino)
10   * 1MHz of clock because SPI conection with IMU is max 1MHz (by datasheet)
11   */
12
13  #include <Arduino.h>
14  #include <stdint.h>
15  #include "SCMD.h"
16  #include "SCMD_config.h" //Contains #defines for common SCMD register names and values
17  #include "Wire.h"
18  #include <SPI.h>
19  #include <MPU9250.h>
20
21  #define LEDPIN PC13 //STM32 LED pin
22  #define SPI_CLOCK 1000000 // 1MHz clock works.
23  #define CS1   PA4 //STM32 NCS pin
24  //MOSI on PA7, MISO  on PA6 and CLK on PA5 by default on STM32
25
26  #define WAITFORINPUT(){
27    while(!Serial.available()){};
28    while(Serial.available()){
29      Serial.read();
30    };
31  }
32
33  MPU9250 mpu(SPI_CLOCK, CS1); //MPU object
34  SCMD DriverOne; //Driver object
```

```
35
36 unsigned long prevMillis=0; //Keeps track of previous MIllis() value
37 unsigned long currMillis=0; //Keeps track of current MIllis() value
38
39 int counter=0; //Counter for speed
40 bool laststate=true; //Current/Latest state of motor (false=decceleration, ...
       true=acceleration)
41 bool waiting=false; //Check if the motor is at a const speed
42
43 int maxspeed=255; //Max velocity to reach
44 int delayramp=100; //Delay for increase in speed, in milisec
45 int delaywait=5000; //Delay on waiting
46
47 void setup() {
48
49    counter=0; //Reset of counter
50    pinMode(LEDPIN, OUTPUT); //LED definition
51    pinMode(CS1, OUTPUT); //Pin definitions
52    digitalWrite(LEDPIN, HIGH);
53    Serial.begin(9600); //Serial Monitor initiation
54    Serial.println("Start");
55    SPI.begin();
56
57      Serial.println("Press any key to continue");
58    WAITFORINPUT();
59
60    //IMU
61    mpu.init(true);
62
63      uint8_t wai = mpu.whoami(); //Connection with mpu
64    if (wai == 0x71){
65      Serial.println("Successful connection");
66    }
67    else{
68      Serial.print("Failed connection: ");
69      Serial.println(wai, HEX);
70    }
71
72    uint8_t wai_AK8963 = mpu.AK8963_whoami(); //Connection with mag
73    if (wai_AK8963 == 0x48){
74      Serial.println("Successful connection to mag");
75    }
76    else{
77      Serial.print("Failed connection to mag: ");
78      Serial.println(wai_AK8963, HEX);
79    }
80
81    mpu.calib_acc();
82    mpu.calib_mag();
83
84    //MOTOR
85    DriverOne.settings.commInterface = I2C_MODE; //Driver mode definition
86    DriverOne.settings.I2CAddress = 0x5D; //Driver adress definition (0x5D by default)
87
88      while(DriverOne.begin() != 0xA9){ //Wait for idle
89      Serial.println("ID Mismatch");
90      delay(200);
91    }
```

```
 92     Serial.println("ID Match");
 93
 94     Serial.println("Waiting for enumeration"); //Wait for peripherals (enumeration)
 95     while(DriverOne.ready() == false);
 96     Serial.println("Ready");
 97
 98     while(DriverOne.busy()); //Enables the driver
 99     DriverOne.enable();
100
101       Serial.println("Send any char to begin main loop.");
102     WAITFORINPUT();
103   }
104
105   void loop() {
106
107   //Rollover is done in the ifs, with currMilils-prevMIllis. if it reaches rollover ...
            the difference remains the sameas it is an unsigned value
108     currMillis=millis();
109
110     if(unsigned int (currMillis-prevMillis)>=delayramp && waiting==false){ //State ...
            of accel or decel. Accounts for millis() override
111       prevMillis=currMillis;
112       if(counter<=maxspeed && laststate==true){ //Acceleration
113         counter=counter+1;
114         if(counter>maxspeed){ //Max speed reach
115           waiting=true;
116           counter=maxspeed;
117         }
118       }else if(counter>=0 && laststate==false){ //Deceleration
119         counter=counter-1;
120         if(counter<0){ //0 speed reach
121           waiting=true;
122           counter=0;
123         }
124       }
125     }
126
127     if(unsigned int (currMillis-prevMillis)>=delaywait && waiting==true){ //State of ...
            waiting. Accounts for millis() override
128       prevMillis=currMillis;
129       waiting=false;
130       if(laststate){ //Start of decel
131         laststate=false;
132       }else{
133         laststate=true; //Start of accel
134       }
135     }
136
137     DriverOne.setDrive(0,0,counter);
138     mpu.read_all();
139
140     Serial.print(mpu.gyro_data[0]);    Serial.print('\t');
141     Serial.print(mpu.gyro_data[1]);    Serial.print('\t');
142     Serial.print(mpu.gyro_data[2]);    Serial.print('\t');
143     Serial.print(mpu.accel_data[0]);   Serial.print('\t');
144     Serial.print(mpu.accel_data[1]);   Serial.print('\t');
145     Serial.print(mpu.accel_data[2]);   Serial.print('\t');
146     Serial.print(mpu.mag_data[0]);     Serial.print('\t');
```

```
147    Serial.print(mpu.mag_data[1]);    Serial.print('\t');
148    Serial.print(mpu.mag_data[2]);    Serial.print('\t');
149    Serial.println(mpu.temperature);
150
151    Serial.print("Millis: ");
152    Serial.print(currMillis);
153    Serial.print(", State: ");
154    if(waiting==false){
155      if(laststate==true){
156        Serial.print("ACCELERATION");
157      }else{
158        Serial.print("DECELERATION");
159      }
160    }else{
161      Serial.print("WAITING");
162    }
163    Serial.print(", Speed: ");
164    Serial.println(counter);
165    delay (10);
166  }
```

Code L.10 Motor driver test with IMU using SPI. Source: Own.

Appendix M

Complementary filter design

M.1 Pitch θ, Roll ϕ and Yaw ψ determination from $3 - axis$ accelerometer and gyroscope

This section aims to approximate roll, pitch and yaw angles with an approximation.

Let θ be the pitch angle and ϕ be the roll angle.
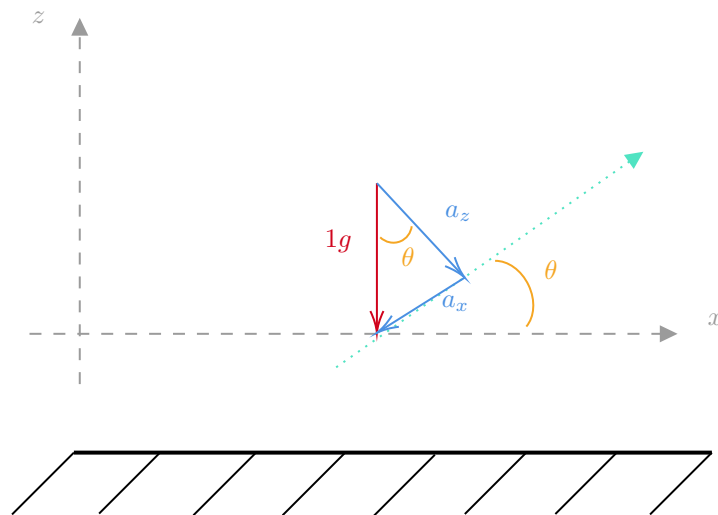
Figure M.1 Acceleration vector decomposition. Source: Own.

The following trigonometric relations can be extracted from the figure above:

$$\tan \theta = \frac{a_x}{a_z} \tag{M.1.1}$$

From the latter expression, the angle θ can be found as:

$$\theta = \arctan \frac{a_x}{a_z} \tag{M.1.2}$$

The same methodology can be used for the roll movement:

$$\phi = \arctan \frac{a_y}{a_z} \tag{M.1.3}$$

NOTE: This is a first approximation, thus, the major limitation found is that this only works for tilt angles from $0 - 89$ °. For angles above this range, the data returned is not relevant. The problem is caused by the trigonometric function tan(), when it reaches 90 ° the function has a vertical asymptote at that angle (see Figure M.2).
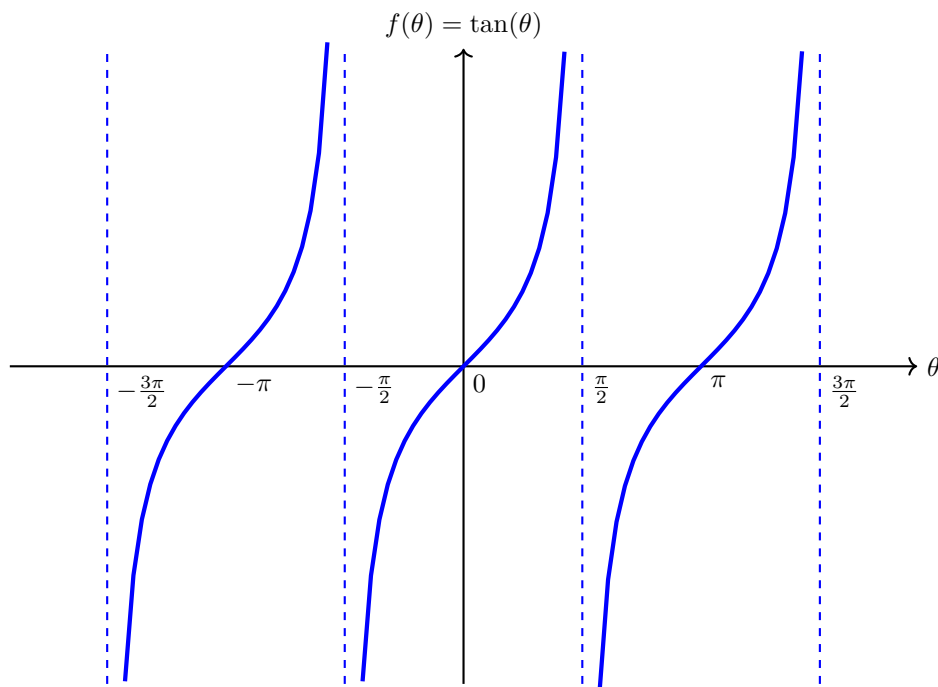
Figure M.2 Tangent function. Source: Own.

See L.2 for the code to get the pitch and roll from the IMU sensor.

Notice the code has a negative sign before the atan operation. This is because the IMU sensor's $\hat{x}$ axis is in the opposite direction of the $\hat{x}$ direction that the student has set initially.
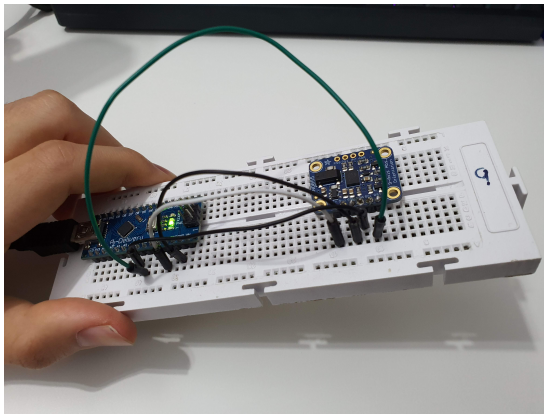
(a) Positive pitch of the
IMU sensor. Source: Own.
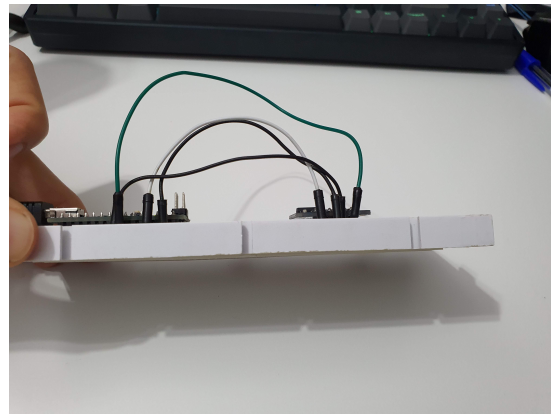
(b) Negative roll of the
IMU sensor. Source: Own.
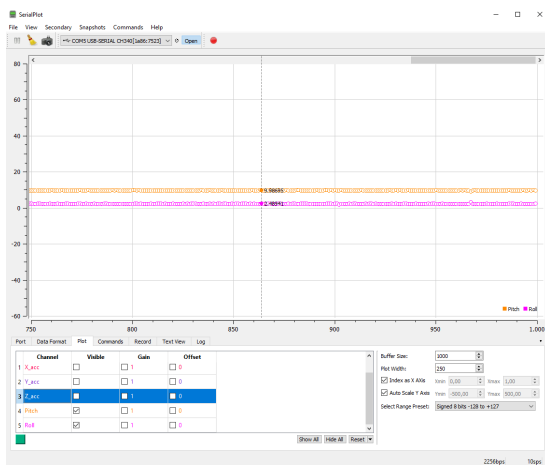
(c) Serial plot data of pitch
of the IMU sensor. Source:
Own.
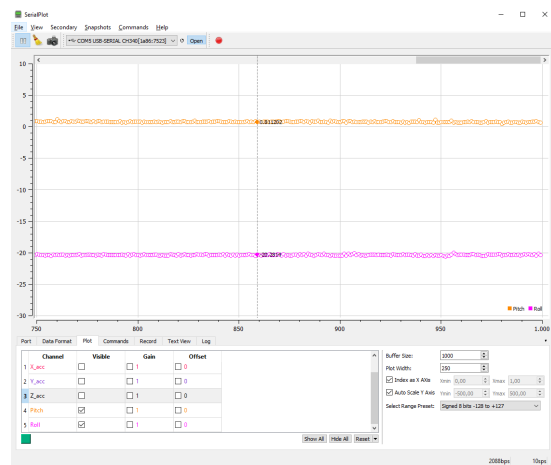
(d) Serial plot data of roll
of the IMU sensor. Source:
Own.

Figure M.3 Pitch and roll of the IMU Sensor. Source: Own.

The orange data is pitch and purple data is roll and both pitch and roll are successfully measured. Nevertheless, one issue that affects the accelerometer is that it is highly susceptible to vibrations and noise. For instance, shaking the sensor will cause the accelerometer misread a pitch, thus, causing an invalid data. The accelerometer is highly sensitive to all kind of vibrations and the system inappropriately interprets vibration as tilt. In other words, even if the IMU does not pitch but suffer vibrations or brusque motion, the output signal will measure a tilt when in fact it is not tilting.

To approach this problem a low pass filter must be added. A low pass filter enables to get rid f the higher frequency movements that the accelerometer senses and takes into account low frequency movements.

The accelerometer measures a pitch θ_{meas} angle and a roll angle ϕ_{meas} (measured values) and let θ_{filt} and ϕ_{filt} be the filtered values of pitch and roll angles, respectively.

The following low pass filter is added:

$$\theta_{filt,new} = [\theta_{filt,old}] \cdot P_1 + [\theta_{meas}] \cdot P_2 \qquad (M.1.4)$$

where

$$P_1 + P_2 = 1 \tag{M.1.5}$$

The above expressions can be interpreted as follows, given a measured angle, the newest pitch angle (most recent angle) is a sum of the prior iteration pitch angle times a parameter P_1 and the actual measured pitch angle times a parameter P_2. Notice that setting a $P_1 \approx 1$ value and $P_2 \approx 0$ shall be understood as the prior iteration angle is most susceptible to remain the actual movement of the IMU (for the test, a $P_1 = 0.9$ and $P_2 = 0.1$ was used). In the end, the most recent pitch angle is set to the old pitch angle for the subsequent iteration.

After adding the filter, Figure M.4 show the unfiltered and the filtered data of the pitch angle. The result now provides evidence to the fact that the accelerometer behaves poorly when it is subjected to high frequency movements. It is important to notice that the filtered value is slower than the unfiltered data. They both get to the same value but at a different pace. Thereby, in order to find the optimal parameters, there must be a balance between accurate data and how fast the IMU accelerometer sensor detects.
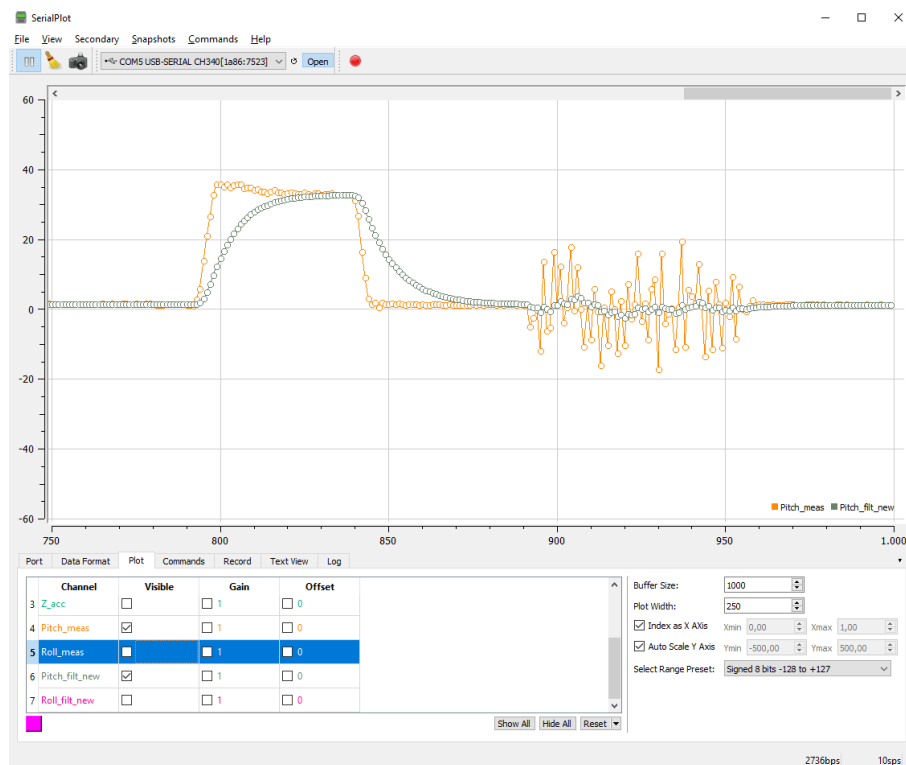
Figure M.4 Filtered positive pitch of the IMU sensor. Source: Own.

Nonetheless, the major advantage is that the filtered data is no susceptible to vibration motions. Overall, it tends to have more stability and smoother outputs.

Measuring rotational velocity with gyroscope

Now, due to the limitations given by the accelerometer. The next step is to use the built-in gyroscope to measure the angular velocity. By means of the uniform angular equation of motion:

$$\theta_{gyro,new} = \theta_{gyro,old} + \omega_y \Delta t \tag{M.1.6}$$

$$\phi_{gyro,new} = \theta_{gyro,old} + \omega_x \Delta t \tag{M.1.7}$$

where the angular position $\theta_{gyro,new}$ is the prior step position plus a change in angular velocity times the variation of time t. Since pitch is a rotation around the y axis, the angular velocity for pitch is ω_y and ω_x for roll.
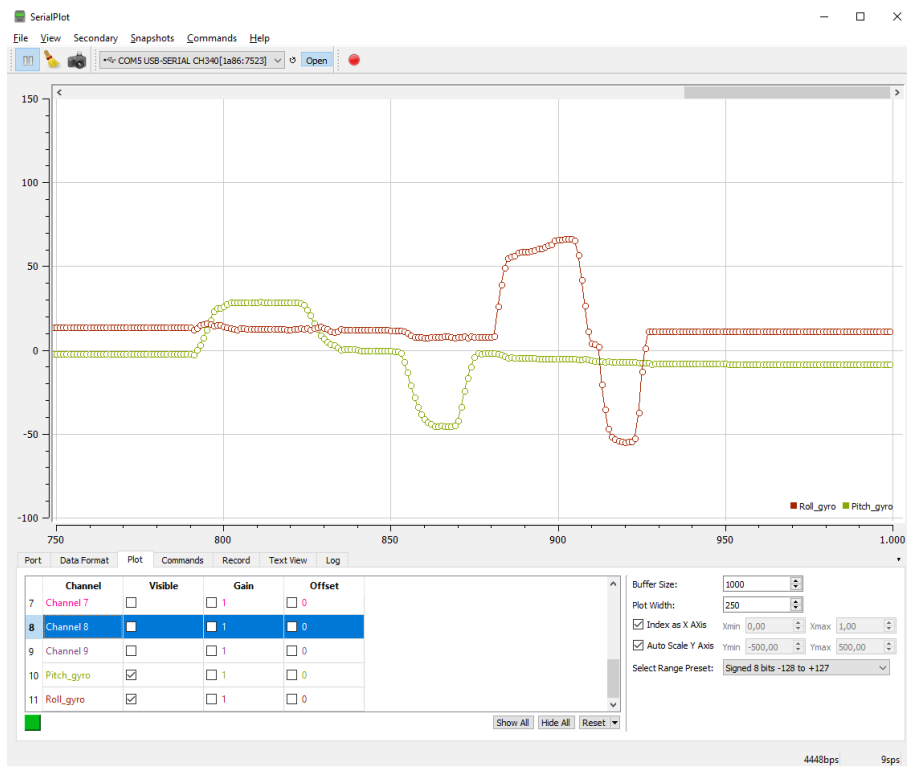
Figure M.5 Pitch and roll from the BNO055 Gyroscope sensor. Source: Own.

Figure M.5 shows the same values of pitch and tilt extracted from the gyroscope sensor. This plot shows first a positive pitch and then a negative roll between $x \in 850 - 875$. Additionally, from between $x \in 930 - 960$, some vibration and high frequency motion was induced to the sensor. Results suggest that kind of high frequency motion does not affect the gyroscope's sensibility. However, notice how the gyroscope shows a drift in the value after keeping it at rest in a flat surface as the original situation ($x \in 960 - 1000$). Thus, it is not possible to only use a gyroscope to get the tilt and roll data.

On balance the following conclusions are extracted:

- The accelerometer is useful for low frequency motion but is highly sensitive to noise and susceptible to vibration.

- The gyroscope is not sensitive to noise but its values drifts over time.

- Gyroscopes short-term data is accurate and accelerometers long-term data is unerring.

To address the aforementioned problem, the upcoming step resides in finding a solution by getting the best from the two worlds: accelerometer and gyroscope. The subsequent section proposes the use of a complementary filter combining both sensors data to acquire an accurate stable tilt and roll.

Design of a complementary filter

At the moment so far, 4 different parameters are extracted from the BNO055 sensor. The pitch and roll angles from the accelerometer sensor (θ_{acc} and ϕ_{acc}) and the pitch and roll angles from the gyroscope sensor θ_{gyro} and ϕ_{gyro}.

$$\left.\begin{array}{c}\theta_{acc}\\[4pt]\phi_{acc}\end{array}\right\}\quad \text{Trust long-term data} \tag{M.1.8}$$

$$\left.\begin{array}{c}\theta_{gyro}\\[4pt]\phi_{gyro}\end{array}\right\}\quad \text{Trust short-term data} \tag{M.1.9}$$

In order to design a complementary filter, all four parameters are kept and the main idea is to combine this four outputs which will allow the sensor to behave accurately in both sort and long term time period [100].

The complementary filter using all four parameters is calculated as follows:

$$\theta = \underbrace{[\theta + \omega_y \Delta t]}_{\theta_{gyro}} \cdot P_A + [\theta_{acc}] \cdot P_B \tag{M.1.10}$$

$$\phi = \underbrace{[\phi + \omega_x \Delta t]}_{\phi_{gyro}} \cdot P_A + [\phi acc] \cdot P_B \tag{M.1.11}$$

The aforementioned expression can be interpreted as follows. On the short term, the sensor will use the data from the gyroscope, because it is very precise and not susceptible to external forces. On the long term, the sensor uses the data from the accelerometer, as it does not drift. Parameters $P_A \approx 1$ and $P_B \approx 0$ in order to filter out high frequencies data for the accelerometer and filter out low frequencies for the gyroscope. It must be noticed that θ_{acc} and ϕ_{acc} are NOT the filtered values but the raw data.

The complementary filter has to be used in a infinite loop. For every iteration, the pitch and roll angles must be updated with the new data of the gyroscope values by means of integration over time. At the same time, the filter checks if the magnitude of the force seen by the accelerometer has a reasonable value that could be the real g-force vector. If the value is too small or too big, it is known for sure that it is a disturbance that shall not be taken into account. Afterwards, pitch and roll angles will be updated with the accelerometer data by taking $P_A = 95\%$ of the gyroscopes angle and adding $P_B = 5\%$ of the angle calculated by the accelerometer. This will ensure that the measurement will not drift, but that it will be very accurate on the short term [100].

Code L.3 implements the complementary filter explained beforehand.
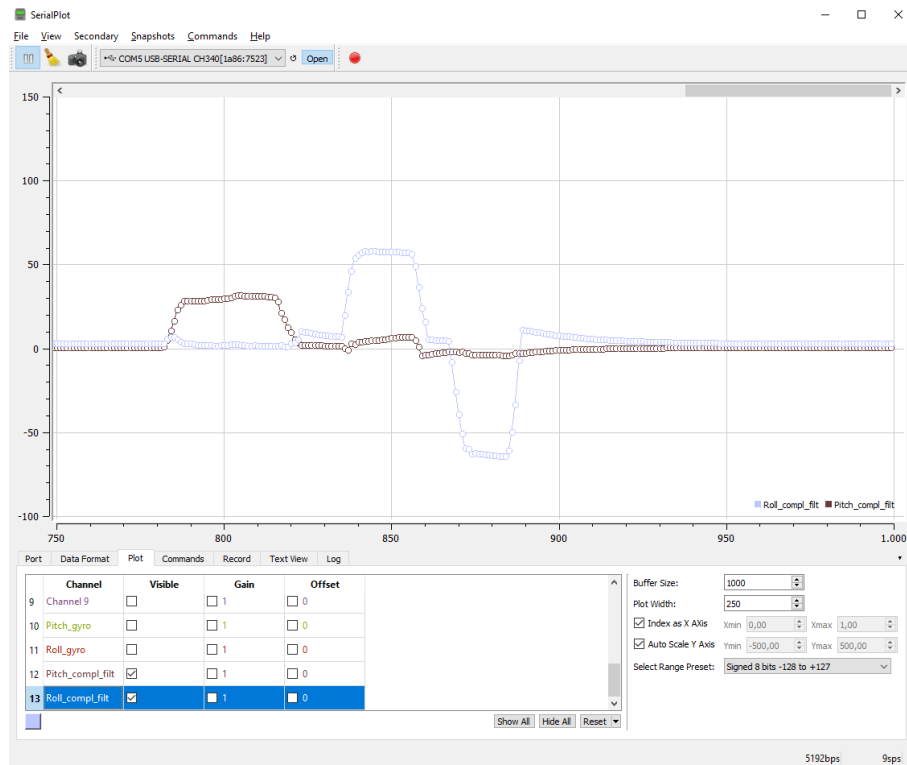
Figure M.6 Pitch and roll from the BNO055 Gyroscope sensor with a
Complementary Filter. Source: Own.

Figure M.6 clearly shows the responsiveness of responsiveness of the gyroscope data at short-term motion
and movements, keeping out the possible noise due to the accelerometer. Besides, stability is reached at
long-term as the sensor trusts the accelerometers data rather than the gyroscopes values. Nonetheless,
there is a modest overshoot when the IMU is brought back to the original position but after that it
immediately compensates out.

Adding Yaw ψ using the magnetometer

So far both pitch θ and roll ϕ were determined thanks to the Earth gravitational force vector as this
vector can be decomposed in two components at the IMU's body axis for any roll or pitch movement.
However, the gravitational force remains the same for any yaw motion, thus, by means of the magnetic
North Pole location, yaw angle ψ can be extracted.

(a) Magnetic North vector aligned with the Magnetometer's x axis. Source: Own.

(b) Magnetic North vector components with the Magnetometer's reference axis. Source: Own.
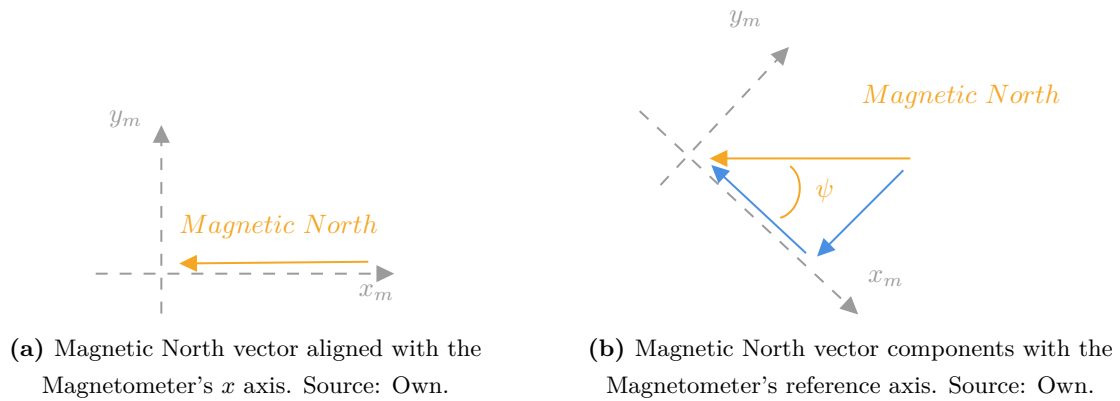
Figure M.7 Magnetometer's reference frame. Source: Own.

Keeping in mind the Earth magnetic North's vector, its components can easily be expressed in term of the magnetometer's reference frame (see Figure M.7). Thus,

$$\tan \psi = \frac{y_m}{x_m} \tag{M.1.12}$$

$$\psi = \arctan \frac{y_m}{x_m} \tag{M.1.13}$$

Nevertheless, the above expressions only works in a 2D motion with the IMU's magnetometer in a flat surface. If the sensor has a positive pitch angle $\theta > 0$ and then it yaws, the above expression does not take into account three dimensional motion. Likewise, projecting the vector into a three dimensional reference frame this problem can plainly be extrapolated to a 3D reference frame.

Here-under are the resultant data:
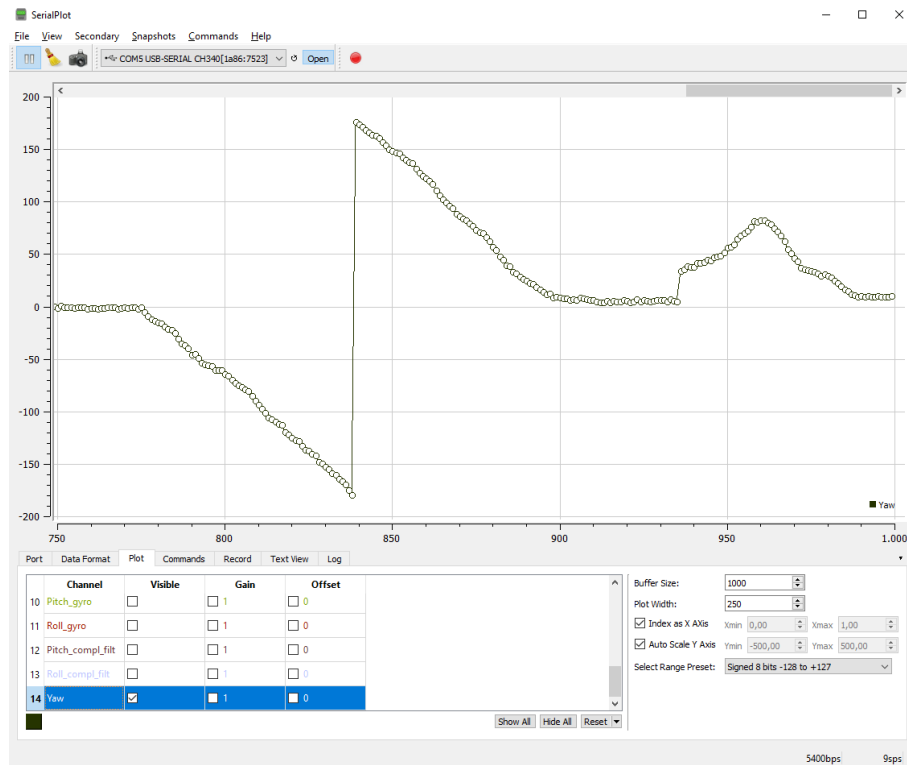
Figure M.8 Magnetometer's yaw plot. Source: Own.

Let's have a closer look at the plot (see Figure M.8). When the IMU's x axis is aligned with the Earth's magnetic North Pole vector, yaw angles is 0 °. Furthermore, as the IMU rotates the resultant yaw angle decreases up to a point where it is pointing towards the South then, a whole rotation is made. This is the reason why there is a jump from -180 ° and 180 °. In a unit circle, they are the same value. Notwithstanding, if the IMU rolls, the sensor interprets the roll as a change in heading (yaw) as the sensor is no longer aligned with the $x - y$ plane.

Appendix N

Python visualization code

This section shows how to extract data from Arduino and pass it into a Python script for further processing. The main requirement is to have python installed as well as serial library:

```
1      pip install serial
```

Code N.1 Libraries to be installed. Source: Own

Once done that, the program can extract the data using the following code (this code works with the IMU_data_acquisition_orientation_all_axis.ino arduino file).

N.1   Import Arduino data code

```
1   # This code imports ARDUINO data into python
2
3   # Libraries
4   import time
5   import serial
6
7   # Code
8   arduinoData = serial.Serial('com5', 115200)  # COM and baud rate
9
10  time.sleep(1)  # Delay of 1 s to ensure the serial port working
11
12  # Loop forever
13  while True:
14      # Loop until there is data into serial port (hang on until there is data)
15      while (arduinoData.inWaiting() == 0):
16          pass
17
18      # Since there is data in serial port, read the data
19      data = arduinoData.readline()
20      data = str(data, 'utf-8')  # Format the data
```

```
21      data = data.split(',')  # Split the data into a list
22
23      # print(data)
24
25      # Extract values from the accelerometer
26      acc_x = float(data[0])
27      acc_y = float(data[1])
28      acc_z = float(data[2])
29
30      # Extract filtered values (Complementary filter) of pitch and roll
31      pitch = float(data[11])
32      roll = float(data[12])
33
34      # Extract yaw
35      yaw = float(data[13])
36
37      print(
38          f"acc_x={acc_x}, acc_y={acc_y}, acc_z={acc_z}, pitch={pitch}, roll={roll}, ...
        yaw={yaw}")
```

Code N.2 Python code to extract Arduino data. Source: Own.

N.2 vPython 3D representation code

```
1  # The following code rotates a reference frame
2
3  # Libraries
4  from vpython import *  # vPython library
5  import numpy as np  # Numpy library
6  import time  # Time library
7  import math  # Math library
8
9  # Libraries
10 import serial
11
12 # Code
13 arduinoData = serial.Serial('com3', 115200)  # COM and baud rate
14
15 time.sleep(1)  # Delay of 1 s to ensure the serial port working
16
17 # View orientation
18 scene.forward = vector(-1, -1, -1)
19
20 scene.width = 600
21 scene.height = 600
22
23
24 # Adding the Arduino
25 breadBoard = box(length=6, width=6, height=6,
26                  color=color.white, opacity=0.25)
27
28 # Compound object
29 object = compound([breadBoard])
```

```
30
31  x_arrow = arrow(length=4, shaftwidth=0.25,
32                  color=color.red, axis=vector(1, 0, 0))
33  y_arrow = arrow(length=4, shaftwidth=0.25,
34                  color=color.green, axis=vector(0, 1, 0))
35  z_arrow = arrow(length=4, shaftwidth=0.25,
36                  color=color.cyan, axis=vector(0, 0, 1))
37
38  x_body_arrow = arrow(length=6, shaftwidth=0.25,
39                       color=color.purple, axis=vector(1, 0, 0))
40  y_body_arrow = arrow(length=6, shaftwidth=0.25,
41                       color=color.magenta, axis=vector(0, 1, 0))
42  z_body_arrow = arrow(length=6, shaftwidth=0.25,
43                       color=color.white, axis=vector(0, 0, 1))
44
45
46  # Loop forever
47  while True:
48      # Loop until there is data into serial port (hang on until there is data)
49      while (arduinoData.inWaiting() == 0):
50          pass
51
52      # Since there is data in serial port, read the data
53      data = arduinoData.readline()
54      data = str(data, 'utf-8')  # Format the data
55      data = data.split(',')  # Split the data into a list
56
57      # Quaternions
58      q0 = float(data[0])
59      q1 = float(data[1])
60      q2 = float(data[2])
61      q3 = float(data[3])
62
63      # Convert quaternions to euler angles
64      roll = -atan2(2*(q0*q1+q2*q3), 1-2*(q1**2+q2**2))
65      pitch = asin(2*q0*q2-q3*q1)
66      yaw = -atan2(2*(q0*q3+q1*q2), 1-2*(q2**2+q3**2))
67
68      rate(50)  # 50fps
69      k = vector(cos(yaw)*cos(pitch),
70                 sin(pitch), sin(yaw)*cos(pitch))
71
72      y = vector(0, 1, 0)
73      s = cross(k, y)  # Side vector
74      v = cross(s, k)  # Up vector
75
76      vrotate = v*cos(roll) + cross(k, v)*sin(roll)
77
78      x_body_arrow.axis = k
79      z_body_arrow.axis = cross(k, vrotate)
80      y_body_arrow.axis = vrotate
81      object.axis = k
82      object.up = vrotate  # Up direction
83      x_body_arrow.length = 4
84      z_body_arrow.length = 4
85      y_body_arrow.length = 4
```

Code N.3 3D CubeSat visualization. Source: Own.

Appendix O

Reaction Wheel Control code

Here is presented the full control algorithm, the control algorithm flowchart is presented again in a bigger scale:

```
                          Start General
                          Control Algorithm


                          Declaration of
                          definitions and
                          variables


                          Define Timers:                              Read IMU          Read OBC data
                          Serial1, Driver, IMU                                          (Turn angle value)
                                                                                                              No

                          Enter to OBC
                          Waiting mode                                                  Is the OBC data
                                                                                        value sent?

                                                                                              Yes
                  No
                          Read OBC mode                                                 Activate Emergency
                                                                                        Stop Timer


                          Is the OBC mode value
                          sent?                                                          Is OBC data value smaller
                                                                                         than Pointing tolerancy ?
                                  Yes

                          Save the mode                                        Yes                              No
                          value for mode
                          selection
                                                                            Fine Pointing              Coarse
                                                                                                        Pointing
                          Enter to selection
                          mode
                                                                                         No              Is the final position
                                                                                                         close enough to the
                  Is the value    Yes      Enter to positioning mode                                     setpoint?
                  mode: '1' ?               with Reaction Wheel

                          False

                  Is the value    Yes      Enter to IMU reading mode
                  mode: '2' ?               (ONLY FOR TESTING
                                            PURPOSES)
                          False
                                                                                                              Yes
                  Is the value    Yes      Enter to motor ON OFF
                  mode '3' ?                mode
                                            (ONLY FOR TESTING
                          False             PURPOSES)

                          Enter default mode
                          (Same as OBC
                          Waiting mode)

                                                                                         Is an Emergency Stop
                  Deactivate                                                             triggered?
                  Emergency Stop
                  Timer                                                         Yes
                                                                                                              No
                                                                            Stop motors

                  Deactivate
                  Emergency Stop
                  Timer


                  Deactivate
                  Emergency Stop
                  Timer
```

Start Fine Control

Retrieve position to reach (setpoint) and limits to [0º,360º]

Does the manoeuvre passes through 0º ?

Yes — Save value of last 'if' and add 180º to setpoint and limits to [0º,360º]

No — Save value of last 'if'

Read IMU

Does the manoeuvre passes through 0º ?

Yes — Add 180º to Yaw angle from IMU reading and limits to [0º,360º]

No

Calculate Yaw angle and setpoint angle in %

Compute proportional error, cumulative error and derivative error

Compute proportional action, integral action and derivative action

Compute PID output and limit to [-255,255]

Update motor speed and direction

Save current error for next iteration

Is Yaw angle close enough to the setpoint?

No

Enter to OBC Waiting mode

End of Fine Pointing

Start Coarse Control

Read IMU to get initial yaw angle

Start of acceleration

Generate acceleration ramp

Read IMU to get yaw angle after acceleration

Is the Reaction Wheel rotating counterclockwise?

Yes — Calculate the variation of angle (CubeSat rotates clockwise)

No — Calculate the variation of angle (CubeSat rotates counterclockwise)

Is the variation of angle negative?

Yes — Add 360º to maintain positive value

Is the variation of angle higher than ½ of the total turn angle?

No — Start of waiting

Yes — Start of deceleration

Is the Reaction Wheel rotating counterclockwise?

Compute the start of the deceleration angle (CubeSat rotates clockwise)

Compute the start of the deceleration angle (CubeSat rotates counterclockwise)

Read IMU

Does the trajectory passes through the 0º angle?

Yes — Does yaw angle increments?

No — Sustract a lap to yaw angle from IMU reading

Yes — Add a lap to yaw angle from IMU reading

Is Yaw angle from IMU reading bigger than the start of deceleration angle?

Does the trajectory passes through the 0º angle?

Yes — Does yaw angle increments?

No — Add a lap to yaw angle from IMU reading

Yes — Sustract a lap to yaw angle from IMU reading

Is Yaw angle from IMU reading smaller than the start of deceleration angle?

THIS ALGORITHM ONLY APPLIES FOR A SIMPLE IMPULSE RAMP

Generate ramp

Set impulse

Read IMU

Is the CubeSat in accelerating phase?

Yes — Is the CubeSat rotating at a constant speed?

No — Continue Coarse Pointing mode

Yes — Continue Coarse Pointing mode

No — Is the CubeSat at rest?

Yes — Continue Coarse Pointing mode

Start of deceleration

Generate deceleration ramp

Read IMU to get yaw angle after deceleration

Is the Reaction Wheel rotating counterclockwise?

Yes — Calculate the total variation of angle (CubeSat rotates clockwise)

No — Calculate the total variation of angle (CubeSat rotates counterclockwise)

Is the variation of angle negative?

Yes — Add 360º to maintain positive value

Is the difference between the total variation angle and the total turn angle within the pointing tolerance?

Yes — Enter to OBC Waiting mode — End of Coarse Pointing

No — Enter to Fine Pointing mode — End of Coarse Pointing

To download the code, check GitHub: Github

```
1  /* Reaction Wheel controller
2   *
3   * The following code provides a control for PLATHON project's CubeSat.
4   *
5   *
6   * The following libraries are used:
7   * STM32 Bluepill microcontroller package from ...
       http://dan.drown.org/stm32duino/package_STM32duino_index.json
8   * SCMD Driver library by SparkFun used (https:// ...
       github.com/sparkfun/SparkFun_Serial1_Controlled_Motor_Driver_Arduino_Library)
9   * MPU9250 IMU library by Rafa Castalla used ...
       (https://github.com/rafacastalla/MPU9250-1)
10  * read_IMU() and IMU setup content code extracted from Andres Gomez and Miquel ...
       Reurer, from the PLATHON group (magnetorquers section)
11  */
12
13  // ...

       DEFINITIONS
14  // ...

       LIBRARIES
15  #include <Arduino.h>      // Arduino library
16  #include <stdint.h>       // Integer types library
17  #include <SCMD.h>         // Serial Controlled Motor Driver library
18  #include <SCMD_config.h>  // Serial Controlled Motor Driver Configuration library
19  #include <SPI.h>          // SPI library
20  #include <MPU9250.h>      // IMU library
21  #include <Wire.h>
22
23  // ...

       DEFINITIONS
24  #define LEDPIN PC13              // Integrated LED of the Bluepill
25  #define CS1 4                    // STM32 NCS Chip select (SPI mode only) pin. ...
       MOSI on PA7, MISO on PA6 and CLK on PA5 by default on STM32
26  #define STM32_CLOCK 72000        // Internal STM32 Clock (72MHz, in kHz so ...
       period is millisec)
27  #define PI 3.14159265            // Number PI
28  #define Rad_to_deg 57.29577951   // Convert radians to degrees
29  #define Deg_to_rad 0.01745329    // Convert degrees to radians
30  #define Final_pointing_tolerance 1 // Tolerance for final pointing (+-1 degree of ...
       tolerance) (Check if position is within the margin)
31  #define Pointing_mode_tolerance 5  // Tolerance for selecting pointing mode (+-5 ...
       degree of tolerance) (Select whether to use coarse or fine mode)
32  #define Accel_tolerance 0.5      // Tolerance for acceleration measurement ...
       (+-0.5 unit of acceleration of tolerance) (Acceptable error of acceleration)
33  #define Gyro_tolerance 1         // Tolerance for gyro measurement (+-1 unit of ...
       gyroscope Z tolerance) (Accounts to know if it is rotating or not at a ...
       constant speed for the ramp)
34
35  SCMD DriverOne;         // Driver Object definition
36  MPU9250 IMU(SPI, CS1); // MPU object definition
```

259

```
37
38   // ...

         GLOBAL VARIABLES
39   float IMU_accel_data_X, IMU_accel_data_Y, IMU_accel_data_Z; // Values of local IMU ...
         accelerometer (m/s^2)
40   float IMU_gyro_data_X, IMU_gyro_data_Y, IMU_gyro_data_Z;    // Values of local IMU ...
         gyroscope (degrees/s)
41   float IMU_mag_data_X, IMU_mag_data_Y, IMU_mag_data_Z;      // Values of local IMU ...
         magnetometer (uT)
42   float Pitch_deg, Roll_deg, Yaw_deg = 0;                     // Pitch, Roll and Yaw ...
         (degrees)
43
44   // Declare variables outside IMU, since variables initiates to 0 without defining ...
         them in function
45   float Accel_total_vector_modulus;       // Accelerometer total vector magnitude ...
         (m/s^2)
46   float Accel_pitch_deg, Accel_roll_deg;  // Accelerometer pitch and roll angles ...
         (degrees)
47   float Gyro_pitch_deg, Gyro_roll_deg;    // Gyroscope pitch and roll angles (degrees)
48   bool Gyro_Accel_sync = false;           // Check if Gyroscope and Accelerometer ...
         are synchronized
49   float Mag_X_calc, Mag_Y_calc;           // Calculated Magnetometer value (uT)
50   float Mag_X_corrected, Mag_Y_corrected; // Corrected value Magnetometer value ...
         (with damping) (uT)
51   float Pitch_rad, Roll_rad;              // Pitch and Roll angles (rad)
52
53   int RW_speed = 0;       // Value of Reaction Wheel speed [from -255 to 255]
54   int OBC_mode_value = 0; // Value of On Board Computer mode (waiting, positioning, ...
         detumbling, etc.)
55   int OBC_data_value = 0; // Value of On Board Computer data (desired angle turn, etc.)
56   int Stop_state = 0;     // If Stop_state != 0, motor stops rotating
57
58   // PID control definitions
59   double PID_error, PID_last_error;            // Initialize error and previousError
60   double PID_cumulative_error, PID_rate_error; // Initialize the cumulative Error ...
         (Integral) and the rate of Error (Derivative)
61   float Current_Yaw_deg = 0;                   // Addition of 180 deg to all values. ...
         The error is a substract, so the difference is the same
62
63   double kp = 1; // Proportional contribution
64   double ki = 0; // Integral contribution
65   double kd = 1; // Derivative contribution
66
67   float Deg_to_reach = 0;  // Setpoint angle (degrees)
68   bool Zero_state = false; // Check if PID encompass yaw of 0 degrees (to turn the ...
         value to a workable zone)
69   int PID_output = 0;      // Output value of the PID [from 0 to 255]
70
71   // ...

         FUNCTIONS
72   // ...

         GENERAL USE FUNCTIONS
73   // ...

         OBC Functions
```

```
74  void OBC_mode_receive()
75  {
76    /*
77      Function to set the Bluepill to receiving mode. Uses Timer CH3. Serial1 as ...
        UART communication
78
79      INPUT: None, but gets mode from User Serial1 input
80      OUTPUT: None, but saves OBC_mode_value
81    */
82
83    // Check if UART Serial1 is available
84    read_IMU();
85
86    if (Serial1.available() > 0)
87    {
88      String bufferString = ""; // String for buffer of Serial1
89      // Keep saving input prompt
90      while (Serial1.available() > 0)
91      {
92        bufferString += (char)Serial1.read(); // Adds chars to the Serial1 buffer
93      }
94      // Conversion from String to int
95      OBC_mode_value = bufferString.toInt();
96      Serial1.print("Mode Number: ");
97      Serial1.println(OBC_mode_value);
98    }
99  }
100
101 void OBC_data_receive()
102 {
103   /*
104     Function to set the Bluepill to receiving data mode. Uses Timer CH3. Serial1 ...
        as UART communication
105
106     INPUT: None, but gets data from User Serial1 input
107     OUTPUT: Save OBC_data_value
108   */
109   read_IMU();
110   // Check if UART Serial1 is available
111   if (Serial1.available() > 0)
112   {
113     String bufferString = ""; // String for buffer of Serial1
114     // Keep saving input prompt
115     while (Serial1.available() > 0)
116     {
117       bufferString += (char)Serial1.read(); // adds chars to the Serial1 buffer
118     }
119     OBC_data_value = bufferString.toInt(); // Conversion from String to int
120   }
121 }
122
123 void EmergencyStop()
124 {
125   /*
126     Function to STOP the motor from rotating.
127
128     INPUT: None, but gets data from User Serial1 input
129     OUTPUT: Save OBC_data_value
```

```
130    */
131
132    // Check if UART Serial1 is available
133    if (Serial1.available() > 0)
134    {
135      String bufferString = ""; // String for buffer of Serial1
136      // Keep saving input prompt
137      while (Serial1.available() > 0)
138      {
139        bufferString += (char)Serial1.read(); // adds chars to the Serial1 buffer
140      }
141      Stop_state = bufferString.toInt();         // Conversion from String to int
142      Serial1.print("Emergency Stop activated"); // In this case turn, but can be ...
         what is needed
143    }
144
145    //  Timer1.attachInterrupt(TIMER_CH4, read_show_IMU);
146    // read_show_IMU();
147    // mode_Select(OBC_mode_value);
148
149    if (Stop_state != 0)
150    {
151      // If it stopped
152      set_impulse(0, 0);
153      Serial1.flush();
154      OBC_data_value = 0;
155      mode_OBC_Input_Wait();
156    }
157  }
158
159  // ...

       Driver Functions
160  void set_impulse(bool RW_direction, int New_RW_speed)
161  {
162    /*
163      Function to set the motor at a fixed direction and speed (B = port 1, A = port 0)
164
165      INPUT:
166        RW_direction: Reaction Wheel direction [Clockwise or Counter Clockwise] in A
167        New_RW_speed: New reaction wheel speed [from 0 to 255]
168      OUTPUT:
169        None, but saves speed of the Reaction Wheel
170    */
171
172    // Account for change in direction
173    // If motor's z-axis direction is coincident with CubeSat's z-axis direction
174    /**
175     * RW_direction == true  ---> CounterClockwise, motor direction = 0
176     * RW_direction == false  ---> Clockwise, motor direction = 1
177     *
178     * */
179    // If motor's z-axis direction is CONTRARY to CubeSat's z-axis direction
180    /**
181     * RW_direction == true  ---> CounterClockwise, motor direction = 1
182     * RW_direction == false  ---> Clockwise, motor direction = 0
183     *
184     * */
```

```
185
186    if (RW_direction) // Reaction Wheel Counter Clock Wise
187    {
188      DriverOne.setDrive(1, 0, New_RW_speed); // Change direction depending on motor ...
          connection
189      RW_speed = New_RW_speed;
190    }
191    else // Reaction Wheel Clock Wise
192    {
193      DriverOne.setDrive(1, 1, New_RW_speed);
194      RW_speed = -New_RW_speed;
195    }
196  }
197
198  void generate_ramp(bool RW_direction, int Acc_ramp_time_duration, int RW_ramp_speed_reach, bool Acc_D
199  {
200    /*
201      Function to generate ramps. In this case it is a Single Impulse.
202      As we dont know the time it lasts, we have to see if speed changes on the ...
          cubesat. That is the use of the while.
203
204      INPUT:
205        RW_direction: Reaction Wheel direction [Clockwise or Counter Clockwise]
206        Acc_ramp_time_duration: Duration of ramp
207        RW_ramp_speed_reach: Final speed reached
208        Acc_Dec_state: Check wether in acceleration state ('1') or deceleration ...
          state ('0')
209      OUTPUT:
210        None
211    */
212
213    Serial1.println("Ramp Start");
214    bool waiting = true;
215
216    set_impulse(RW_direction, RW_ramp_speed_reach);
217
218    Timer1.attachInterrupt(TIMER_CH4, read_show_IMU);
219    waiting = true;
220
221    while (waiting)
222    { // Range of tolerance
223      if (Acc_Dec_state)
224      { // If accelerating
225        if (abs(IMU_gyro_data_Z) > Gyro_tolerance)
226        { // If gyro is not 0 or so, means it has constant speed of rotation
227          if (abs(IMU_accel_data_X) < Accel_tolerance)
228          { // Stop of acceleration
229            waiting = false;
230          }
231        }
232      }
233      else
234      { // If decelerating
235        if (abs(IMU_gyro_data_Z) < Gyro_tolerance)
236        { // If gyro is not 0 or so, means it has constant speed of rotation
237          if (abs(IMU_accel_data_X) < Accel_tolerance)
238          { // Stop of acceleration
239            waiting = false;
```

```
240            }
241          }
242        }
243        delay(1); // Delay for ensuring getting inside the while loop
244      }
245    waiting = true;
246    Timer1.detachInterrupt(TIMER_CH4);
247  }
248
249  // ...

        IMU Functions
250  void read_IMU()
251  {
252    /*
253      Function to read IMU_Data (and refreshes the global variables for IMU data).
254
255      INPUT:
256      None
257      OUTPUT:
258      None, but saves IMU variables and Roll, Pitch and Yaw
259    */
260
261    // Begin reading IMU and pitch, yaw, roll
262    IMU.readSensor();
263
264    // Accelerations
265    IMU_accel_data_X = IMU.getAccelX_mss();
266    IMU_accel_data_Y = IMU.getAccelY_mss();
267    IMU_accel_data_Z = IMU.getAccelZ_mss();
268    // Gyroscope
269    IMU_gyro_data_X = (IMU.getGyroX_rads() * Rad_to_deg);
270    IMU_gyro_data_Y = (IMU.getGyroY_rads() * Rad_to_deg);
271    IMU_gyro_data_Z = (IMU.getGyroZ_rads() * Rad_to_deg);
272    //Magnetometer
273    IMU_mag_data_X = IMU.getMagX_uT();
274    IMU_mag_data_Y = IMU.getMagY_uT();
275    IMU_mag_data_Z = IMU.getMagZ_uT();
276
277    // GetTime of IRS, in case value is correct if timer definition changes.
278    // The dT is the time step of each interrupt, thus, each read of the IMU
279    // First division of clock by preescaler (frequency of timer), then inversion ...
         (period of timer), then multiplication by overflow (period of channel)
280    ...
         float dT = ((1 / ((float)STM32_CLOCK / (float)Timer1.getPrescaleFactor())) * Timer1.getOverflow()
         dT in seconds
281
282    // Get the angle Pitch and Roll angle(w = w0 + dT*angular_vel)
283    Gyro_pitch_deg += dT * (IMU.getGyroY_rads()) * Rad_to_deg;
284    Gyro_roll_deg += dT * (IMU.getGyroX_rads()) * Rad_to_deg;
285
286    // Gimbal lock compensation
287    Gyro_pitch_deg = Gyro_pitch_deg + Gyro_roll_deg * sin(dT * (IMU.getGyroZ_rads()));
288    Gyro_roll_deg = Gyro_roll_deg - Gyro_pitch_deg * sin(dT * (IMU.getGyroZ_rads()));
289
290    // Modulus of the acceleration vector, calculate the total (3D) vector
291    ...
         Accel_total_vector_modulus = sqrt((IMU.getAccelX_mss() * IMU.getAccelX_mss()) + (IMU.getAccelY_ms
```

```
292    ...
         Accel_pitch_deg = asin((float)IMU.getAccelX_mss() / Accel_total_vector_modulus) * Rad_to_deg; //
         Calculate the pitch angle from accelerometer
293    ...
         Accel_roll_deg = asin((float)IMU.getAccelY_mss() / Accel_total_vector_modulus) * Rad_to_deg;  //
         Calculate the roll angle from accelerometer
294
295    /* Accelerometer calibration */
296
297    // To calibrate the accelerometer, put the values    0 . 0  in the variables and ...
         uncomment the 3 Serial1.prints
298    // When compiling, leave the IMU immobile so that the accelerometer calibrates ...
         properly.
299    // Once the values     a r e  obtained, they are noted and it is recompiled as it ...
         had been before.
300
301    Accel_pitch_deg -= -11.17;
302    Accel_roll_deg -= -0.45;
303
304    //    Serial1.print(Accel_pitch_deg,6);
305    //    Serial1.print("\t");
306    //    Serial1.println(Accel_roll_deg,6);
307
308    // If gyroscope and accelerometer are synchronized
309    if (Gyro_Accel_sync)
310    {
311      //  ----- Gyro & accel have been synchronized
312      Gyro_pitch_deg = Gyro_pitch_deg * 0.95 + Accel_pitch_deg * 0.05; // Correct ...
         the drift of the gyro pitch angle with the accelerometer pitch angle. ...
         Original values were 0.9996 and 0.0004m, respectively.
313      Gyro_roll_deg = Gyro_roll_deg * 0.95 + Accel_roll_deg * 0.05;    // Correct ...
         the drift of the gyro roll angle with the accelerometer roll angle
314    }
315    else
316    { //  The 0.98 and 0.02 values are taken from http:// ...
         www.pieter-jan.com/node/11else{
317      //  ----- Synchronize gyro & accel
318      Gyro_pitch_deg = Accel_pitch_deg; // Set the gyro pitch angle equal to the ...
         accelerometer pitch angle
319      Gyro_roll_deg = Accel_roll_deg;   // Set the gyro roll angle equal to the ...
         accelerometer roll angle
320      Gyro_Accel_sync = true;           // Set the IMU started flag
321    }
322
323    // Final corrected Pitch and Roll angle (degrees)
324    Pitch_deg = Pitch_deg * 0.9 + Gyro_pitch_deg * 0.1;
325    Roll_deg = Roll_deg * 0.9 + Gyro_roll_deg * 0.1;
326
327    // Final corrected Pitch and Roll angle (radians)
328    Pitch_rad = -Roll_deg * Deg_to_rad;
329    Roll_rad = Pitch_deg * Deg_to_rad;
330
331    // Calculated values of Magnetometer along X and Y axis
332    ...
         Mag_X_calc = IMU.getMagX_uT() * cos(Pitch_rad) + IMU.getMagY_uT() * sin(Roll_rad) * sin(Pitch_rad
333    Mag_Y_calc = IMU.getMagY_uT() * cos(Roll_rad) + IMU.getMagZ_uT() * sin(Roll_rad);
334
335    // Correct drif (same as Pitch_deg)
```

```
336    Mag_X_corrected = Mag_X_corrected * 0.9 + Mag_X_calc * 0.1;
337    Mag_Y_corrected = Mag_Y_corrected * 0.9 + Mag_Y_calc * 0.1;
338
339    // The magnetic north has been modified, originally, it was from X to Y (towards ...
           which the reading 0 was 90 degrees offset) and without the negative sign of ...
           the beginning (towards which the reading would add the angles ...
           counterclockwise instead of clockwise).
340
341    Yaw_deg = -atan2(Mag_Y_corrected, Mag_X_corrected) * Rad_to_deg;
342
343    // If you want to be precise, add the declination of the geographical place ...
           where you are (https: // www.ign.es/web/gmt-declinacion-magnetica)
344    // Yaw_deg += Declination;
345
346    // The next two lines contain the value of Yaw_deg between 0 and 360 degrees.
347    if (Yaw_deg < 0)
348      Yaw_deg += 360;
349    if (Yaw_deg >= 360)
350      Yaw_deg -= 360;
351  }
352
353  void show_IMU()
354  {
355    /*
356      Function to show IMU_Data on the Serial1 Monitor
357
358      INPUT:
359      None
360      OUTPUT:
361      None
362    */
363
364    // Print accelerometer values
365    //    Serial1.print("Ax: ");
366    Serial1.print(IMU_accel_data_X, 4);
367    Serial1.print(';');
368    Serial1.print(IMU_accel_data_Y, 4);
369    Serial1.print(';');
370    //   Serial1.print(IMU_accel_data_Z,4);
371    //   Serial1.println("");
372
373    // Print Gyro values
374    // Serial1.print(" / G: ");
375    // Serial1.print(IMU_gyro_data_X,4);  Serial1.print(';');
376    // Serial1.print(IMU_gyro_data_Y,4);  Serial1.print(';');
377    Serial1.print(IMU_gyro_data_Z, 4);
378    Serial1.print(';');
379    // Serial1.println("");
380
381    // Print Mag values
382    //    Serial1.print("MAGS: ");
383    //    Serial1.print(IMU_mag_data_X,4);     Serial1.print(';');
384    //    Serial1.print(IMU_mag_data_Y,4);     Serial1.print(';');
385    //    Serial1.print(IMU_mag_data_Z,4);
386    //    Serial1.println("");
387
388    // Print orientation angles
389    //  Serial1.print(" / D: ");
```

```
390    //   Serial1.print(Pitch_deg, 4);
391    //   Serial1.print(';');
392    //   Serial1.print(Roll_deg, 4);
393    //   Serial1.print(';');
394    Serial1.print(Yaw_deg, 4);
395    Serial1.println("");
396  }
397
398  void read_show_IMU()
399  {
400    /*
401      Function to both read and show IMU_Data.
402
403      INPUT:
404      None
405      OUTPUT:
406      None, but saves IMU variables and Roll, Pitch and Yaw
407    */
408
409    read_IMU();
410    show_IMU();
411  }
412
413  // ...

        PID Functions
414  void computePID()
415  {
416    /*
417      Function to calculate PID
418
419      INPUT:
420      None
421      OUTPUT:
422      None, but saves PID values
423    */
424
425    // Time step
426    ...
        float dT = ((1 / ((float)STM32_CLOCK / (float)Timer1.getPrescaleFactor()))) * Timer1.getOverflow()
        in seconds
427
428    read_show_IMU();
429
430    Current_Yaw_deg = Yaw_deg;
431
432    // Check if PID passes through 0 degrees
433    if (Zero_state)
434    {
435      // Saves if the pid pass through 0, if it does it moves the zone away from 0
436      // New variable to not modify the Yaw_deg value
437      Current_Yaw_deg = Yaw_deg + 180; // Addition of 180 deg to all values. The ...
        error is a substract, so the difference is the same
438    }
439
440    if (Current_Yaw_deg >= 360)
441      Current_Yaw_deg -= 360; // One of them will increase over 360, it is a correction
442
```

```
443    // Percentage
444    volatile float Yaw_deg_perc = (Current_Yaw_deg - 360) / (-360) * 100;
445
446    // Transform angle to percentage (Deg_to_reach == setPoint)
447    volatile float Deg_to_reach_perc = (Deg_to_reach - 360) / (-360) * 100;
448
449    // Errors
450    PID_error = Deg_to_reach_perc - Yaw_deg_perc;        // Calculate error ...
          (Proportional)
451    PID_cumulative_error += PID_error * dT;              // Calculate the cumulative ...
          error (Integral)
452    PID_rate_error = (PID_error - PID_last_error) / dT; // Calculate the rate of ...
          error (Derivative)
453
454    // PID Control
455    float PID_P = kp * PID_error;             // Proportional
456    float PID_I = ki * PID_cumulative_error; // Integral
457    float PID_D = kd * PID_rate_error;       // Derivative
458
459    /*
460      // Limit
461      // Ensure not overflow
462      if (PID_P > 255)
463        PID_P = 255;
464      if (PID_P < -255)
465        PID_P = -255;
466      if (PID_I > 255)
467        PID_I = 255;
468      if (PID_I < -255)
469        PID_I = -255;
470      if (PID_D > 255)
471        PID_D = 255;
472      if (PID_D < -255)
473        PID_D = -255;
474    */
475
476    PID_output = PID_P + PID_I + PID_D; //  PID control
477
478    // Prevent overflow
479    if (PID_output > 255)
480      PID_output = 255;
481    if (PID_output < -255)
482      PID_output = -255;
483
484    bool RW_direction = true; // true=positive (CounterClockwise), false=negative ...
          (Clockwise)
485
486    if (PID_output < 0)
487    {
488      PID_output = -PID_output;
489      RW_direction = false;
490    }
491
492    set_impulse(RW_direction, PID_output);
493
494    // Save current error and time for next iteration
495    PID_last_error = PID_error; // Save current error
496  }
```

```
497
498  // ...

         MODES OF OPERATION FUNCTIONS
499  void mode_OBC_Input_Wait()
500  {
501    /*
502      0. Default mode, OBC Reading
503
504      INPUT:
505      None
506      OUTPUT:
507      None, but exits to mode_Select
508    */
509    if (Stop_state != 0)
510    {
511      Stop_state = 0;
512      Timer1.detachInterrupt(TIMER_CH3);
513    }
514    OBC_mode_value = 0;
515    Timer1.attachInterrupt(TIMER_CH3, OBC_mode_receive);
516    while (OBC_mode_value == 0)
517    {
518      delay(1); // If not used the while function does not work
519      // it can be added more conditions to evade being blocked until a data is ...
             received.
520      // for example, it could function an interrupt with a forced exit and an if ...
             after or something
521    }
522    Timer1.detachInterrupt(TIMER_CH3);
523    Serial1.println(OBC_mode_value);
524    mode_Select(OBC_mode_value);
525  }
526
527  void mode_Select(int mode_value)
528  {
529    /*
530      Function to select Mode
531
532      INPUT:
533      mode_value ['0': Waiting; '1': Positioning; '2': Reading]
534      OUTPUT:
535      None, but exits to selected mode
536    */
537
538    switch (mode_value)
539    {
540    default: // Mode OBC Input Waiting (0): Waiting for OBC
541      Serial1.println("Reading mode from OBC");
542      mode_OBC_Input_Wait();
543      break;
544    case 1: // Mode Positioning RW only
545      Serial1.println("Mode Positioning");
546      mode_Positioning_RW();
547      break;
548    case 2: // Mode IMU reading (TEST mode)
549      Serial1.println("Reading IMU");
550      mode_IMU_reading();
```

```
551      break;
552    case 3: // Mode Motor ON OFF (TEST mode)
553      Serial1.println("Mode motor ON OFF");
554      mode_motor_on_off();
555      break;
556    }
557  }
558
559  void mode_Positioning_RW()
560  {
561    /*
562      1. Mode Positioning (RW only)
563
564      INPUT:
565      None
566      OUTPUT:
567      None, but exits to suitable positioning type
568    */
569
570    OBC_mode_value = 0;
571    Serial1.println("Positioning begin");
572    // Insert code to select if the positioning will be coarse (impulses) or fine (PD)
573    /* It should be something like:
574  * read IMU degree value
575  * get OBC value from OBC and save in global variable
576  * comparison to OBC value (wanted position), add to the void function definition
577  * if (comparison<acceptable_error){
578  * positioning_fine();
579  * }else{
580  * positioning_coarse();
581  * }
582  */
583    read_IMU();
584    Serial1.print("Axis Z position: ");
585    Serial1.println(Yaw_deg);
586    Serial1.println("Insert degree value to turn");
587    OBC_data_value = 0;
588    Timer1.attachInterrupt(TIMER_CH3, OBC_data_receive);
589    while (OBC_data_value == 0)
590    {
591      delay(1); // If not used the while function does not work
592    }
593    Timer1.detachInterrupt(TIMER_CH3);
594
595    Serial1.print("Value to turn: "); // In this case turn, but can be what is needed
596    Serial1.print(OBC_data_value);
597
598    // Initiate EmergencyStop routine
599    Timer1.attachInterrupt(TIMER_CH3, EmergencyStop);
600
601    // Depend on the tolerance enable Fine or Coarse positioning
602    if (abs(OBC_data_value) <= Pointing_mode_tolerance)
603    {
604      positioning_Fine();
605    }
606    else
607    {
608      positioning_Coarse();
```

```
609     }
610   }
611
612   void positioning_Coarse()
613   {
614     /*
615       1.1. Mode Positioning Coarse: For now simple
616
617       INPUT:
618       None
619       OUTPUT:
620       None, but exits to positioning_Fine or mode_select
621     */
622
623     Serial1.println("Mode Positioning Coarse");
624
625     // Stores a new variable to not overwrite the original value, also for motor as ...
            it needs a positive value and direction
626     float degree_turn_value = OBC_data_value;
627     bool RW_direction = true; // true=positive (CCW), false=negative (CW) along ...
          CubeSat's z-axis
628                             // In case OBC_data_value is positive, value remains ...
          the same and rw direction is true, the default value of the variable
629                             // In case OBC_data_value is negative, value change to ...
          positive and rw direction is false
630
631     if (degree_turn_value < 0)
632     {
633       degree_turn_value = -degree_turn_value;
634       RW_direction = false;
635     }
636
637     ...
          float Initial_IMU_degree_value, Final_IMU_degree_value, Delta_degree_ramp, Degree_stop_wait;
638     float Prev_Yaw_deg, Check_Yaw_deg;
639     int overlap_count = 0;
640     bool waiting = true;
641
642     // ---------------------------------------------------------ACCELERATION
643
644     read_show_IMU();
645     Initial_IMU_degree_value = Yaw_deg; // Stores initial degree, only in Z
646     int initial_RW_speed = RW_speed;
647     generate_ramp(RW_direction, 0, 30, 1);
648     // 0 will not be used as we dont define the time the motor lasts to do an ...
          impulse. 255 is the max value
649     // This values will be substituted by how we want the time and speed to reach in ...
          the ramp.
650     read_show_IMU();
651     Final_IMU_degree_value = Yaw_deg; // Stores final degree, only in Z
652
653     Delta_degree_ramp = Final_IMU_degree_value - Initial_IMU_degree_value; // degree ...
          turnt on acc, only in Z.
654
655     if (RW_direction == false) // RW Clockwise
656     {
657       Delta_degree_ramp = Final_IMU_degree_value - Initial_IMU_degree_value; // ...
          degree turnt on acc, only in Z. Should be positive
```

271

```
658    }
659
660    if (RW_direction)
661    {
662      Delta_degree_ramp = Final_IMU_degree_value - Initial_IMU_degree_value; // ...
         degree turnt on acc, only in Z. Should be positive
663    }
664    else
665    {
666      Delta_degree_ramp = -(Final_IMU_degree_value - Initial_IMU_degree_value); // ...
         degree turnt on acc, only in Z. Should be positive
667    }
668
669    if (Delta_degree_ramp < 0)
670    {
671      Delta_degree_ramp += 360; // In case its negative adds 360
672      // Negative cases: changes goes by 0 . EX: 330 to 30 when CCW (should be 60 ...
         but calculus is 30-330= -300)
673      //                                   EX: 30 to 330 when CW  (should be 60 ...
         but calculus is 30-330= -300)
674    }
675    //   Serial1.print("ID: ");
676    //   Serial1.print(Initial_IMU_degree_value);
677    //   Serial1.print(" / T: ");
678    //   Serial1.print(Delta_degree_ramp); // Difference of ange
679    //   Serial1.print(" / ED: ");
680    //   Serial1.println(Final_IMU_degree_value); // Final, End angle
681    // -----------------------------------------------------------WAITING
682    if ((degree_turn_value - 2 * Delta_degree_ramp) > 0) // if it is <0, skip the ...
         waiting phase, deceleration must be done immediately after, and still it ...
         would be too much turn.
683    {
684      waiting = true;
685      if (RW_direction)
686      {
687        // CASE CCW
688
689        /*
690        RW CCW
691        CB CW
692        IMU positive
693
694        Angulo inicial 300
695
696        gira 150   en sentido CB CW = degree_turn_value
697
698        angulo final acc 350   = Final_IMU_degree_value
699
700        Delta_degree_ramp=350-300=50
701
702        Degree_stop_wait=350+(150-2*50)=400;  40
703
704        */
705        ...
         Degree_stop_wait = Final_IMU_degree_value + (degree_turn_value - 2 * Delta_degree_ramp); // ...
         Get value of degree to start stop
706
707        Timer1.attachInterrupt(TIMER_CH4, read_show_IMU);
```

```
708        Serial1.println("Waiting");
709         while (waiting)
710         { // Stays as long as waiting is true.
711
712           // CAUTION WITH READING VALUES; AS IT IS ALWAYS FROM 0 TO 360
713           //  Degree_stop_wait will always be > Yaw_deg. If Yaw_deg>> (ex: 359  ),  ...
        Degree_stop_wait can be >360. Thus the value of If Yaw_deg would never reach ...
        Degree_stop_wait
714           //  Degree_stop_wait cant be decreased, as the way to check if its reached ...
        is by a greater. If it is decreased by 360   (so it stays in relative place), ...
        it could be < Yaw_deg and would immediately exit the while without waiting.
715           //  the way to do is check if there is a heavy change on Yaw_deg (pass on ...
        0) to check if adding or substracting a lap, making it go out of the 0 and ...
        360 range.
716           // It should not be a high acceleration enough to make a jump of degree of ...
        180   in such short time,so it should be fine.
717           // Caution disconnection from IMU, could give a false jump, that is why a ...
        check on Yaw_deg first.
718
719           if (Yaw_deg < 0.0001 && Yaw_deg > -0.0001)
720           { // if it is almost exactly 0 then most probably there is a ...
        disconnection. Close numbers should not trigger it.
721           }
722           else if (Yaw_deg - Prev_Yaw_deg < -180)
723           {                   // Checks for a heavy change (greater than half ...
        turn). Done by changes in 0, like jumping from 359 to 0.
724             overlap_count += 1; // adds a lap
725           }
726           else if (Yaw_deg - Prev_Yaw_deg > +180)
727           {                   // Checks for a heavy change (greater than half ...
        turn). Done by changes in 0, like jumping from 0 to 359. (Not possible in ...
        theory, as it increases, but deviations could mess it up)
728             overlap_count -= 1; // substracts a lap
729           }                     // if neither are triggered, change has been samall ...
        or no change has been done yet
730         Prev_Yaw_deg = Yaw_deg;
731         Check_Yaw_deg = Yaw_deg + (360 * overlap_count); // Rewriting of value, in ...
        new variable so it does not add itself.
732         if (Check_Yaw_deg > Degree_stop_wait)
733         {                   // As it is CCW, degree increases. When reading is > to ...
        stop value, exits the while
734           waiting = false; // exit condition
735         }
736         delay(1); // To solve errors
737         // No writing of read_IMU as it is already done by a timer. Just to remind ...
        Yaw_deg is constantly reading values.
738       }
739     Timer1.detachInterrupt(TIMER_CH4);
740   }
741   else
742   {
743     // CASE CW
744     // if its <0, it must be done immediately after, and still it would be too ...
        much turn.
745       ...
        Degree_stop_wait = Final_IMU_degree_value - (degree_turn_value - 2 * Delta_degree_ramp); // ...
        Get value of degree to start stop
746
```

```
747        Timer1.attachInterrupt(TIMER_CH4, read_show_IMU);
748        while (waiting)
749        { // Stays as long as waiting is true.
750          Serial1.println("Waiting");
751
752          // CAUTION WITH READING VALUES; AS IT IS ALWAYS FROM 0 TO 360
753          //  Degree_stop_wait will always be < Yaw_deg. If Yaw_deg<< (ex: 1  ),  ...
      Degree_stop_wait can be <0. Thus the value of If Yaw_deg would never reach ...
      Degree_stop_wait
754          //  Degree_stop_wait can't be increased, as the way to check if its ...
      reached is by a greater. If it is increased by 360  (so it stays in relative ...
      place), it could be > Yaw_deg and would immediately exit the while without ...
      waiting.
755          //  the way to do is check if there is a heavy change on Yaw_deg (pass on ...
      0) to check iff adding or substracting a lap, making it go out of the 0 and ...
      360 range.
756          // It should not be a high acceleration enough to make a jump of degree of ...
      180  in such short time, so it should be fine.
757          // Caution disconnection from IMU, could give a false jump, that is why a ...
      check on Yaw_deg first.
758
759          Check_Yaw_deg = Yaw_deg; // New variable so it does not change during an ...
      iteration
760          if (Check_Yaw_deg < 0.0001 && Check_Yaw_deg > -0.0001)
761          { // if it is almost exactly 0 then most probably there is a ...
      disconnection. Close numbers should not trigger it.
762          }
763          else if (Check_Yaw_deg - Prev_Yaw_deg > +180)
764          {                       // Checks for a heavy change (greater than half ...
      turn). Done by changes in 0, like jumping from 359 to 0. (Not possible in ...
      theory, as it decreases, but deviations could mess it up)
765            overlap_count += 1; // adds a lap
766          }
767          else if (Check_Yaw_deg - Prev_Yaw_deg < -180)
768          {                       // Checks for a heavy change (greater than half ...
      turn). Done by changes in 0, like jumping from 0 to 359.
769            overlap_count -= 1; // substracts a lap
770          }                       // if neither are triggered, change has been samall ...
      or no change has been done yet
771
772          Prev_Yaw_deg = Check_Yaw_deg;
773          Check_Yaw_deg = Check_Yaw_deg + (360 * overlap_count); // Rewriting of ...
      value. It does not add itself as it gets the read value in each iteration. In ...
      other words, it will not go through the iteration with numbers outside 0 and ...
      360 range
774          if (Check_Yaw_deg < Degree_stop_wait)
775          {                       // As it is CCW, degree increases. When reading is > to ...
      stop value, exits the while
776            waiting = false; // exit condition
777          }
778          delay(1); // To solve errors
779          // No writing of read_IMU as it is already done by a timer. Just to remind ...
      Yaw_deg is constantly reading values.
780        }
781        Timer1.detachInterrupt(TIMER_CH4);
782      }
783    }
784
```

```
785    // ---------------------------------------------------------DECELERATION
786    //  Serial1.print("ID: ");
787    //  Serial1.print(Final_IMU_degree_value);
788    //  Serial1.print(" / ED: ");
789    //  Serial1.println(Degree_stop_wait);
790
791    // Initial_RW_Speed returns speed to original value instead of 0
792    generate_ramp(RW_direction, initial_RW_speed, 0, 0);
793    // This is to assure the impulse is the same, as if there is some momentum ...
          accumulation, returning to 0 would be a different impulse
794    // 0 will not be used as we dont define the time the motor lasts to do an ...
          impulse. 255 is the max value
795    // This values will be substituted by how we want the time and speed to reach in ...
          the ramp.
796
797    read_show_IMU();
798    Final_IMU_degree_value = Yaw_deg; // can be overwritten, final degree of manoeuvre
799    if (RW_direction)
800    {
801      Delta_degree_ramp = Final_IMU_degree_value - Initial_IMU_degree_value; // ...
          Should be positive
802    }
803    else
804    {
805      Delta_degree_ramp = Initial_IMU_degree_value - Final_IMU_degree_value; // ...
          Should be positive
806    }
807    if (Delta_degree_ramp < 0)
808    {                              // can be overwritten, real turn of manoeuvre
809      Delta_degree_ramp += 360; // In case its negative adds 360
810      // Negative cases: changes goes by 0  .  EX: 330 to 30 when CCW (should be 60 ...
          but calculus is 30-330= -300)
811      //                                 EX: 30 to 330 when CW  (should be 60 ...
          but calculus is 30-330= -300)
812    }
813
814    //  Serial1.print("ID: ");
815    //  Serial1.print(Degree_stop_wait);
816    //  Serial1.print(" / T: ");
817    //  Serial1.print(Delta_degree_ramp);
818    //  Serial1.print(" / ED: ");
819    //  Serial1.println(Final_IMU_degree_value);
820
821    ...
          if ((Delta_degree_ramp - degree_turn_value) < Final_pointing_tolerance && (Delta_degree_ramp - de
822    {
823      // Detach EmergencyStop
824      Timer1.detachInterrupt(TIMER_CH3);
825
826      // Real turn vs wanted turn, checks if it is considered good
827      mode_OBC_Input_Wait(); // Valid position
828    }
829    else
830    {
831      // Recalculate the OBC_data_value for PID
832      OBC_data_value = OBC_data_value - Delta_degree_ramp;
833      positioning_Fine(); // Correction
834    }
```

```
835  }
836
837  void positioning_Fine()
838  {
839    /*
840      1.2. Mode Positioning Fine: PID
841
842      INPUT:
843      None
844      OUTPUT:
845      None, but exits to mode_select
846    */
847
848    Serial1.println("Mode Positioning Fine");
849
850    bool waiting = true;
851
852    read_show_IMU();
853    Deg_to_reach = OBC_data_value + Yaw_deg; // Get value to reach, contained in 360
854    if (Deg_to_reach < 0)
855    {
856      Deg_to_reach += 360;
857      Zero_state = true; // Used to store if it passes 0.
858      // A PD passing through 0 could give great problems, as it has a very big ...
           change in value. Further used in computePID()
859    }
860    else if (Deg_to_reach >= 360)
861    {
862      Deg_to_reach -= 360; // These two lines contain the value of the Yaw_deg in ...
           0-360 degrees.
863      Zero_state = true;
864    }
865    else
866    {
867      Zero_state = false;
868    }
869
870    if (Zero_state)
871    {
872      Deg_to_reach + 180;
873      if (Deg_to_reach >= 360)
874        Deg_to_reach -= 360;
875    }
876
877    Timer1.attachInterrupt(TIMER_CH4, computePID);
878    while (waiting)
879    { // Range of tolerance
880      ...
         if (abs(IMU_gyro_data_Z) < Gyro_tolerance && abs(IMU_accel_data_X) < Accel_tolerance && abs(Deg_t
881      { // we consider it is stopped, modify values to be accurate
882        waiting = false;
883      }
884      delay(1); // Change
885    }
886    waiting = true;
887    Timer1.detachInterrupt(TIMER_CH4);
888
889    // At exit, RW_speed could not be 0
```

```
890    OBC_data_value = 0;
891    Serial1.println("End of manoeuvre");
892
893    // Detach EmergencyStop
894    Timer1.detachInterrupt(TIMER_CH3);
895
896    mode_Select(OBC_mode_value);
897  }
898
899  void mode_IMU_reading()
900  {
901    /*
902      2- IMU reading (TEST PURPOSE Function)
903
904      INPUT:
905      None
906      OUTPUT:
907      None, but saves IMU variables and Roll, Pitch and Yaw
908    */
909
910    OBC_mode_value = 0;
911    Timer1.attachInterrupt(TIMER_CH4, read_show_IMU);
912    Timer1.attachInterrupt(TIMER_CH3, OBC_data_receive);
913    while (OBC_data_value == 0)
914    {
915      delay(1); // If not used the while function does not work
916    }
917    Timer1.detachInterrupt(TIMER_CH3);
918    Timer1.detachInterrupt(TIMER_CH4);
919    OBC_data_value = 0;
920    mode_Select(OBC_mode_value);
921  }
922
923  void mode_motor_on_off()
924  {
925    /*
926      3- Motor ON/OFF (TEST PURPOSE Function)
927
928      INPUT:
929      None
930      OUTPUT:
931      None
932    */
933    OBC_mode_value = 0;
934    set_impulse(true, 100);
935    Timer1.attachInterrupt(TIMER_CH3, OBC_data_receive);
936    while (OBC_data_value == 0)
937    {
938      delay(1); // If not used the while function does not work
939    }
940    Timer1.detachInterrupt(TIMER_CH3);
941    set_impulse(true, 0);
942    OBC_data_value = 0;
943    mode_OBC_Input_Wait();
944  }
945
946  // ...
```

```
         VOID SETUP
947  void setup()
948  {
949    delay(1000); //  wait to let open the serial
950    // ...

         Timers
951    Timer1.pause();
952    Timer1.setPrescaleFactor(7200); // 72MHz Clock / 7200 = 10KHz timer
953    Timer1.setOverflow(1000);        // Overflow occurs at 1000, each 100 ms timer ...
         restarts
954
955    Timer1.setMode(TIMER_CH3, TIMER_OUTPUT_COMPARE); // Configure channel to ...
         OUTPUTCOMPARE: Channel for OBC read values
956    Timer1.setMode(TIMER_CH4, TIMER_OUTPUT_COMPARE); // Channel for IMU read values
957    Timer1.setCompare(TIMER_CH3, 1);                 // Phase value in Overflow range
958    Timer1.setCompare(TIMER_CH4, 1);
959
960    Timer1.refresh(); // Refresh timer and start over
961    Timer1.resume();
962
963    // ...

         Initiations
964    Serial1.begin(9600);
965    SPI.begin();
966
967    Serial1.println("START");
968
969    // ...

         Setups
970
971    pinMode(LEDPIN, OUTPUT); // Integrated LED
972    pinMode(CS1, OUTPUT);    // NCS Pin definition
973
974    // ...

         Driver Setup
975    Serial1.println("Configuring Driver...");
976    DriverOne.settings.commInterface = I2C_MODE; // Driver Comm Mode
977    DriverOne.settings.I2CAddress = 0x5D;        // Driver Address (0x5D by Default)
978
979    while (DriverOne.begin() != 0xA9)
980    { // Driver wait for idle
981      Serial1.println("ID Mismatch");
982      delay(200);
983    }
984    Serial1.println("ID Match");
985
986    Serial1.println("Waiting for enumeration"); // Driver wait for peripherals
987    while (DriverOne.ready() == false)
988      ;
989    Serial1.println("Ready");
990
991    while (DriverOne.busy())
992      ; // Driver enable
993    DriverOne.enable();
```

```
994    Serial1.println("Driver Ready to Use!");
995
996    // ...

         IMU Setup
997    Serial1.println("Configuring MPU9250...");
998    int status = IMU.begin();
999    if (status < 0)
1000   {
1001     Serial1.println("IMU initialization unsuccessful");
1002     Serial1.println("Check IMU wiring or try cycling power");
1003     Serial1.print("Status: ");
1004     Serial1.println(status);
1005     while (1)
1006     {
1007     }
1008   }
1009
1010   IMU.setGyroRange(IMU.GYRO_RANGE_500DPS);
1011
1012   // ...

         IMU Calibration
1013   // Use for getting the values      o n  calibration setting below
1014
1015   // To calibrate the magnetometer or the compass, everything is commented except ...
         the last 3 lines.
1016   // It moves in the shape of an eight approximately 2 min. It is recommended to ...
         carry out several times until the measurements are fine-tuned.
1017   // Uncomment everything and enter the values   o b t a i n e d  from the MagBias and ...
         ScaleFactor to view the results of the magnetometer.
1018   //
1019   //       IMU.calibrateMag();
1020   //       Serial1.println("Done");
1021   //
1022   //       Serial1.print(IMU.getMagBiasX_uT());
1023   //       Serial1.print(",");
1024   //       Serial1.print(IMU.getMagBiasY_uT());
1025   //       Serial1.print(",");
1026   //       Serial1.println(IMU.getMagBiasZ_uT());
1027   //
1028   //       Serial1.print(IMU.getMagScaleFactorX());
1029   //       Serial1.print(",");
1030   //       Serial1.print(IMU.getMagScaleFactorY());
1031   //       Serial1.print(",");
1032   //       Serial1.println(IMU.getMagScaleFactorZ());
1033
1034   // Reactioni wheels + Magnetorquer
1035   IMU.setMagCalX(9.42, 0.91); // The first value corresponds to the MagBias, and ...
         the second the ScaleFactor.
1036   IMU.setMagCalY(41.82, 0.98);
1037   IMU.setMagCalZ(-6.59, 1.14);
1038
1039   // Only reaction wheels
1040   //  IMU.setMagCalX(14.06, 0.84); // The first value corresponds to the MagBias, ...
         and the second the ScaleFactor.
1041   //  IMU.setMagCalY(29.85, 1.36);
1042   //  IMU.setMagCalZ(-32.30, 0.94);
```

```
1043
1044    Serial1.println("MPU9250 Ready to Use!");

1045

1046    Serial1.println("Reading mode from OBC");
1047    mode_OBC_Input_Wait();
1048  }

1049

1050  // ...


         VOID LOOP
1051  void loop()
1052  {
1053  }
```

Code O.1 Full control of the Cubesat with coarse and fine pointing mode. Source: Own.

Table O.1 Variables explanation of the control algorithm. Source: Own.

Name	Description	Type	Units	Ambit
LEDPIN	Integrated LED of the Bluepill	#define	-	Bluepill configuration
CS1	STM32 NCS Chip select (SPI mode only) pin	#define	-	Bluepill configuration
STM32_CLOCK	Internal STM32 Clock	#define	kHz	Bluepill configuration
PI	Number PI	#define	-	Math values
Rad_to_deg	Convert radians to degrees	#define	-	Math values
Deg_to_rad	Convert degrees to radians	#define	-	Math values
Final_pointing_tolerancy	Tolerancy for final pointing	#define	Deg (Âž)	Tolerancies
Pointing_mode_tolerancy	Tolerance for selecting pointing mode	#define	Deg (Âž)	Tolerancies
Accel_Tolerance	Tolerance for acceleration measurement	#define	m/s^2	Tolerancies
Gyro_Tolerance	Tolerance for gyro measurement	#define	Deg/s	Tolerancies
DriverOne	Driver Object definition	SCMD	-	Object initiations
IMU	MPU Object definition	MPU9250	-	Object initiations
IMU_accel_data_X	Values of local IMU accelerometer along X axis	float	m/s^2	IMU Values
IMU_accel_data_Y	Values of local IMU accelerometer along Y axis	float	m/s^2	IMU Values
IMU_accel_data_Z	Values of local IMU accelerometer along Z axis	float	m/s^2	IMU Values
IMU_gyro_data_X	Values of local IMU gyroscope along X axis	float	Deg/s	IMU Values
IMU_gyro_data_Y	Values of local IMU gyroscope along Y axis	float	Deg/s	IMU Values
IMU_gyro_data_Z	Values of local IMU gyroscope along Z axis	float	Deg/s	IMU Values
IMU_mag_data_X	Values of local IMU magnetometer along X axis	float	uT	IMU Values
IMU_mag_data_Y	Values of local IMU magnetometer along Y axis	float	uT	IMU Values
IMU_mag_data_Z	Values of local IMU magnetometer along Z axis	float	uT	IMU Values

`Pitch_deg`	Pitch angle	float	Deg (Âž)	Orientation angles
`Roll_deg`	Roll angle	float	Deg (Âž)	Orientation angles
`Yaw_deg`	Yaw angle	float	Deg (Âž)	Orientation angles
`Accel_total_vector_modulus`	Accelerometer total vector magnitude	float	m/s^2	Angles determination
`Accel_pitch_deg`	Accelerometer Pitch angle	float	Deg (Âž)	Angles determination
`Accel_roll_deg`	Accelerometer Roll angle	float	Deg (Âž)	Angles determination
`Gyro_pitch_deg`	Gyroscope Pitch angle	float	Deg (Âž)	Angles determination
`Gyro_roll_deg`	Gyroscope Roll angle	float	Deg (Âž)	Angles determination
`Gyro_Accel_sync`	Check if Gyroscope and Accelerometer are synchronized	bool	-	Angles determination
`Mag_X_calc`	Calculated Magnetometer value along X axis	float	uT	Angles determination
`Mag_Y_calc`	Calculated Magnetometer value along Y axis	float	uT	Angles determination
`Mag_X_corrected`	Corrected Magnetometer value along X axis	float	uT	Angles determination
`Mag_Y_corrected`	Corrected Magnetometer value along Y axis	float	uT	Angles determination
`Pitch_rad`	Pitch angle	float	rad	Angles determination
`Roll_rad`	Roll angle	float	rad	Angles determination
`RW_speed`	Value of Reaction Wheel speed	int	-	Motor values
`OBC_mode_value`	Value of mode sent by the On Board Computer	int	-	OBC readings
`OBC_data_value`	Value of turn sent by the On Board Computer	int	Deg (Âž)	OBC readings
`Stop_State`	Check if an Emergency Stop is performed	int	Deg (Âž)	OBC readings
`PID_error`	Proportional error from PID	double	-	Fine Positioning (PID)
`PID_last_error`	Previous proportional error from PID	double	-	Fine Positioning (PID)
`PID_cumulative_error`	Cumulative (Integral) error from PID	double	-	Fine Positioning (PID)
`PID_rate_error`	Rate (Derivative) error from PID	double	-	Fine Positioning (PID)
`kp`	Proportional constant	double	-	Fine Positioning (PID)
`ki`	Integral constant	double	-	Fine Positioning (PID)

kd	Derivative constant	double	-	Fine Positioning (PID)
Deg_to_reach	Setpoint angle	float	Deg (Âž)	Fine Positioning (PID)
Zero_state	Check if PID encompass yaw of 0 degrees	bool	-	Fine Positioning (PID)
PID_output	Output value of the PID	int	bytes	Fine Positioning (PID)
RW_direction	Direction of the Reaction Wheel	bool	-	(Inside functions) Motor settings
New_RW_speed	Speed to set on the Reaction Wheel	int	bytes	(Inside functions) Motor settings
Acc_ramp_time_duration	Duration of ramp	int	s	(Inside functions) Motor settings
RW_ramp_speed_reach	Final speed reached	int	bytes	(Inside functions) Motor settings
Acc_Dec_state	Check wether in acceleration or deceleration state	bool	-	(Inside functions) Motor settings
dT	Time step of each interrupt, calculated from Timer	float	s	(Inside functions) IMU reading and Fine Positioning (PID)

Table O.2 Functions of the control algorithm. Source: Own.

Name	Description	Input	Output	Ambit
`OBC_mode_receive()`	Get mode from OBC	None, but gets mode from User Serial1 input	None, but saves OBC_mode_value	OBC readings
`OBC_data_receive()`	Get data from OBC	None, but gets mode from User Serial1 input	None, but saves OBC_data_value	OBC readings
`EmergencyStop`	Stops the motor if the user calls an emergency stop	None, but gets stop state from User Serial1 input	None	OBC readings
`set_impulse()`	Set the motor at a fixed direction and speed	RW_direction, New_RW_speed	None, but saves RW_speed	Motor settings
`generate_ramp()`	Creates a ramp	RW_direction, Acc_ramp_time_reach, RW_ramp_speed_reach, Acc_Dec_state	-	Motor settings
`read_IMU()`	Read data from the IMU	None	None, but saves IMU variables, pitch, roll and yaw	IMU reading
`show_IMU()`	Show data from the IMU on the Serial1 monitor	None	None	IMU reading
`read_show_IMU()`	Read then show data from the IMU	None	None, but saves IMU variables, pitch, roll and yaw	IMU reading
`computePID()`	Compute the PID and updates the motor speed and direction	None	None, but saves PID values	Fine Positioning (PID)
`mode_OBC_Input_Wait()`	Default mode, waits for a mode input from OBC	None	None, but exits to `mode_Select()`	Modes
`mode_Select()`	Decides the mode to execute depending on the mode read	mode_value	None, but exits to selected mode	Modes
`mode_Positioning_RW()`	Waits for a data input from OBC and decides the positioning type (coarse or fine)	None	None, but exits to suitable positioning type	Modes

positioning_Coarse()	Executes a coarse pointing control (ramps and waiting)	None	None, but exits to positioning_Fine() or mode_Select()	Modes
positioning_Fine()	Executes a fine pointing control (PID)	None	None, but exits to mode_Select()	Modes
mode_IMU_reading()	Test purpose function to read and show IMU values constantly	None	None, but saves IMU values and exits to mode_Select()	Modes
mode_motor_on_off	Test purpose function to start and stop the motor	None	None, but exits to mode_Select()	Modes