POLITECNICO DI TORINO

MASTER's Degree in ELECTRONIC ENGINEERING

Development and Testing of a Board for the Control of Attitude of Mini Satellites

Supervisors

Prof. Fabrizio STESINA

Candidate

Alessandro CAMPISI

Academic Year 2023/2024

Abstract

The following thesis topic is about the development and testing of a Board working as ADCS for a CubeSat. This project is part of a collaboration between the u3S Laboratory at the University of Bologna and the STAR Laboratory at Politecnico of Turin for a distributed attitude determination and control systems testing environment for CubeSats.

The developed system has the goal of performing a Dethumbling maneuver, controlling the orientation of a 0.5U Cubesat composed of other Sub-Systems like the On-Board Computer and Electrical Power System: it takes operative mode commands from a Ground Station composed of a Personal Computer to reach a certain target in terms of angular speed, drives the actuators and collect data about the internal state to send back to the Ground Station.

The thesis starts with a brief introduction to CubeSat's history of launches, introduces the ADCS systems principles, and provides some information about the standards in space environments. Then, it explains in detail the Cubesat behavior from a system point of view; gives an in-depth focus on hardware description, design, and prototyping with particular attention to the Magnetorquers and Electric Drivers design choices; then moves to a Firmware description of the board.

Finally, it describes the Facility in Forlì of the University of Bologna where the tests were performed. It discusses the setup in preparation for the tests and the final results and achievements obtained.

Acknowledgements

I would like to thank my supervisor, Prof. Fabrizio Stesina for its invaluable advices, patience and for leaving me the freedom to make mistakes and learn new things in a field unknown to me before.

I would also like to thank Prof. Dario Modenini for giving me the opportunity to work in an innovative and inspiring environment like the one of the CubeDyna Facility in Forli at the Department of Industrial Engineering(UniBo) and Dr. Andrea Curatolo for its incredible technical support during the final tests campaign.

Moreover, I would express my gratitude to Edoardo, Simone, and Marco for their massive support during these intense months of work together with the guys from Team Diana and Team Cubesat with which I spent pleasant moments between a coffee break and another.

Lastly, I am extremely grateful to my parents and friends which never failed to give me strength during these years of university.

Table of Contents

List of Tables

List of Figures

Acronyms

CDH
: Command and Data Handling

RF
: Radio Frequency

ADCS
: Attitude and Determination Control System

SBRF
: Satellite's Body Reference Frame

PCB
: Printed Circuit Board

OBC
: On-Board Computer

GS
: Ground Station

PCI
: Peripheral Component Interconnect

IMU
: Inertial Measurement Unit

Dlab
: Distributed Lab

ADC
: Analog to Digital Converter

UART
: Universal Asynchronous receiver/transmitter

USART
: Universal Synchronous/Asynchronous receiver/transmitter

OpMode

Operative Mode

PWM

Pulse-Width Modulation

PID

Proportional Integrative Derivative

SPI

Serial Peripheral Interface

PGA

Programmable Gain Amplifier

LPUART

Low-Power UART

NTC

Negative Temperature Coefficient

QSPI

Quad SPI

AC

Alternate Current

EMI

Electromagnetic Interference

DC

Duty Cycle

MUX

Multiplexer

USB

Universal Serial Bus

CTS

Clear To Send

SMD

Surface Mounted

RTOS

Real Time Operative System

TCB

Task Control Block

RAM

Random Acces Memory

IC

Integrated Circuit

CAN

Controlle Area Network

CPU

Central Processing Unit

HC

Helmotz Cage

CM

Centre of Mass

CR

Centre of Rotation

LEO

Low Earth Orbit

PLA

Polylactic Acid

RMS

Root Mean Square

CTP

Cubesat Test Platform

EEPROM

Electrically Erasable and Programmable Memory

Chapter 1

Introduction to Cubesat

This Chapter is meant to introduce the world of CubeSats and their evolution throughout the years.

1.1 Cubesat Standard

[1],CubeSat is a class of satellites that adopt a standard size and form factor, which unit is defined as "U".
A 1U CubeSat is a 10 cm cube with a mass of up to 2 kg.
This standard primary objective is to provide specifications for the design of CubeSats ranging from 1U to 12U. The standard secondary objective is to provide information on available CubeSat dispensers and their corresponding interfaces.

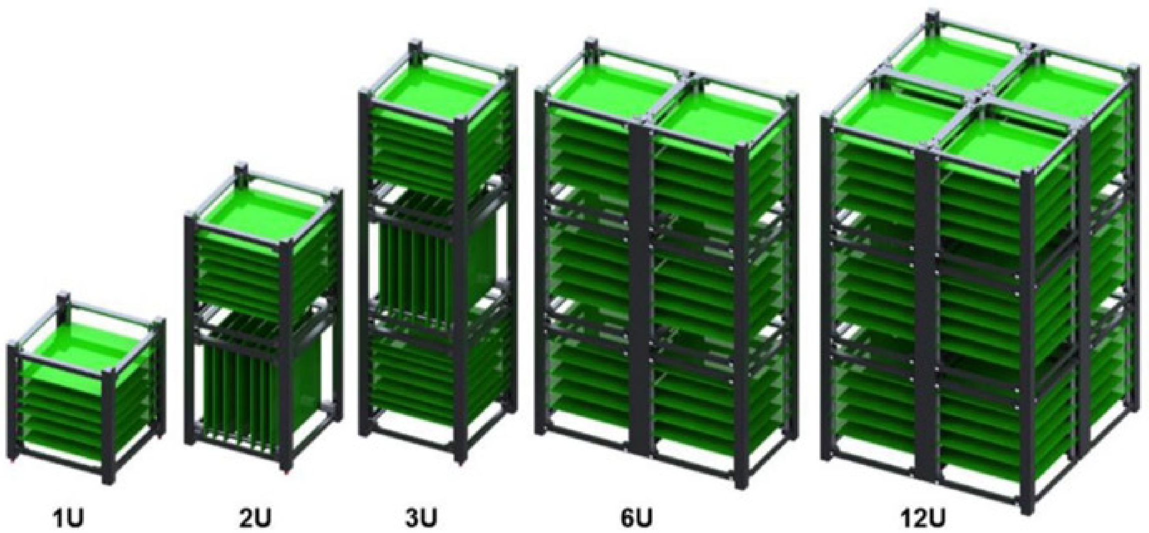


Figure 1.1: The current Cubesat Family

CubeSats are typically built up from standard cubic units each measuring 10 cm x 10 cm x 10 cm. The number of units depends on the CubeSat's mission, but tends to be between 2 and 16, resulting in a mass of just 3–32 kg. These little satellites have a fraction of the mass, and cost, of more traditional satellites.

1.1.1 History of Cubesat in Brief

The **CubeSat Standard** was formed when two professors, Jordi Puig-Suari of California Polytechnic State University and Bob Twiggs of Stanford University, proposed it in 1999 to allow university students to design, build, test, and operate simple spacecraft. The first six Cubesats were later launched in 2003, Russia's Plesetsk launch site and the first NASA CubeSat was GeneSat-1 – launched in 2006. GeneSat-1's success was an important factor in the increase of CubeSat popularity. Until 2013, CubeSats were a small field dominated by academia and university projects.
After 2013, most of the CubeSats launched were for amateur or commercial purposes. As of 2012 only 72 of CubeSats were on orbit, but as of January 2019 over 900 have been successfully deployed on orbit.
Cubesats were made possible by the ongoing miniaturization of electronics, which allows instruments such as cameras to ride into orbit at a fraction of the size of what was required at the beginning of the space age in the 1960s. The applicability of these satellites is being used more and more every year.

[2],From the first launches in the late nineties till nowadays the launch of nanosats in orbit increased significantly. As can be seen in figure 1.2, it increased by almost 800% and the counts for features compared to the total launches reduced a lot to nearly 0 from 2014 till now. As reported on the "Nanosats Database" the number of nanosats and CubeSats launched in total reach 5000 units and the trend is destined to arise in the future, considering for example that the prediction of nanosats launches in 2024 doubles compared to the year 2022.

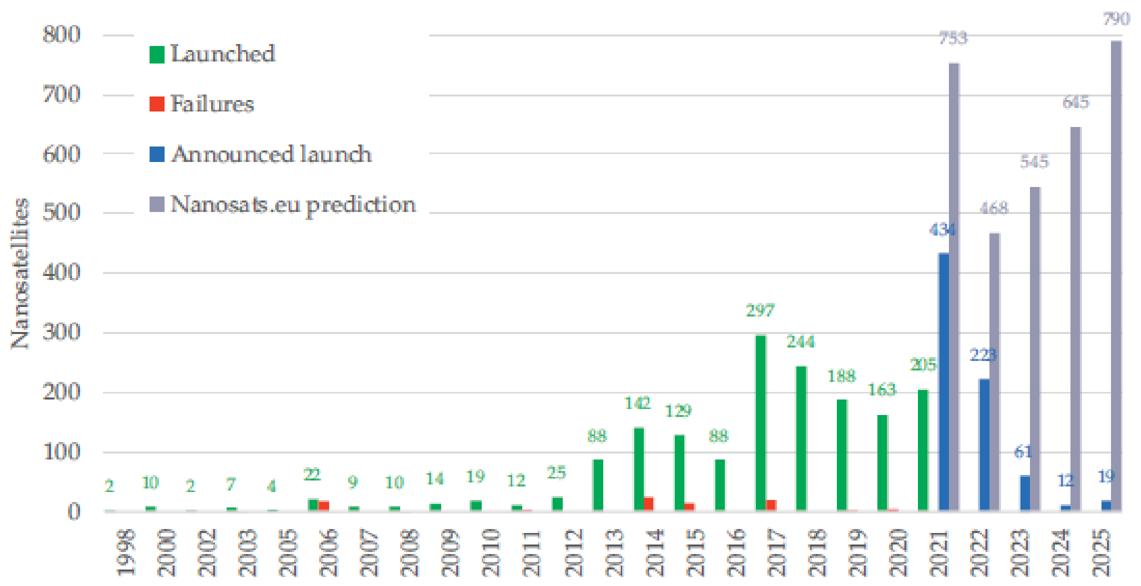


Figure 1.2: Nanosatellite launches and forecasts trend over the years

Of all of these launches the majority of them nowadays(Figure 1.3) are carried out by companies, but from the year 2013 the number of launches made by universities increased and the trend probably will increasingly grow in the next years.

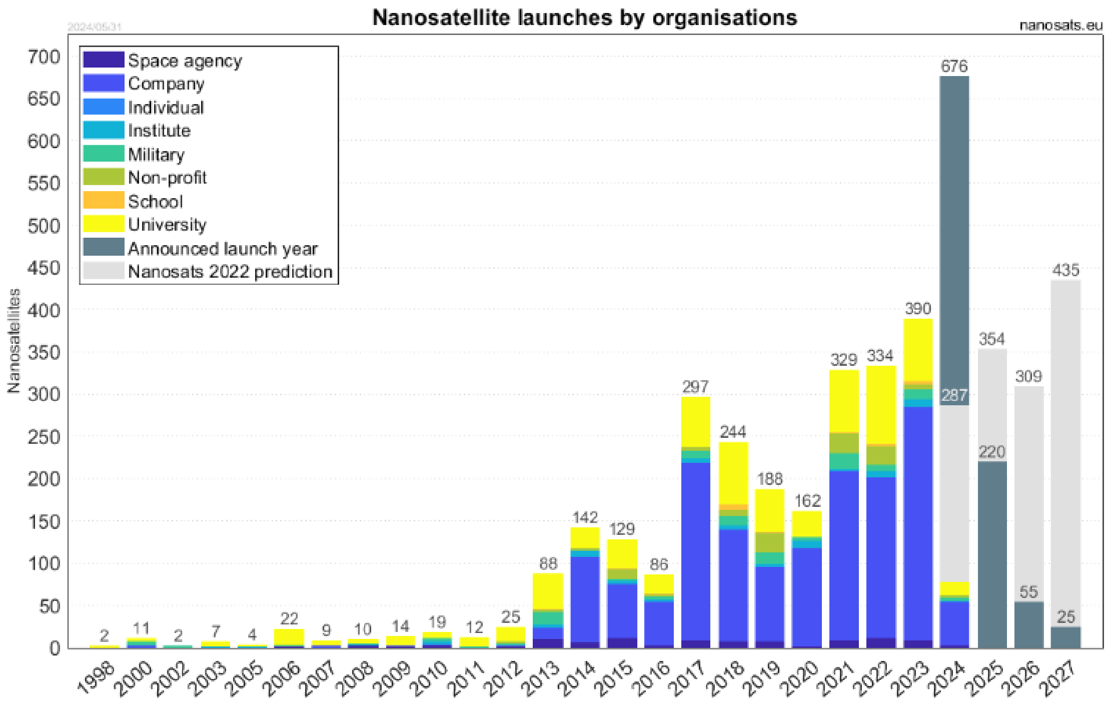


Figure 1.3: Nanosatellite launches by organisations over the years

1.2 Introduction to ADCS

Cubesats are composed of many sub-systems such as the Command and Data Handling(**CDH**), Radio Frequency(**RF**)Communication and the Attitude and Determination Control System (**ADCS**) [3]. Usually, the ADCS incorporates elements that are specific only to this sub-system and not of others such as the processor(microcontroller), sensors, and dedicated actuators. ADCS systems can use various methods for determining attitude, which are most commonly vector observations of particular objects.
Different methods are employed for both determining and controlling a satellite's attitude, the most common of which are listed below:

- **Sun sensors**: are visible-light detectors widely used due to their limited dimension, accuracy and reliability, which are used to estimate the Sun's direction in the Satellite's Body Reference Frame (**SBRF**).
  They can be pretty accurate but it depends on the availability of sunlight in space, for example during eclipses their accuracy reduces a lot. CubeSat projects often use solar panels such as sun sensors: each panel gives a different power output depending on the the incident angle of the sun rays and the sun vector can be reconstructed using trigonometry [4].

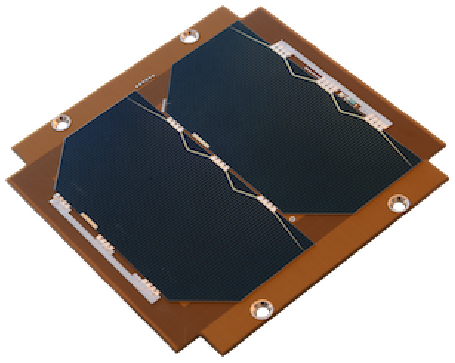


Figure 1.4: Example of Sun Sensor

- **Gyroscope**: a spinning wheel, which retains the orientation of its spinning axis due to the conservation of angular momentum. Modern gyroscopes are lightweight and measure angular velocity and acceleration accurately.
  However, due to axial precession and the fact that the gyroscope does not keep its orientation when not spinning, gyroscopes alone cannot be used to determine a Satellite's Attitude.

- **Magnetic sensors(Magnetometer)**: measure the magnetic field and its strength in three dimensions and therefore return the magnetic field vector. Comparing the magnetic field reading to an onboard Earth's magnetic field model can determine the satellite's attitude. The magnetometer allocation is based on the ADCS design and usually being placed on the system's printed circuit board (PCB) or an external sensor head.

For attitude control, satellites can use two types of actuation::

- Passive : such as **Gravity Booms** or **Permanent Magnets**
- Active : such as **Magnetic Coils(Magnetorquers)**, **Reaction Wheels**(electrical motor),**Thrusters**(fuelled motor) , or **Solar Sails**

1U satellites mainly use active types of actuators, Magnetorquers and Reaction Wheels:

- Magnetorquers create a magnetic dipole that interacts with the Earth's magnetic field and creates a small external force(torque) on the satellite's body.
- Reaction wheels,similar to DC motors in the concept, which can be spun to create an internal force on the satellite's body.
  Concerning the first they can generate quick, sharp changes to the attitude of the CubeSat.

1.3 PCI Standard and Boards Mechanical Dimension

To ensure mechanical compatibility between all electrical boards and with the whole mechanical structure of a CubeSats an industrial standard was developed in 1992 by the PC/104 Consortium : the objective was to incorporate a PCI bus[5] on the PC/104 form factor, which became to be known as PC/104-Plus. This architecture provides a link to versatile legacy hardware while meeting the high-speed requirements for present and future hardware.
Moreover, it provides some guidelines about the mechanical dimensions of boards.

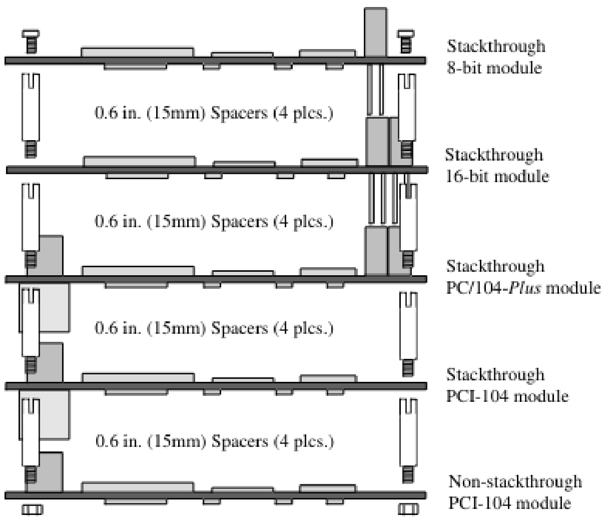

Stackthrough 8-bit module
0.6 in. (15mm) Spacers (4 plcs.)
Stackthrough 16-bit module
0.6 in. (15mm) Spacers (4 plcs.)
Stackthrough PC/104-*Plus* module
0.6 in. (15mm) Spacers (4 plcs.)
Stackthrough PCI-104 module
0.6 in. (15mm) Spacers (4 plcs.)
Non-stackthrough PCI-104 module

Figure 1.5: Module Stack configuration before

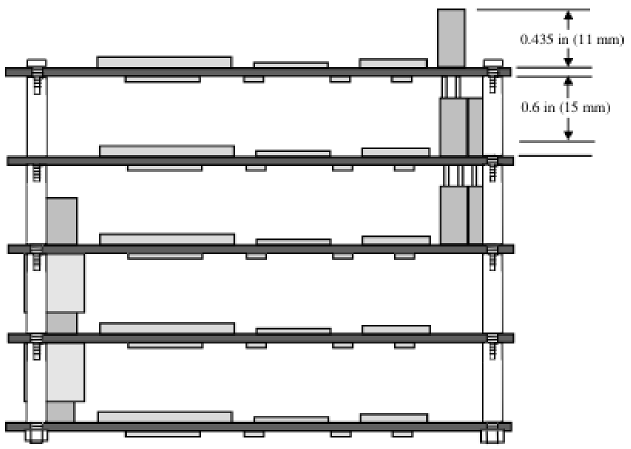


Figure 1.6: Module Stack configuration after

The two figures above (Figure 1.5 and 1.6) show a typical stackup configuration for the board using the PCI-104 standard. To have a standard for the distances in vertical between boards Spacers are used, starting from the bottom they are blocked to the board through a screw, and for the last spacer, it is attached to the board with a bolt.

Chapter 2

System Overview

In this chapter will be presented an overview about the board main elements and their role in the system.

2.1 ADCS Block Scheme

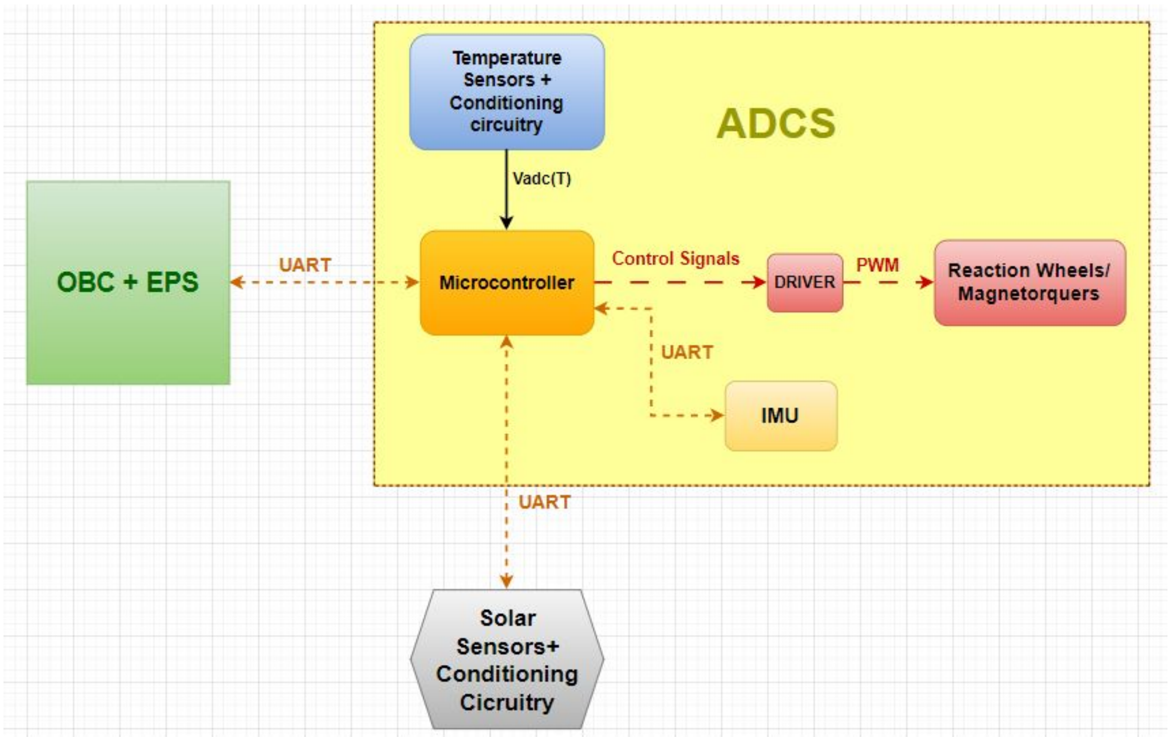

Temperature Sensors + Conditioning circuitry
ADCS
Vadc(T)
OBC + EPS
UART
Microcontroller
Control Signals
DRIVER
PWM
Reaction Wheels/ Magnetorquers
UART
IMU
UART
Solar Sensors+ Conditioning Cicruitry

Figure 2.1: System Block Diagram

As shown in figure 2.1 the systems is composed of:

- A **Microcontroller**, whose role is to take all the information coming from sensors to know the actual position in the space and speed of the satellite, compare them with the desired position and speed and provide an output to actuators, which will be the ones that will correct the trajectory. In this sense the microcontroller is the main part of the Control System;
- **Temperature sensors and conditioning circuitry** which will be used to monitor the the temperature of actuators ic Drivers, solar sensors, and the microcontroller processor;

- **Solar Sensors** which will be used to acknowledge the position of the Sun in space;
- **Reaction wheels**, one of the two types of actuators. These kinds of actuators are responsible for the propulsion of the satellite in space;
- A **Magnetorquer**, the second type of actuator which will be used to provide a torque to orientate the satellite with respect to the Earth's Magnetic Field;
- **Gyroscope** and **Magnetometer**,whose,inside an **IMU** system, will provide measurements about the dynamic state of the satellite in space(Angular speed and Magnetic Field).

2.2 Distributed Lab Interfaces and Telemetry exchange

The system that was implemented for this thesis is a **Distributed Lab(DLab)** and consists of 3 main parts: an ADCS board, an **On-Board Computer**(**OBC**)board which embeds the **Electrical Power System**(**EPS**) and the **Ground Station** (**GS**),which is a PC.
In figure 2.2 a general scheme of the DLab is shown.

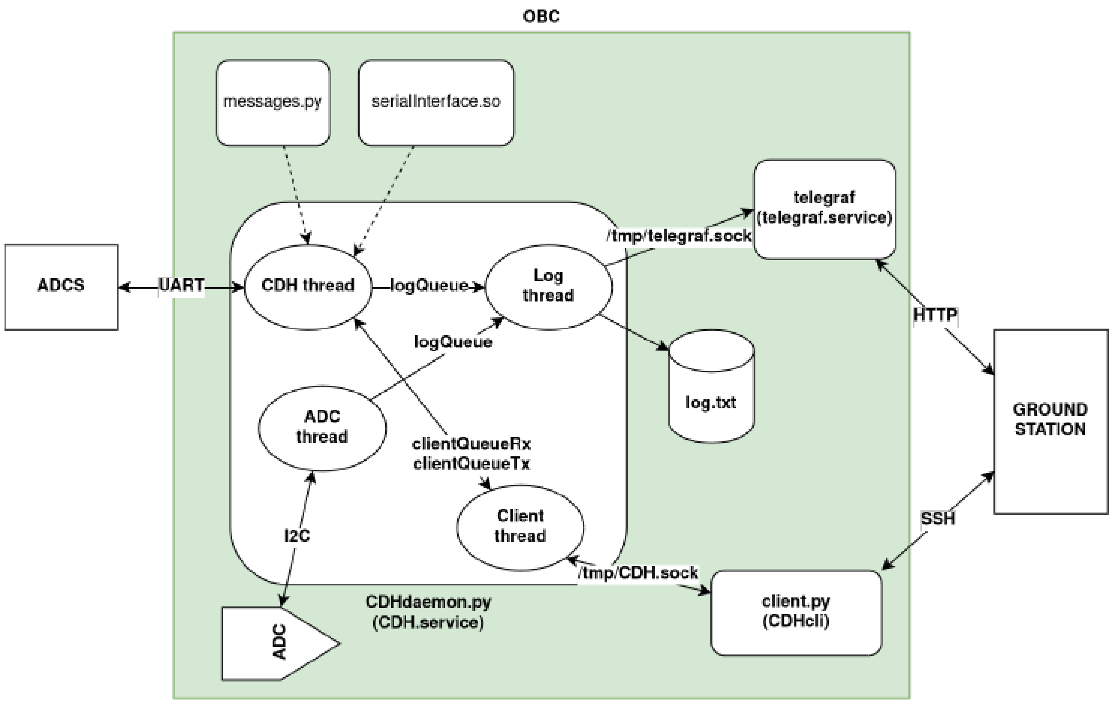


Figure 2.2: OBC threads scheme and Interfaces between systems

2.2.1 OBC

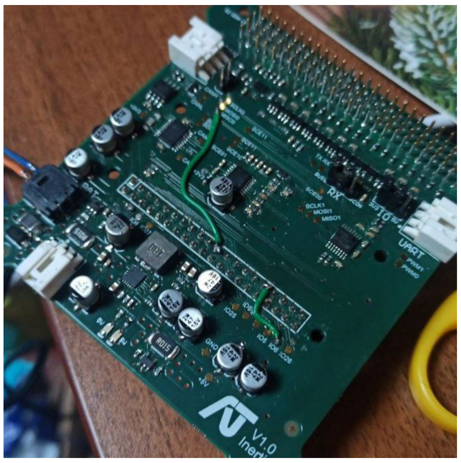


Figure 2.3: One of the available OBC PCBs

In figure 2.2 is shown the core software of the OBC: it automatically starts after every boot of the board.
It's a Python program subdivided into four threads:

- **CDH thread**: this is the "main" thread, whose purpose is, as the name suggests, getting commands from the user and handling data from ADCS, implementing the communication logic with the latter.
- **Client thread**: this thread is responsible for communication with the user, which is performed through another program (Client) that uses a Unix socket as a channel, similarly to how Telegraf does.
- **Log thread**: is the one responsible for logging data, both to Telegraf and optionally to a local log file.
- **ADC thread**: manages and samples the OBC's ADC.

The ADCS and OBC boards exchange data through the UART protocol, in particular :

- The OBC sends to ADCS OpMode(Operative Modes) packets and Target(desired angular velocity) packets,together with info about internal power state(coming from its ADC)
- The ADCS sens to OBC packets about IMU readings, Temperature sensors, and Actuators Currents.

2.2.2 ADCS Board

The ADCS board is a board to determine and control the attitude of a micro-satellite(CubeSat). It takes operative commands by the OBC board(UART) to provide current to magnetorquers to reach a determined orientation in the space. To do so a simple **PD control law** is implemented: this law takes the desired angular velocity value sent by the OBC and converts the request into a Duty Cycle to provide, through **PWM signals**, to the H-bridge ICs that drive the Magnetorquers in the three Axis. There are 2 main Operative Modes(**OpModes**):

- **OpMode 0**: it is the IDLE and SAFETY state of the ADCS. In this state the ADCS stops PWM command to Magnetorquers whenever a problem occurs during the tests(Overheating of the ICs or Overcurrents in the Power lines);

- **OpMode 1**: it is the state where control is implemented. For this experiment a **Dethumbling maneuver** is applied: given that the CubeSat spins with angular velocities different from 0 the Dethumbling allows to stop the Cubesat, bringing the angular velocity to 0. In alternative, it can be used to rotate the CubeSat with constant angular velocity from a stationary condition.

We'll see later that another OpMode was introduced for the Final Tests at the Facility.

2.2.3 Ground Station(GS)

To manage the communication between the OBC and the Ground station a data logging infrastructure has been created: the GS hosts the database(**InfluxDB**) and the User Interface (**Grafana**). On the OBC a **Raspberry Pi** is present, it serves as the Central unit of the OBC: it produces metrics and sends them to GS via an InfluxDB agent (**Telegraf**).
The picture below shows the implementation infrastructure for the Ground Station:

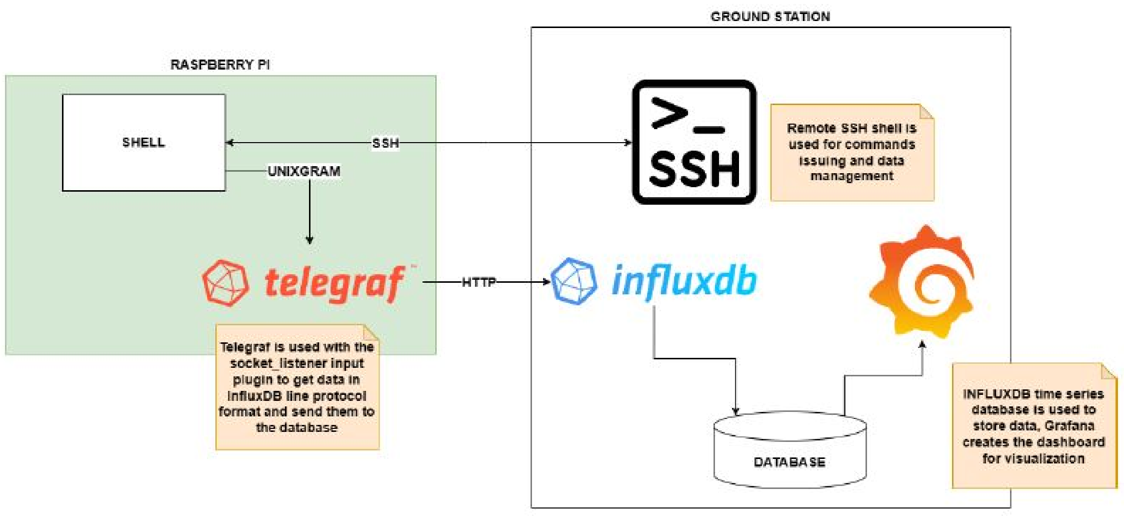

GROUND STATION
RASPBERRY PI
SHELL
SSH
UNIXGRAM
SSH
Remote SSH shell is used for commands issuing and data management
telegraf
HTTP
influxdb
Telegraf is used with the socket_listener input plugin to get data in InfluxDB line protocol format and send them to the database
DATABASE
INFLUXDB time series database is used to store data, Grafana creates the dashboard for visualization

Figure 2.4: Ground station infrastucture

InfluxDB is a time series database, which is suited to store data samples spanning over time.

Telegraf is the standard agent for InfluxDB, its purpose is to serve as an entry point for data to be sent to the database, potentially without directly writing queries or code. To do so it provides a huge variety of input/output plugins that can automatically gather metrics from the computer or systems or interface with all kinds of Web services. In this thesis, it is used simply as a bridge to send data to InfluxDB using its line protocol format (strings containing the measurements) using the socket_listener input plugin.

Grafana is a Dashboard making package that is typically used together with InfluxDB, its purpose is to create nice-looking graphs, charts, and all sorts of visualization for data stored in influxDB.
Below is shown how the Dashboard presents itself on Grafana:

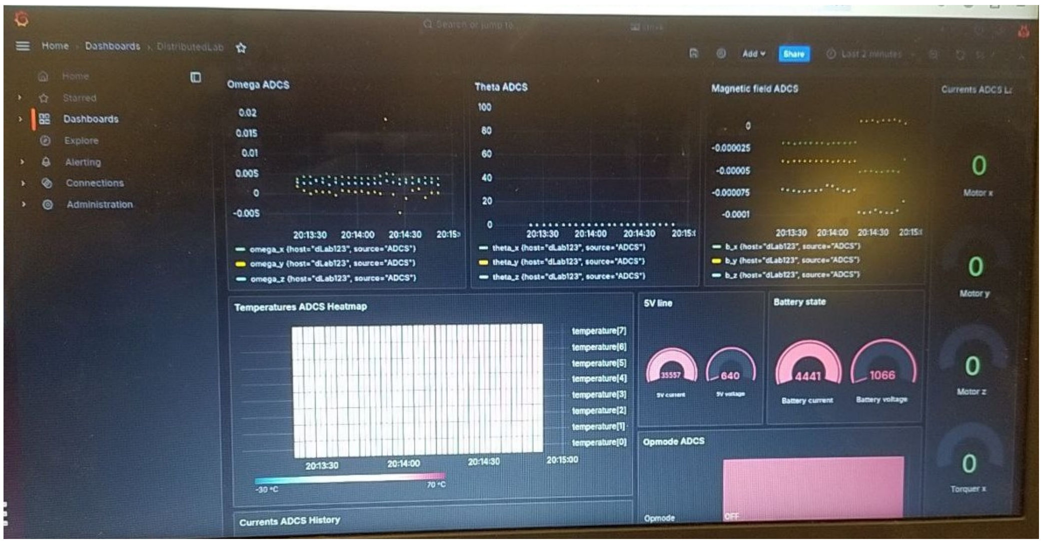


Figure 2.5: Dashboard visualization on Grafana

2.3 Control - Dethumbling

Normally, attitude control has two stages which have the **De-Tumbling** phase before the attitude pointing phase. The **Tumbling** of the satellites is induced by the rotation of launchers. The **De-Tumbling** is necessary to reduce the rotation rate of the satellites released from the launchers: it is the first task of the ADCS before any attitude control after entering the orbit[6]. This thesis aims to develop a PCB board to control actively the attitude of a 0.5U Cubesat.In particular, only the Dethumbling phase will be considered and an Attitude Control Algorithm will be implemented. It means that actuators will use only three **Magnetorquers** to control the orientation of the CubeSat.
This control algorithm will be then tested in the structure present at the Facility of the University of Bologna in Forlì, to be validated.

2.3.1 Control Model

During the design phase at the system level, it was decided to use a simple **PD controller** to implement the Dethumbling: since we implement a Dethumbling maneuver we need to stop from a condition where the satellite is spinning, so starting from the Desiderd Attitude composed of the only desire angular speed, coming from the Ground Station directive, the actuation of the Magnetorquers is done and the feedback comes from the readings of the IMU device. The error between the desired target(target angular speed) and the one read by the IMU will produce an error that will be given to a PD controller which will define which is the mechanical torque to develop at the actuator level to adjust the trajectory, in particular, to reduce the angular speed of the system.
It means the controller is a **speed controller**. Now it is reported a simple block scheme of the control:

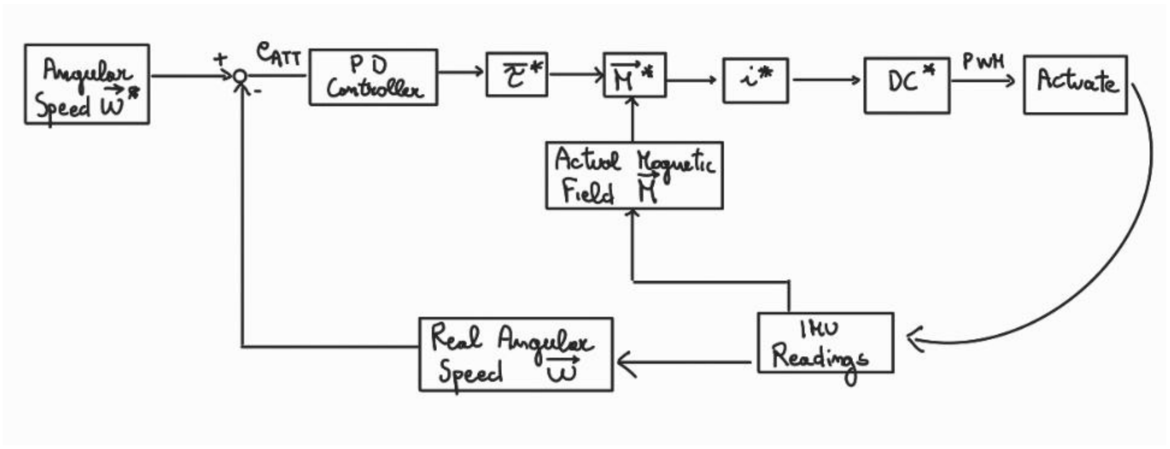

Angular Speed $\vec{\omega}^*$
+
-
e_{ATT}
PD Controller
$\vec{\tau}^*$
$\vec{M}^*$
i^*
DC^*
PWM
Activate
Actual Magnetic Field $\vec{M}$
IMU Readings
Real Angular Speed $\vec{\omega}$

Figure 2.6: PD control block scheme

Every "desired" variable will be indicated with an asterisk.
Starting from the desired angular speed: the GS will send to the OBC an Opmode command equal to 1 to say that the control mode will be started. Once it is received it will send to the OBC the desired **angular speed**:

$$\vec{e}_{ATT} = \vec{\omega}^* - \vec{\omega} \tag{2.1}$$

As written in equation 2.1 the desired angular speed is compared to the one coming from the IMU reading and an error is produced.
Then the derivative of the error is calculated as in equation 2.2:

$$\frac{d\vec{e}_{ATT}}{dt} = \frac{\vec{\omega}^* - \vec{\omega}}{dt} \tag{2.2}$$

The desired torque will be produced as a combination of the P and D control parameters and the values previously calculated in 2.1 and 2.2 as in equation 2.3:

$$\vec{\tau}^* = \overline{P} \cdot \vec{e}_{ATT} + \overline{D} \cdot \frac{d\vec{e}_{ATT}}{dt} \tag{2.3}$$

where the matrixes $\overline{P}$ and $\overline{D}$ are diagonal matrixes: $\overline{P} = \begin{bmatrix} P_x & 0 & 0 \\ 0 & P_y & 0 \\ 0 & 0 & P_z \end{bmatrix}$ and $\overline{D} = \begin{bmatrix} D_x & 0 & 0 \\ 0 & D_y & 0 \\ 0 & 0 & D_z \end{bmatrix}$.

From this the desired produced **Magnetic Dipole** ([7]) by the Magnetorquer is found as:

$$\vec{M}^* = \frac{\vec{\tau}^* \cdot \vec{B}}{||\vec{B}||^2} \tag{2.4}$$

Notice that to calculate this last vector the readings of the Magnetic Field from the IMU are used.

Then it is used to calculate the needed current to have that Magnetic Dipole:

$$\vec{I}^* = \frac{\vec{M}}{N \cdot A} \tag{2.5}$$

where N is the number of turns of the Magnetorquer and A is the Magnetorquer coil's area.
Finally, knowing the Magnetorquer equivalent Resistance and its Supply Voltage it is possible to calculate the desired Duty Cycle:

$$\vec{DC}^* = \frac{\vec{I} * R_{Magn}}{Vdd} \tag{2.6}$$

It is clear that this control law doesn't take into account all the nonidealities of the system, it is not complete: it doesn't use the angular acceleration measured by the IMU and there is no inner current loop control or external position control loop. However, this must be seen as a starting point for future works: the P and D parameters of the controller will be found by tentatives in the test and are not intended to be found theoretical if not as starting reference values to find the best ones.

Chapter 3

Hardware Description

In this chapter the design and components choices will be discussed:

3.1 Requirements for the ADCS Board

In this section requirements at system-level regarding the board are discussed:

- **Mechanical Dimensions**:In figure 3.1 the main mechanical sizes are indicated. The most important dimensions to consider are the board's dimension which must be 90x96 mm, the holes for the spacers and screws must be of type M3(3mm), the distance between the place for the PCI bus(whose dimensions are indicated in figure 3.2 and 3.3) and the spacers hole(on the left), as well as the distance from the board margin(as indicated in the figure), must be respected for all the stacked-up boards. These dimensions are essential to be respected in the PCB design phase otherwise the couldn't be a possibility for a correct assembly and integration of the boards.

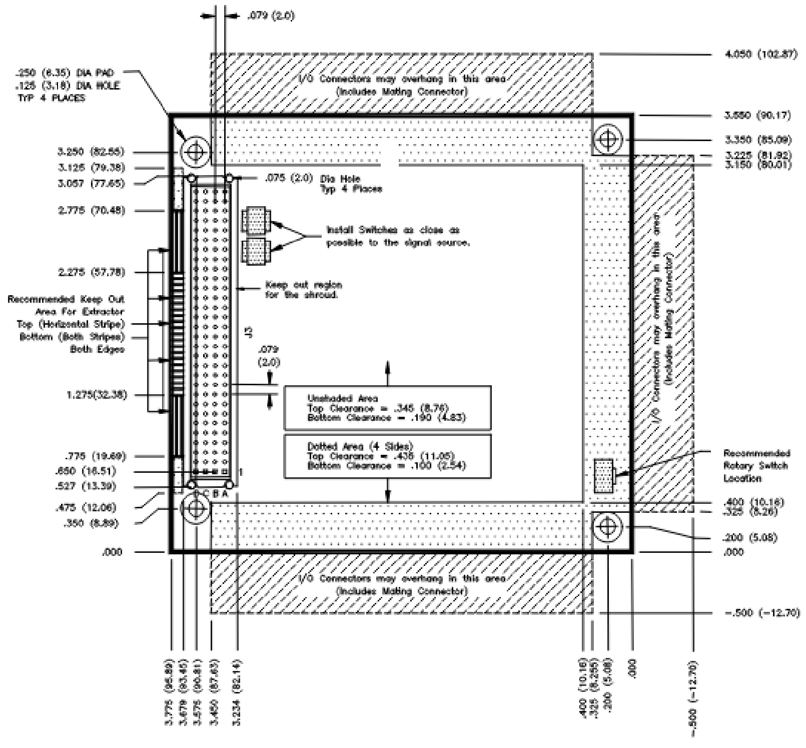


Figure 3.1: Board Mechanical Dimensions

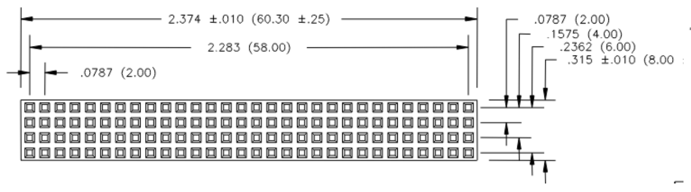


Figure 3.2: PCI Connector Mechanical dimensions A

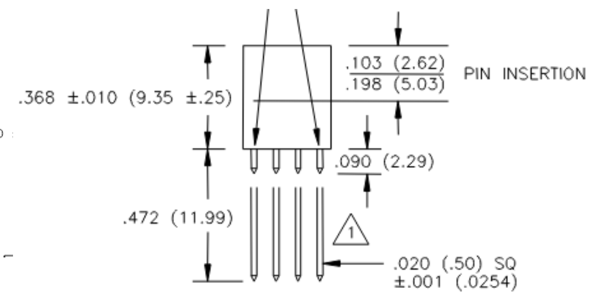
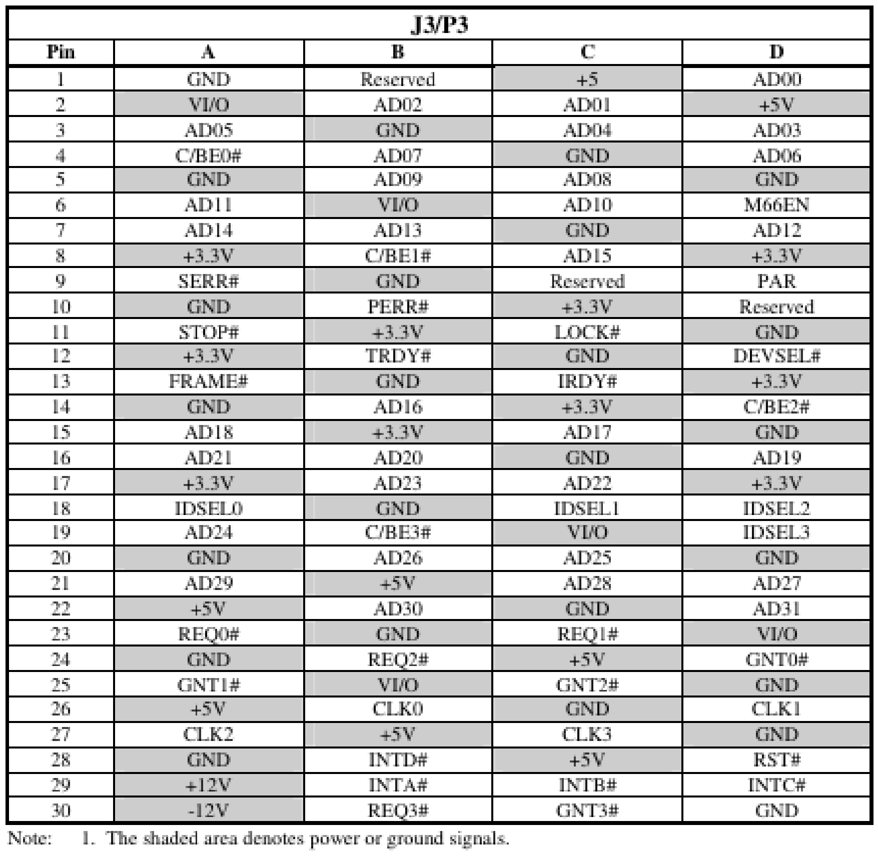


Figure 3.3: PCI Connector Mechanical dimensions B

- **Bus Signals Assignments**: the PCI Connector has a total of 120 pins organized in 4 columns of 30 pins every pin has its label, which means that the power signals(Vcc, GND) as well as communication signals, etc... must be put in the same pins for every board who's stuck-up.
  Below(Figure 3.4) a table of all the signals is shown.



J3/P3				
Pin	A	B	C	D
1	GND	Reserved	+5	AD00
2	VI/O	AD02	AD01	+5V
3	AD05	GND	AD04	AD03
4	C/BE0#	AD07	GND	AD06
5	GND	AD09	AD08	GND
6	AD11	VI/O	AD10	M66EN
7	AD14	AD13	GND	AD12
8	+3.3V	C/BE1#	AD15	+3.3V
9	SERR#	GND	Reserved	PAR
10	GND	PERR#	+3.3V	Reserved
11	STOP#	+3.3V	LOCK#	GND
12	+3.3V	TRDY#	GND	DEVSEL#
13	FRAME#	GND	IRDY#	+3.3V
14	GND	AD16	+3.3V	C/BE2#
15	AD18	+3.3V	AD17	GND
16	AD21	AD20	GND	AD19
17	+3.3V	AD23	AD22	+3.3V
18	IDSEL0	GND	IDSEL1	IDSEL2
19	AD24	C/BE3#	VI/O	IDSEL3
20	GND	AD26	AD25	GND
21	AD29	+5V	AD28	AD27
22	+5V	AD30	GND	AD31
23	REQ0#	GND	REQ1#	VI/O
24	GND	REQ2#	+5V	GNT0#
25	GNT1#	VI/O	GNT2#	GND
26	+5V	CLK0	GND	CLK1
27	CLK2	+5V	CLK3	GND
28	GND	INTD#	+5V	RST#
29	+12V	INTA#	INTB#	INTC#
30	-12V	REQ3#	GNT3#	GND

Note: 1. The shaded area denotes power or ground signals.

Figure 3.4: PCI Bus Signal Assignments

- **Power Supply**: To evaluate which voltage regimes to adopt for the ADCS board, it is necessary to consider what the **Electrical Power System**(**EPS**) provides in terms of voltage and current regimes. Once this is done, a decision can be made on whether to use converters to lower or raise the available voltage levels;

- **Communication Protocol Interfaces**: In a CubeSat, it is crucial for subsystems to exchange data to monitor each one's status, read sensor measurements, decide on the type of maneuver to be performed, and generally manage the operational state of the CubeSat. Since our ADCS board is supported by an **On-Board Computer**(**OBC**), which communicates with a **Ground Station**(**GS**), it is necessary first to establish which communication protocol to use to allow communication between the two systems. Subsequently, during the selection of the integrated circuits to be mounted on the board, it is important to understand which of these have a digital communication interface and thus determine whether it is necessary to use additional communication lines in addition to the one already mentioned. In general, various communication protocols can be used in a CubeSat, but the ones used essentially are two:

  - 1) **SPI** ;
  - 2) **UART**: .

- **Microcontroller**: To evaluate which microcontroller to choose some aspects must be taken into account:
  - Power Requirements in terms of voltages and currents;
  - Communication Interfaces(how many lines available and variety of communication protocols adopted);
  - Type of ICs, sensors, and in general circuitry present on the board(Analog or Digital);
  - Performance of the CPU.
- **Drivers and Magnetorquers**: Magnetorquers must generate enough magnetic moment to interact with the Earth's Magnetic Field and produce the desired torque for Attitude Control. Moreover, they must be realized with coils good enough to support the maximum amount of delivered current without experiencing Overheating.
  Whereas electronic drivers must be able to support a wide range of supply voltage and current, together with a good range of driving control frequency for the H_bridge transistors;

3.2 Microcontroller

After considering all of these aspects the STM32L452RE Nucleo Board was chosen([8]). Some details about the board:

- It mounts and Arm 32-bit Cortex-M4 processor with fpu which can reach frequencies up to 80Mhz;
- The power supply range is between 1,71 and 3.6 V;
- There are up to 83 fast I/Os, most 5 V-tolerant;
- 12 timers are available: a 16-bit advanced motor-control timer, a 32-bit and three 16-bit general purpose timers, two 16- bit basic timers, two low-power 16-bit timers (available in Stop mode),2 watchdogs and a SysTick timer;
- There are plenty of analog peripherals:
  - One 12-bit ADC 5 Msps, up to 16-bit with hardware oversampling, 200 μA/Msps;
  - One 12-bit DAC output channel, low-power sample and hold;
  - One operational amplifier with built-in PGA;
  - Two ultra-low-power comparators;

  - Accurate 2.5 V or 2.048 V reference voltage buffered output.
- 17 communication interfaces are available:
  - A USB 2.0 full-speed crystal less solution with LPM and BCD
  - One SAI (serial audio interface)
  - Four I2C FM+(1 Mbit/s), SMBus/PMBus
  - Three USARTs (ISO 7816, LIN, IrDA, modem)
  - One UART (LIN, IrDA, modem)
  - One LPUART (Stop 2 wake-up)
  - Three SPIs (and 1x Quad SPI)

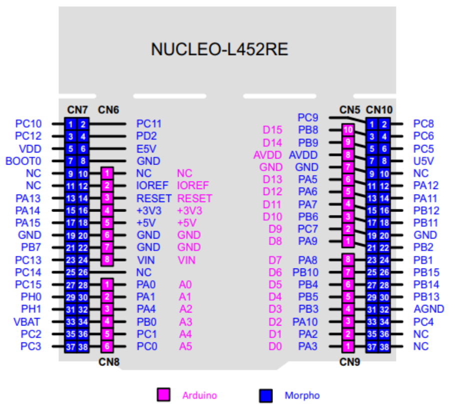


Figure 3.5: Nucleo Board L452RE Pinout

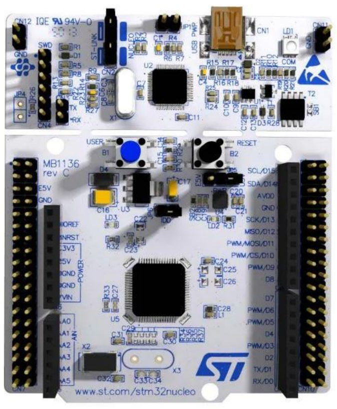


Figure 3.6: Nucleo Board L452RE

3.3 ADC and Multiplexer

To read the temperature coming from NTC sensors a chain of multiplexers plus an external ADC was chosen.

3.3.1 ADC

The AD7788 from Analog Devices([9]) was chosen, which is a low-power, low-noise front end for low-frequency measurement applications. It contains a low-noise 16-bit sigma-delta adc. It operates from an internal clock. Therefore, the user does not have to supply a clock source to the devices: its output rate is 16.6 HZ, which gives simultaneous 50Hz/60Hz rejection.
Its functional block diagram is:

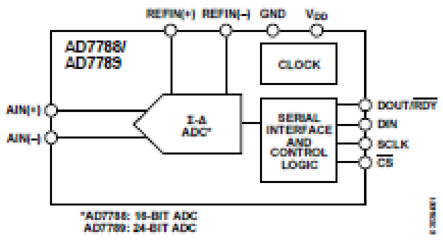


Figure 3.7: ADC Functional Block Diagram

The device operates with a single power supply from 2.5 V to 5.25 V. This external ADC is chosen against the internal one provided by the Nucleo board because it has a higher resolution(Nucleo's ADC has a 12-bit resolution) and as will be shown later the ports available on the Nucleo board are not sufficient to access its internal ADC channels.

Pin Configuration and Function Description

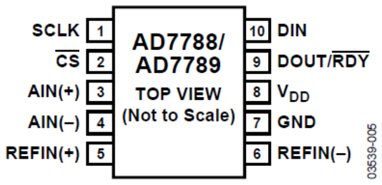


Figure 3.8: Pin configuration

There is a total number of 10 pins on the ICs :

- **PIN 1: SCLK(Serial Clock Input)** = for Data Transfers to and from the ADC. The serial clock can be continuous, with all data transmitted in a continuous train of pulses. Alternatively, it can be a noncontinuous clock with the information being trans- mitted to or from the ADC in smaller batches of data.
- **PIN 2: CS(Chip Select Input)** = This is an active low logic input used to select the ADC. CS can be used to select the ADC in systems with more than one device on the serial bus or as a frame synchronization signal in communicating with the device. CS can be hardwired low, allowing the ADC to operate in 3-wire mode with SCLK, DIN, and DOUT/RDY that is used to interface with the device.
- **PIN 3: AIN(+)(Positive Analog Input)** = it is the positive terminal of the fully differential analog input.
- **PIN 4: AIN(-)(Negative Analog Input)** it is the negative terminal of the fully differential analog input.
- **PIN 5: REFIN(+)(Positive Reference Input)** = it ranges between VDD and GND + 0.1 V. The nominal reference voltage (REFIN(+) - REFIN(-)) is 2.5 V, but the device functions with a reference from 0.1 V to V_{DD}.
- **PIN 6: REFIN(-)(Negative Reference Input)** = it ranges between GND and V_{DD} - 0.1 V.
- **PIN 7: GND** = Ground Reference Point.
- **PIN 8: V_{DD}** = Supply Voltage between 3 V or 5 V nominal.
- **PIN 9: DOUT/$\overline{RDY}$** = The DOUT/$\overline{RDY}$ falling edge can be used as an interrupt to a processor, indicating that valid data is available. With an external serial clock, the data can be read using the DOUT/$\overline{RDY}$ pin. With CS low, the data/control word information is placed on the DOUT/$\overline{RDY}$ pin on the SCLK falling edge and is valid on the SCLK rising edge. The end of a conversion is also indicated by the $\overline{RDY}$ bit in the status register. When CS is high, the DOUT/$\overline{RDY}$ pin is three-stated, but the $\overline{RDY}$ bit remains active.
- **PIN 10: DIN(Serial Data Input)** = to the Input Shift Register on the ADC. Data in this shift register is transferred to the control registers within the ADC; the register selection bits of the communications register identify the appropriate register.

Internal(On-Chip) Registers

The ADC is controlled and configured via a number of on-chip registers which are: **Communication Register**(**CR**), **Mode Register**(**MR**), **Status Register**(**SR**) and **Data Register**(**DR**).

- **Communication Register**
  The Communications Register is an 8-bit, WRITE-ONLY register. All communications to the device must start with a write operation to the communications register. The data written to the communications register determines whether the next operation is a READ or WRITE operation, and to which register this operation takes place.
- **Mode Register**
  The mode register is an 8-bit register from which data can be read from or written to. This register is used to configure the ADC for range, to set unipolar or bipolar mode, or to place the device into power-down mode.
- **Status Register**
  The status register is an 8-bit, read-only register. To access the ADC status register, the user must write to the communications register, select the next operation to be read, and load Bit RS1 and Bit RS0 with 0.
- **Data Register**
  The conversion result from the ADC is stored in this data register. This is a read-only register. On completion of a read operation from this register, the $\overline{RDY}$ bit/pin is set.

For further details about the register's contents the datasheet() is available.

Digital Communication Interface

In order to operate this ADC a digital interface is available, in fact, it is possible to communicate with this ADC considering different protocols:

- **SPI**
- **QSPI**
- **MICROWIRE**

In our case, we chose to use the SPI protocol since it is available on the Nucleo Board. As described in the datasheet there are mainly three conversion modes:

- Continuous conversion Mode
- Single conversion Mode
- Continuous Reading Mode.

Continuous conversion Mode

This is the default power-up mode. The ADC continuously converts, and the $\overline{RDY}$ pin in the status register goes low each time a conversion is complete. If CS is low, the DOUT/$\overline{RDY}$ line also goes low when a conversion is complete. To read a conversion, the user can write to the communications register, indicating that the next operation is a read of the data register. The digital conversion is placed on the DOUT/$\overline{RDY}$ pin as soon as SCLK pulses are applied to the ADC. DOUT/ $\overline{RDY}$ returns high when the conversion is read. The user must ensure that the data register is not being accessed at the completion of the next conversion or else the new conversion word is lost.

Single conversion Mode

In single-conversion mode, the ADC is placed in power-down mode between conversions. When a single conversion is initiated by setting MD1 to 1 and MD0 to 0 in the mode register, the ADC powers up, performs a single conversion, and then returns to power-down mode. The devices require 1 ms to power up and settle. It then performs a conversion, requiring a time period of 2t$_{ADC}$. DOUT/$\overline{RDY}$ goes low to indicate the completion of a conversion. When the data word has been read from the data register, DOUT/$\overline{RDY}$ goes high. If CS is low, DOUT/$\overline{RDY}$ remains high until another conversion is initiated and completed.

Continuous Reading Mode

Rather than write to the communications register each time a conversion is completed to access the data, the ADC can be placed in continuous read mode. By writing 001111XX to the communications register, the user needs only to apply the appropriate number of SCLK cycles to the ADC, and the data word is automatically placed on the DOUT/$\overline{RDY}$ line when a conversion is complete. When DOUT/$\overline{RDY}$ goes low to indicate the end of a conversion, sufficient SCLK cycles must be applied to the ADC, and the data conversion is placed on the DOUT/$\overline{RDY}$ line. When the conversion is read, DOUT/$\overline{RDY}$ returns high until the next conversion is available. In this mode, the data can be read only once. Also, the user must ensure that the data word is read before the next conversion is complete.

3.3.2 Multiplexer

The Multiplexer is an SN74HC4851PWR from Texas Instruments [10] which is an 8-channel analog multiplexer/demultiplexer with injection-current effect control, which allows signals at disabled analog input channels to exceed the supply voltage without affecting the signal of the enabled analog channels and this eliminates the need for external diode/resistor networks typically used to keep the analog channel signals within the supply-voltage range.

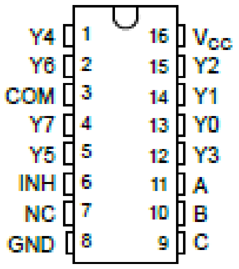


Figure 3.9: SN74HC4851PWR Pinout

3.3.3 Temperature Sensor

[11],The choosen temperature sensor is a NTCLE203E3103FB0 from Vishay [12]. This thermistor has a nominal resistance at 25 °C of 10 kΩ(R$_{25}$) with a tolerance of ±1% and a B = 3977 K with a tolerance of ±0.75%, which is the material constant expressed in Kelvin. This parameter controls the R$_T$ slope, and allows computing the thermistor characteristic curve, as explained in Vishay application note [13].
Figure 3.10 shows a typical curve behavior of the Resistance of an NTC sensor as a function of the temperature(K).

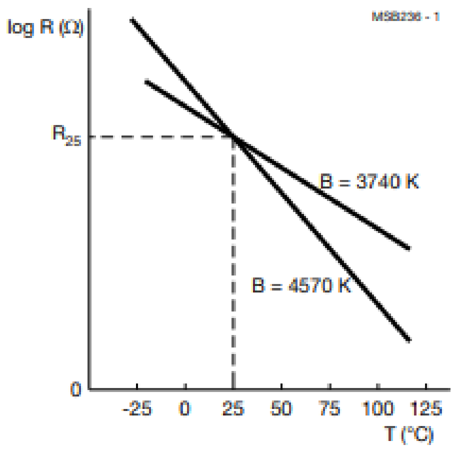


Figure 3.10: Typical resistance as a function of temperature for an NTC temperature sensor

The equation to retrieve the B value is:

$$B_{T1/T2} = \ln\left(\frac{R_1}{R_2}\right) \cdot \frac{1}{\frac{1}{T_1} - \frac{1}{T_2}} \tag{3.1}$$

Where T$_x$ and R$_x$ represents a chosen absolute temperature in Kelvin and the corresponding thermistor resistance value at that temperature in Ω.
From this equation and by knowing the thermistor resistance value at 25 °C from the datasheet, we can compute the thermistor resistance variation with temperature using the following equation:

$$R_1 = R_2 * e^{(\frac{1}{T_1} - \frac{1}{T_2}) * B} \tag{3.2}$$

3.3.4 Measurement chain

[11], The Measurement chain is made up of 8 NTCs which together with a resistor each form a voltage division that goes as input to a Multiplexer which selects one reading among all the voltages and puts it in input to the ADC. For this reason, the ADC is exploited in a Single-ended configuration.
A capacitor is put in parallel to the thermistor to help filter out high-frequency noise. In the figure (3.11) the conditioning circuitry for the Thermistors is shown, inserted in the overall Measurement Chain. This is clearly a simple schematic, where only two sensors are connected, the real circuit will be shown later. Now let's proceed to calculate the equivalent Voltage that will be put in input the ADC to be converted for every thermistor and then retrieve the inverse formula for the Temperature to be used even in the Firmware code.
The equivalent voltage, as said before, is a voltage division between the thermistor and the resistor:

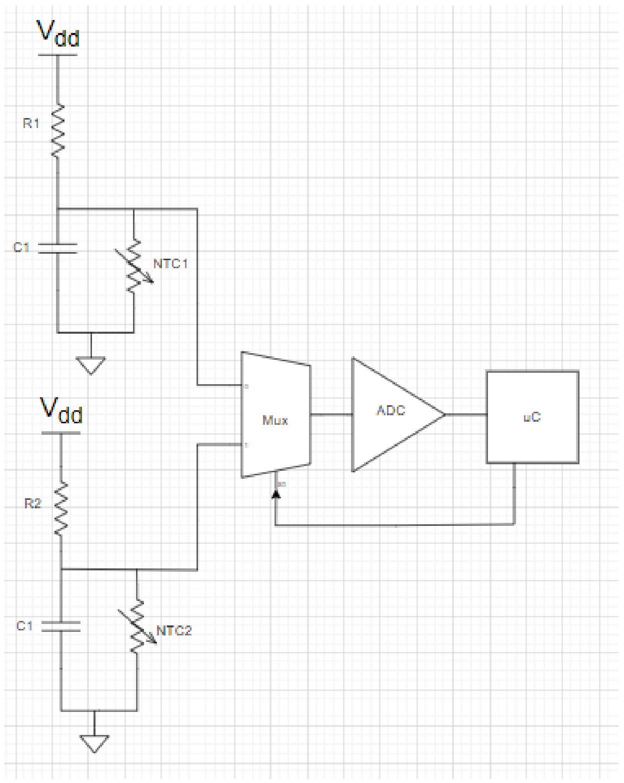


Figure 3.11: Measurement Chain for the Thermistor

$$V_{eq} = V_{dd} * \frac{R_{NTC}}{R_{NTC} + R_x} = V_{dd} * \frac{1}{1 + \frac{R_x}{R_{25}} * e^{(\frac{1}{298.15} - \frac{1}{T}) * B}} \tag{3.3}$$

The equivalent resistance seen by the input of the Mux is:

$$R_{eq} = R_{NTC} // R_x = \frac{R_{NTC} * R_x}{R_{NTC} + R_x} = \frac{R_x}{1 + \frac{R_x}{R_{25}} * e^{(\frac{1}{298.15} - \frac{1}{T}) * B}} \tag{3.4}$$

Inverting the 3.3 we obtain as expression for the Temperature:

$$T = \frac{298.15K * B}{B - 298.15K * \ln[(\frac{V_{dd}}{V_{eq}} - 1) * \frac{R_{25}}{R_x}]} \tag{3.5}$$

3.4 IMU

The Inertial Measurement Unit(IMU) is a crucial element for an ADCS board since it provides measurements to understand the state of the system and make decisions on the movement to perform in terms of rotation and/or "propulsion". For this project the IMU used is the MTi-3 from Xsens [14] (figure 3.12). This device is essentially a SOC(System-On-Chip), it is a self-contained Attitude and Heading Reference System (AHRS) as a 12.1 x 12.1 mm module, so it is easily integrable in a board due to the small dimensions. It integrates a magnetometer, a gyroscope, and an accelerometer whose technical specifications are:

- **Gyroscope**: standard full range of measurement of 2000 deg/s;
- **Magnetometer**: standard full range of measurement of +/- 8 G(Gauss);
- **Accellerometer**: standard full range of measurement of 16 g;

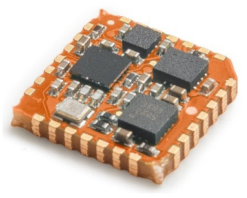


Figure 3.12: The component

It allows the exchange of data using three communication protocols(SPI, UART, I2C) and it has its own Software Suite (MT Manager, Firmware updater, Magnetic Field Mapper) where it is possible to perform self-calibration for all three integrated types of sensors.

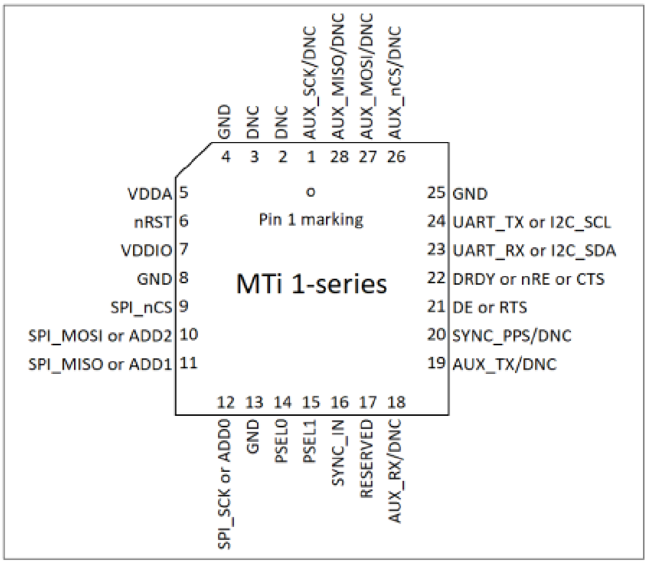
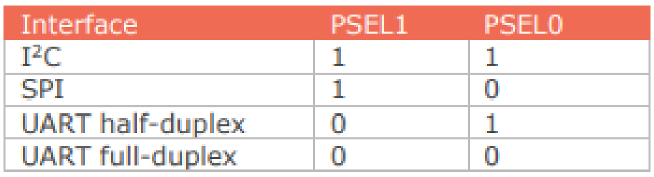


Figure 3.13: IMU Pin Configuration

The user can select which communication protocol interface to employ for the IMU through the peripheral selection pins PSEL0 and PSEL1. The hardware reads the state of these pins at start-up and configures its peripheral interface according to the figure 3.11. To change the selected interfaces, the user must first set the desired state of the PSEL0 and PSEL1 pins, and then reset the device. The IMU has internal pull-ups on the PSEL0 and PSEL1 pins so if these pins are left unconnected, the peripheral interface selection defaults to I2C (PSEL0 = 1.PSEL1 = 1).



Interface	PSEL1	PSEL0
I²C	1	1
SPI	1	0
UART half-duplex	0	1
UART full-duplex	0	0

Figure 3.14: IMU Peripheral selection

The MTi uses a right-handed coordinate system as the basis of the sensor frame(figure 3.15). The accelerometer determines the origin of measurements. Figure 3.16 shows the location of the accelerometer of the MTi 1-series.

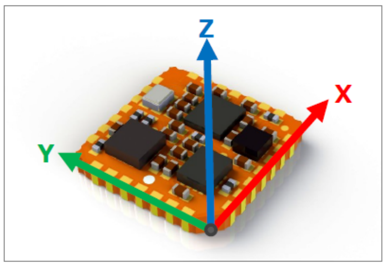


Figure 3.15: IMU Reference Plane

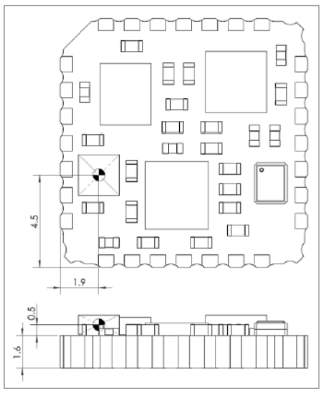


Figure 3.16: Origin of IMU reference plane

3.5 Magnetorquers

The objective of the actuation is to control the orientation of satellite faces in the 3 axis. So three Magnetorquers have been designed.

3.5.1 Theoretical introduction

The type of Magnetorquer([15]) we are using, referred to as **air core magnetorquer**(figure 3.17), is basically an electromagnetic actuator that consists of a multi-turned coil which uses an external applied magnetic field to develop a magnetic torque when current flows through its wire. This lately will allow to orient the face of the Cubesat in which the magnetorquer is allocated.

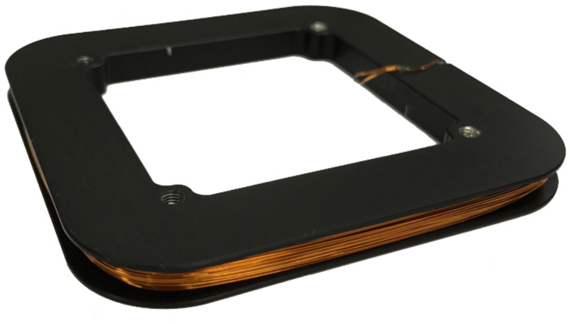


Figure 3.17: An air core magnetorquer with squared shape

The magnetic moment of a coil with N turns is given by the equation 2.6 below, where $\vec{n}$ is a unit vector perpendicular to the coil's plane, N is the number of turns, A is the coil's area, and I is the current flowing in the coil.

$$\vec{m} = N\vec{A}I = NAI\vec{n} \tag{3.6}$$

The dipole moment created by the magnetorquer $\vec{m}$ interacts with the magnetic field $\vec{B}$ and the developed magnetic torque is represented by the formula 3.7, this means that the magnetic torque is perpendicular to both the magnetic field and the magnetic dipole moment.

$$\vec{\tau} = \vec{m} \times \vec{B} \tag{3.7}$$

Considering the square coil as an ideal electrical conductor with constant section and uniform distribution mass, the resistance of the wire R may be calculated using the Second Ohm's Law, considering that the length L of the wire will be the number of turns times the perimeter of the square(in figure 3.18 there's a descriptive image([16]) of the dimensions of the magnetorquer, considering it a square the two sides of the square are equal and so w = h = a):

$$R = \rho\frac{L}{S} = \rho\frac{4aN}{S} \tag{3.8}$$

From which the turns variable N is explicit:

$$N = \frac{RS}{4a\rho} \tag{3.9}$$

Where ρ is the electrical resistivity of the coil, a is the side length of the squared magnetorquer and S is the section of the wire which is calculated as $S = \pi * r^2$ where r,(which is equal to 0,315 mm), is the radius of the section of the file which can be approximated as a circle.

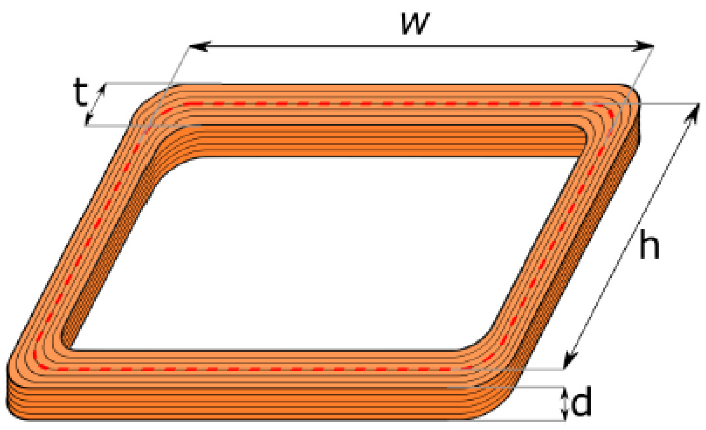


Figure 3.18: Air-cored magnetorquer's dimensions

3.5.2 Design choices

To design the magnetorquers there were to consider some things:

- Available copper coil: there where limited availability for the coil;
- Maximum deliverable current: as will be explained later the maximum current that could flow through the actuator is 2 A;
- Minimum equivalent resistance: for the same reason for which we want the current to be under 2 A we want the resistance to be at least 6 Ω in order to have a maximum current surely under the current max threshold. Moreover, higher resistance means even less possibility to overheat for the magnetorquers at the equal value of current;

It was chosen to optimize the number of turns of the magnetorquers N according to the maximum value of generated torque and magnetic field(considered at the equator), so the generated dipole moment was supposed constant and the best combination of a number of turns and current was chosen.
In particular, for the max generated Mechanical Torque it was chosen a value of 0.1 $\cdot 10^{-3} N \cdot m$, and for the magnetic field, a value of 30 μT, calculated at the equator, resulting in a max Magnetic Dipole for the Magnetorquers of 33.3333 $A \cdot m^2$.
Moreover, it was considered for the side a length of 8.5 cm, so the area A of the actuator is 0.07225 m^2, fixed.

m$[A \cdot m^2]$	33.3333	33.3333	33.3333
N[-]	462	308	231
A$[m^2]$	0,007225	0,007225	0,007225
I$[A]$	1	1.5	2

Table 3.1: Values for magnetorquer dipole moment

The thread-off between a good number of turns and a maximum current value, not higher than 2 A, leads to a value of Turns of about 300, with a peak current value of 1.5 A. So this resulted to be the chosen parameters configuration for all the magnetorquers.

After finding the optimal number of turns the 3 magnteorquers were realized. In Figure 3.19 the realized magnetorquers assembled to the CubeSat structure are shown. Moreover, the frame of reference of the IMU is indicated in the picture

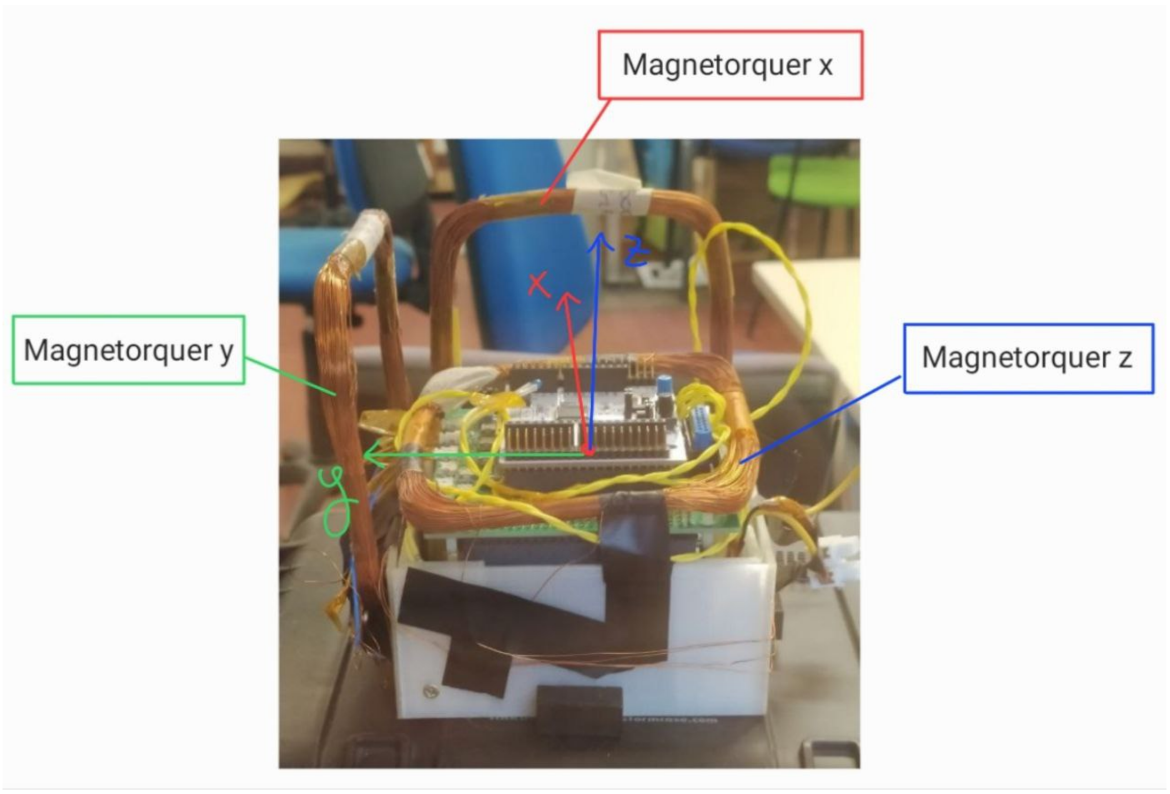


Figure 3.19: Magnetorquers for every axis + IMU frame Reference

Their realization encountered a problem that where told before: the copper coil wasn't enough to obtain three magnetorquers of 300 turns each, so after realizing the magnetorquers we obtained two of them of 300 Turns each and one of 210 Turns.
Considering the last Magnetorquer, if we calculate the possible generated magnetic dipole looking at the current we obtain the values shown in the table 3.2

m$[A * m^2]$	33.3795	22.7586
A$[m^2]$	0,07225	0,07225
I$[A]$	2.2	1.5

Table 3.2: Values for the magnetorquer of 210 Turns

This means that we could have 2 possibilities: to have the same magnetic dipole the current should increase over the limit we imposed of 2A, in the other hand to have the same maximum current of the other magnetorquers we should account for a reduced magnetic dipole, which in turns leads to a lower developed magnetic torque.

Since we can't afford a too high driving current we have to accept the third magnetorquer to develop less torque. As we'll see later in the final tests this will create some problems.

3.5.3 Electrical characterization

Equivalent resistance

Now let's discuss the equivalent resistance of the built magnetorquers. To estimate this value we refer to the formula 3.8, since we know that the coil is made of copper its resistivity constant ρ is equal to $1.68 * 10^-8 \Omega m$. Putting in a table the obtained values we have:

	Magnetorquer X	**Magnetorquer Y**	**Magnetorquer Z**
N	300	300	210
ρ **[Ω· m]**	1.68×10^{-8}	1.68×10^{-8}	1.68×10^{-8}
a [m]	0.085	0.085	0.085
r [m^2]	3.15	3.15	3.15
S [m^2]	3.115×10^{-7}	3.115×10^{-7}	3.115×10^{-7}
Ideal R [Ω]	5.5	5.5	3.85
Real R [Ω]	30.7	30.7	23

Table 3.3: Resistance values for the magnetorquers

As it can be seen the ideal value of resistance resulting from the formula doesn't correspond to the real one measured using a multimeter. This is because of many reasons: non-optimal construction of magnetorquers since they were made by hand, the wires were attached using a special glue for aerospace application only with hands. Moreover, the total length of the wire is an approximated value, it is approximately near the one used for the calculations. Considering that, the constraint on the equivalent real resistance in all three cases is respected: it exceeds the 6 Ω which is important if we want to keep the maximum current under the 2 A.
However, the optimum design of magnetorquer resistance is out of the scope of this thesis work

Resonant frequency

In order to drive correctly a magnetorqer some things must be taken into account. The amount of current injected into a magnetorquer must be enough to not let it heat due to the joule effect, still allowing it to provide the maximum magnetic dipole possible useful for the control. Moreover, since a Magnetorquer can be characterized as an RL circuit, the resonant frequency must be evaluated. As explained in this article [16], providing a PWM square wave with a frequency close to the resonant frequency minimizes the ripple which is the AC side of the Square wave signal provided by the driver to the Magnetorquer. So, after the design and realization of the magnetorquers, their resonant frequency has been evaluated using a signal generator and an oscilloscope: depending on the driving circuit, the driving voltage of a Magnetorquer can have a ripple with associated amplitude and frequency. Its response to the ripple voltage depends on its impedance for that specific frequency, in fact, a Magnetorquer can be modeled as an RL circuit without the back-emf voltage typical of mechanical actuators(DC motor, Stepper motor, etc..). In figure 3.20 is provides a typical inductor Impedance Magnitude and Phase plot considering a span of frequencies and it is valid even for Magnetorquers.

Depending on the driving frequency the Magnetorquer can express a **dominant capacitive impedance** behavior for low frequencies, where its equivalent impedance has a prevalent negative imaginary part or it can manifest a **dominant inductive impedance** behavior for high

frequencies, it means it's equivalent impedance has a prevalent positive imaginary part.
The peak on the graph instead is an intermediate and "equilibrium" situation where no dominance is present from one side or the other but the imaginary part of the impedance is a combination of the two natures.

The frequency at which this happens is called a **Self-Resonant Frequency**. At the Resonant Frequency, the ripple current is lowest in magnitude. Both, ripple voltage and ripple current, must be avoided due to related EMI. While the effects of the ripple voltage can be significantly reduced using conductive shields, the ripple current is more problematic: Magnetorquers generate a relatively strong variable magnetic field, proportional with the ripple current, that spreads over a large volume and this cannot be accepted if the task of a Magnetorquer is to generate a Magnetic Dipole to provide in turns a good Mechanical Torque interacting with a magnetic field.
Additionally, the power associated with the ripple components is dissipated, with no contribution to the dipole moment, resulting in a reduction of the power efficiency.

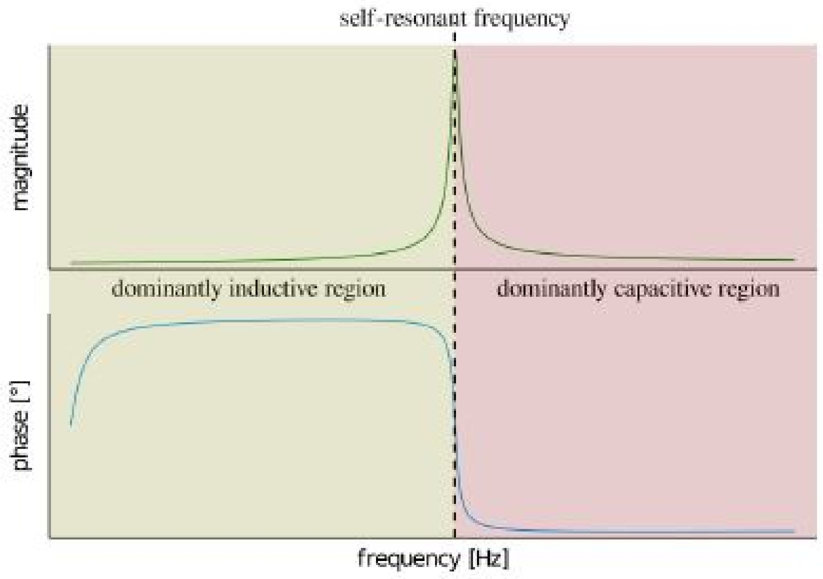


Figure 3.20: Typical impedance profile of an inductor

For this reason the Self-Resonant Frequency of the three the Magnetorquer was evaluated. To do so we have to consider the concept behind the frequency response of the Magnetorquer: since at the Resonant Frequency there's a peak in the Impedance response it means that the voltage peak of the magnetorquer will be the highest at that frequency. Exploiting this concept we performed this experiment in the laboratory : We connected a Magnetorquer to a signal generator and we generated a 50%, 12 V square wave starting from a low frequency. In the meantime, the two poles of the magnetorquer were connected to an oscilloscope to plot the wave and print its peak-to-peak amplitude voltage. We increased gradually the frequency of the input signal from a few Hz and we saw that the amplitude of the signal on the magnetorquers started to rise, till a specific value. Passed that value the amplitude started to decrease continuing to increase the frequency.
That frequency value is the Resonant Frequency and figure 3.21 shows a simplified scheme of the setup in the laboratory.

We used a 50% square because at equal frequency the ripple is higher at that Duty Cycle so minimizing the ripple at 50% will automatically minimize it for all the other Duty Cycles.

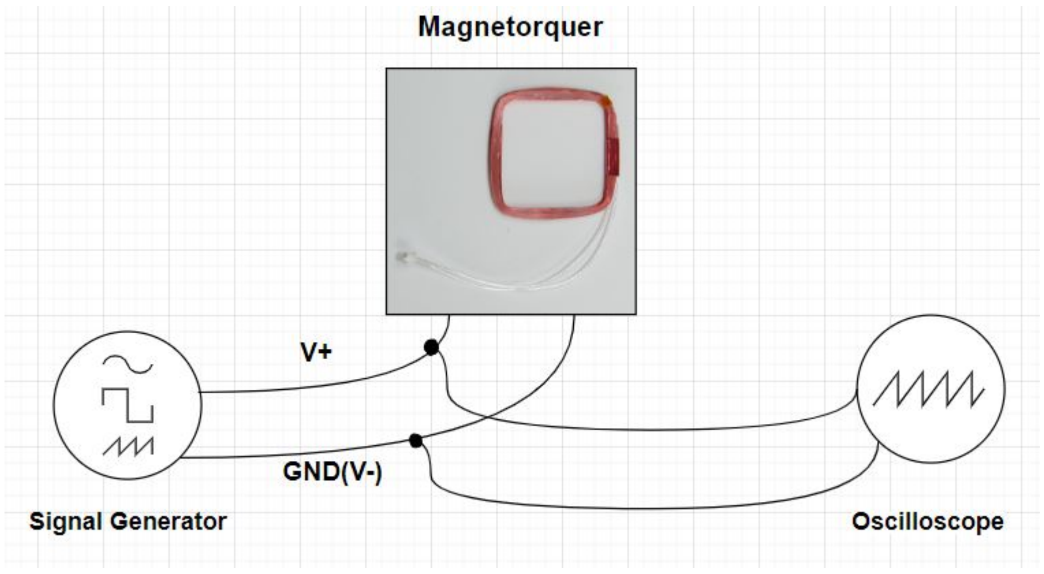


Figure 3.21: Test in laboratory to find the resonant frequency

The table below shows the found Resonant Frequencies and we can notice that since two over three magnetorquers have the same number of turns and characteristics it was expected that they had approximately the same resonant frequency.

	Magnetorquer X	**Magnetorquer Y**	**Magnetorquer Z**
N	300	300	210
Resonant Frequency $[Hz]$	70	70	123.5

Table 3.4: Found Resonant frequencies values for the three Magnetorquers

Obtaining these values we achieved an important result: as it will be explained in the next section the drivers of the magnetorquers can be subjected to a driving PWM frequency not higher than **200 KHz**. Finding these values we are quite below this threshold so we can exploit the ripple minimization without problems.

3.6 Actuators Driver

To select the motor driver IC some points were taken into account:

- The motor driver should have required a few external components in order not to use too much space on the surface of the board since 5 of them are needed;
- It should drive the motors and magnetorquer, so should have been electrically compatible with them, and in the meantime, the current flowing through actuators should have been measured.

Due to these motivations, the choice is made on the DRV8251A from Texas Instruments([17]), which is an integrated motor driver with N-channels H-bridge, charge pump, current sense feedback, current regulation, and protection circuitry.

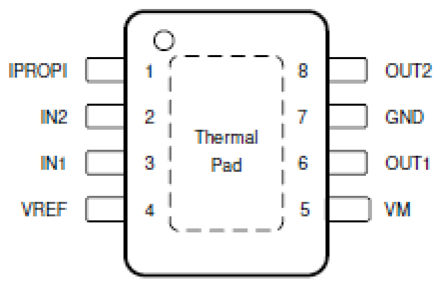


Figure 3.22: DRV8251A Pinout

3.6.1 Pin Description

In figure 3.22 the driver pinout is shown. The pins are respectively:

- PIN 1: IPROPRI = Analog current output proportional to load current;
- PIN 2 e 3: IN2 e IN1 = Logic inputs, they control the H-bridge output and have internal pulldowns;
- PIN 4: VREF = Analog input, can be applied a voltage between 0 and 5 V;
- PIN 5: VM = The 4.5 V to 48 V power supply. Connect a 0.1μF bypass capacitor to ground, as well as sufficient bulk capacitance, rated for the VM voltage.
- PIN 6 e 8 = OUT1 e OUT2 = H-bridge outputs, connect them directly to the motor or other inductive load.
- PIN 7 : GND = Device power ground;
- PIN 9 : PAD = Thermal pad.

3.6.2 Internal Circuitry Description

An internal current mirror architecture on the IPROPRI pin implements current sensing and regulation. This eliminates the need for a large power shunt resistor, saving board area and reducing system cost.
The IPROPRI current-sense output allows a microcontroller to detect motor stalls or changes in load conditions. The external voltage reference pin, VREF, determines the threshold of current regulation during start-up and stall events without interaction from a microcontroller

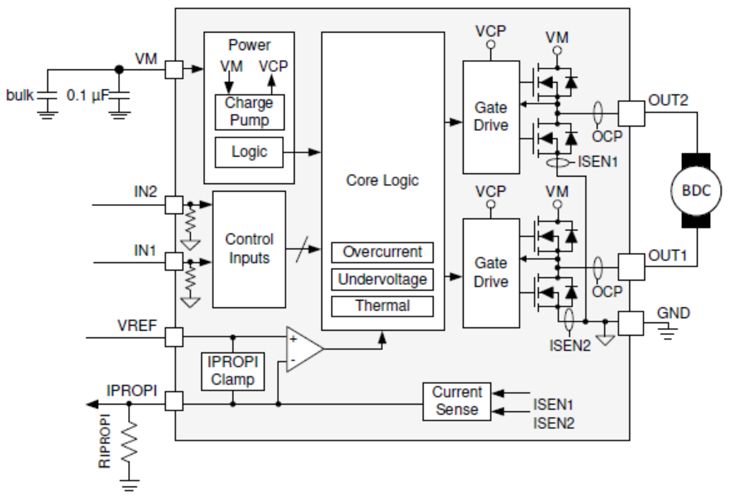
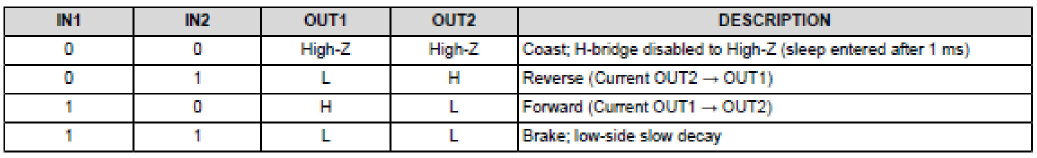


Figure 3.23: Functional Block Diagram

3.6.3 Bridge control

The DRV8251A output consists of four N-channel MOSFETs that are designed to drive high current. These outputs are controlled by the two logic inputs IN1 and IN2.



IN1	IN2	OUT1	OUT2	DESCRIPTION
0	0	High-Z	High-Z	Coast; H-bridge disabled to High-Z (sleep entered after 1 ms)
0	1	L	H	Reverse (Current OUT2 → OUT1)
1	0	H	L	Forward (Current OUT1 → OUT2)
1	1	L	L	Brake; low-side slow decay

Figure 3.24: H-Bridge Control Table

According to the table 3.24 in order to drive the component it must switch between the driving phase(Forward or Reverse) and the slow-decay phase(Brake). It means that for example if we want to drive with forward mode at 70% duty cycle IN1 must remain high all the time(100% duty cycle) and IN2 must be a PWM 50% square wave.
This is the way we will drive our actuators.

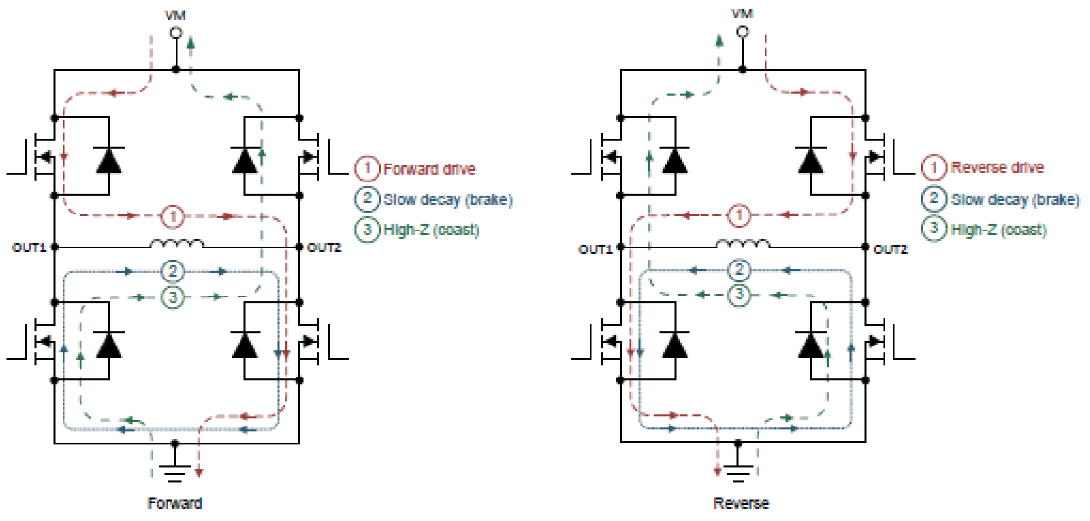


Figure 3.25: H-Bridge Current Paths

Figure 3.25 shows the current path in the three operating modes of the drive. Since we'll change between the Forward and Reverse modes we can notice that the current flows always from up to down, as we will describe soon this is crucial in the choice of the conditioning circuitry for the **Current Sensing**.

3.6.4 Current Sensing

The IPROPI pin outputs an analog current proportional to the current flowing through the low-side power MOSFETs in the H-bridge scaled by $\mathbf{A_{IPROPI}}$:

$$\mathbf{I_{PROPI}} = \mathbf{I_M} * \mathbf{A_{IPROPI}} \tag{3.10}$$

Where $\mathbf{I_{PROPI}}$ is measured in μA, $\mathbf{I_M}$ is the current flowing in the load and as described in the datasheet $\mathbf{A_{IPROPI}}$ has a value of 1575 μA/A.
The motor current is measured by an internal current mirror architecture on the low-side FETs which removes the need for an external power sense resistor as shown in 3.26. The current mirror architecture allows for the motor winding current to be sensed in both the drive and brake low-side slow-decay periods allowing for continuous current monitoring in typical bidirectional brushed DC motor applications

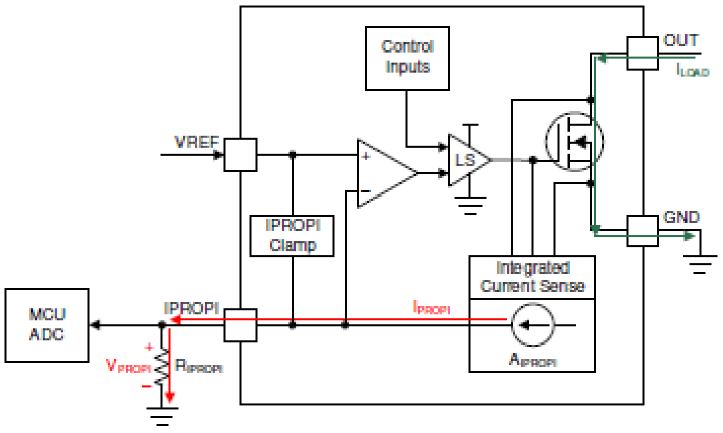


Figure 3.26: Integrated Current Sensing

The I_{PROPI} pin should be connected to an external resistor (R_{IPROPI}) to ground in order to generate a proportional voltage (V_{IPROPI}) on the I_{PROPI} pin with the I_{PROPI} analog current output. This allows for the load current to be measured as the voltage drop across the R_{IPROPI} resistor with a standard analog to digital converter (ADC). The R_{IPROPI} resistor can be sized based on the expected load current in the application so that the full range of the controller ADC is utilized.
Additionally, the DRV8251A device implements an internal I_{PROPI} voltage clamp circuit to limit V_{IPROPI} with respect to V_{VREF} on the V_{VREF} pin and protect the external ADC in case of output overcurrent or unexpected high current events.
The corresponding IPROPI voltage to the output current can be calculated as:

$$\mathbf{V_{IPROPI}} = \mathbf{I_{PROPRI}} * \mathbf{R_{IPROPI}} \tag{3.11}$$

3.6.5 Current Regulation

The DRV8251A device integrates current regulation using a fixed off-time current chopping scheme. This allows the devices to limit the output current in case of motor stall, high torque, or other high current load events without involvement from the external controller.

The current chopping threshold (I_{TRIP}) is set through a combination of the V_{VREF} voltage (V_{REF}) and I_{PROPI} output resistor (R_{IPROPI}). This is done by comparing the voltage drop across the external R_{IPROPI} resistor to V_{REF} with an internal comparator.

$$\mathbf{I_{TRIP}} * \mathbf{A_{IPROPI}} = \frac{\mathbf{V_{VREF}}}{\mathbf{R_{IPROPI}}} \tag{3.12}$$

3.6.6 Current Sensing Conditioning Circuitry

The possibility of using a simple resistor to read indirectly the actuator current makes the design of the circuit extremely simple. However, the ranges in terms of currents and voltages involved must be respected. In particular, since the resistor will be connected to the ADC channel of the Nucleo board, the maximum voltage it can experience must be 3.3V.

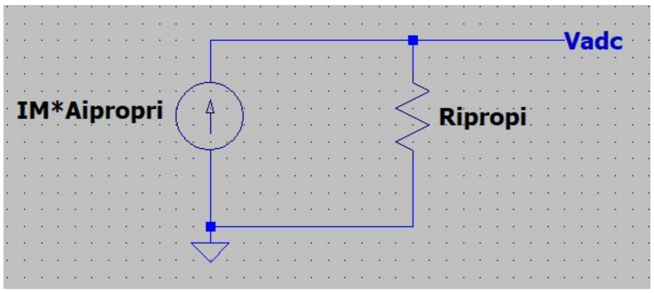


Figure 3.27: Schematic of the conditioning circuitry for the reading of actuator current

As shown in figure 3.27 the current flowing through the resistor can be seen as a current generator whose value is the actuator current($\mathbf{I_M}$) scaled of the current mirror ratio $\mathbf{A_{IPROPI}}$(=1575 μ/A)
.
From the datasheet [17], reading the table on recommended operating conditions, that the maximum deliverable output current from the driver is 4.1 A($\mathbf{I}_{out}$) when the motor voltage($\mathbf{V}_m$) is higher than 5.5v, and we have 12 v. Knowing that the current that flows through the resistor is te motor current scaled of a factor of $\mathbf{A_{IPROPI}}$we expect that the maximum current flowing on the resistor can be of about 6.4 mA. However, the datasheet suggests to arrive a maximum of 3 mA. This puts a lower boundary in the choice of the range for the resistor value. But let's proceed with order:
First, we must have clear the range of current our mechanical load can tolerate. This requirement was explained in the section 3.5, where a maximum value of 1.5 A was calculated, considering the number of turns and the characteristics of the wire. However, this value should be in the middle of the current range because that is the best value for the resulting mechanical torque. So we set a range, for every magnetorquer of about 1A÷2A, this is the maximum current range we want for our mechanical loads. We choose to have a range on the maximum output current to add flexibility to the available mechanical torque. In order to be able to obtain this range the simplest choice is to adoperate a potentiometer in series with a resistor: in this way when the potentiometer resistance is 0 the current will flow only through the fixed resistor. The value chosen for the resistors are 1 kΩ for the fixed resistor and the same 1 kΩ for the potentiometer, so that the resistor range is 1kΩ÷ 2kΩ. In the following, the calculation for this result is shown:
Since the chosen Magnetorquer current range is referred to as the maximum we must consider that it is a range on I_{TRIP} which is the upper limit applied by the driver hardware for the current output. So from 3.12 we explicit and calculate the I_{TRIP} range corresponding to the R_{IPROPI} range:

$$\mathbf{I_{TRIP}} = \frac{\mathbf{V_{VREF}}}{\mathbf{R_{IPROPI}} * \mathbf{A_{IPROPI}}} \tag{3.13}$$

- $\mathbf{I_{TRIPmax}} = 2.1$ A
- $\mathbf{I_{TRIPImin}} = 1.05$ A

So we almost obtained the same range we wanted. Clearly, the result is not the same because we choose commercial resistor values and not the exact values needed to obtain 1A÷2A.
So now we can proceed to calculate which is the range obtained in terms of current on the resistor $\mathbf{I_{IPROPI}}$. We apply the 3.10 and we obtain for the current :

- $\mathbf{I_{PROPImax}} = 3.3$ mA
- $\mathbf{I_{PROPImin}} = 1.64$ mA

And for the maximum voltage across the resistor, applying 3.11:

- $\mathbf{V_{IPROPI}} = 3.3$ v
- $\mathbf{V_{IPROPI}} = 3.28$ v

Putting the results in a table for the max and min cases we finally have:

	MIN	MAX
$\mathbf{I_{TRIP}}$	0 A	2.1 A
$\mathbf{V_{IPROPI}}$	0 V	3.3 V
$\mathbf{I_{PROPI}}$	0 mA	3.3 mA

Table 3.5: Case of R$_{IPROPI}$ = 1kΩ

	MIN	MAX
$\mathbf{I_{TRIP}}$	0 A	1.05 A
$\mathbf{V_{IPROPI}}$	0 V	3.28 V
$\mathbf{I_{PROPI}}$	0 mA	1.64 mA

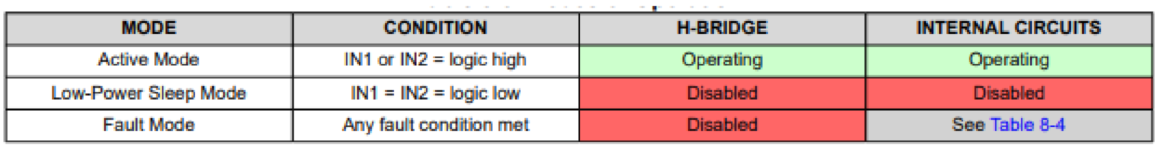
Table 3.6: Case of R$_{IPROPI}$ = 2kΩ

As we can notice the maximum value for the $\mathbf{I_{PROPI}}$ in the case of a resistance of 1kΩ exceeds the suggested value by the datasheet of 3 mA. However these are all ranges so the equivalent total resistance sensed by the ADC channel will be fixed in a specific value and in case of need it will be changed manually, so it is not probable to arrive at a maximum current of 2.1 A for the experiment we perform.
To conclude the 3.2 will be inverted, expliciting the current, to retrieve it in the Firmware code, since V_{IPROPI} is the voltage converted by the ADC.

3.6.7 Device Functional Modes

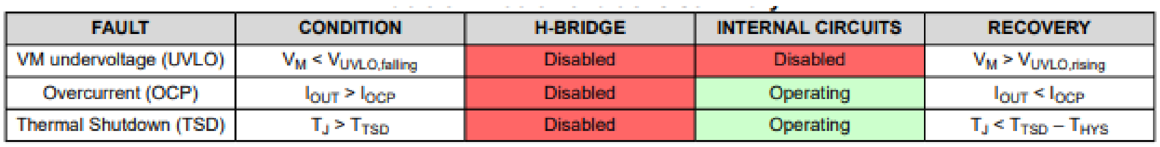
There are mainly three functional modes:

- **Active Mode**: After the supply voltage on the V$_M$ pin has crossed the undervoltage threshold V$_{UVLO}$, the IN$_x$ pins are in a state other than IN$_1$ and IN$_2$ = 0, and t$_{WAKE}$ has elapsed, the device enters active mode. In this mode, the H-bridge, charge pump, and internal logic are active and the device is ready to receive inputs;
- **Low-Power Sleep Mode**: When the IN$_1$ and IN$_2$ pins are both low for time t$_{SLEEP}$, the device enters a Low-Power sleep mode. In sleep mode, the outputs remain High-Z and the device draws minimal current from the supply pin (I$_{VMQ}$). If the device is powered up while all inputs are low, it immediately enters sleep mode. After any of the input pins are set high for longer than the duration of t$_{WAKE}$, the device becomes fully operational;
- **Fault Mode**: The device enters a fault mode when a fault is encountered. This is utilized to protect the device and the output load. The device behavior in the fault mode is described in figure 3.24 and depends on the fault condition. The device will leave the fault mode and re-enter the active mode when the recovery condition is met.



MODE	CONDITION	H-BRIDGE	INTERNAL CIRCUITS
Active Mode	IN1 or IN2 = logic high	Operating	Operating
Low-Power Sleep Mode	IN1 = IN2 = logic low	Disabled	Disabled
Fault Mode	Any fault condition met	Disabled	See Table 8-4

Figure 3.28: Summary Table of the modes of operation of the Driver



FAULT	CONDITION	H-BRIDGE	INTERNAL CIRCUITS	RECOVERY
VM undervoltage (UVLO)	$V_M < V_{UVLO,falling}$	Disabled	Disabled	$V_M > V_{UVLO,rising}$
Overcurrent (OCP)	$I_{OUT} > I_{OCP}$	Disabled	Operating	$I_{OUT} < I_{OCP}$
Thermal Shutdown (TSD)	$T_J > T_{TSD}$	Disabled	Operating	$T_J < T_{TSD} - T_{HYS}$

Figure 3.29: Fault Conditions Summary Table

3.7 PCB Design

In this section, the choices made in terms of PCB design will be explained.
The board was developed by using the open source software KIcad(version 7.0) which allows for creating the schematic, then realizing the relative PCB, and generating the Gerber files useful for the manufacturer to produce the board.

Figure 3.30 shows the overall schematic in Kicad but in this section, every component will be explored and explained in terms of design choices.

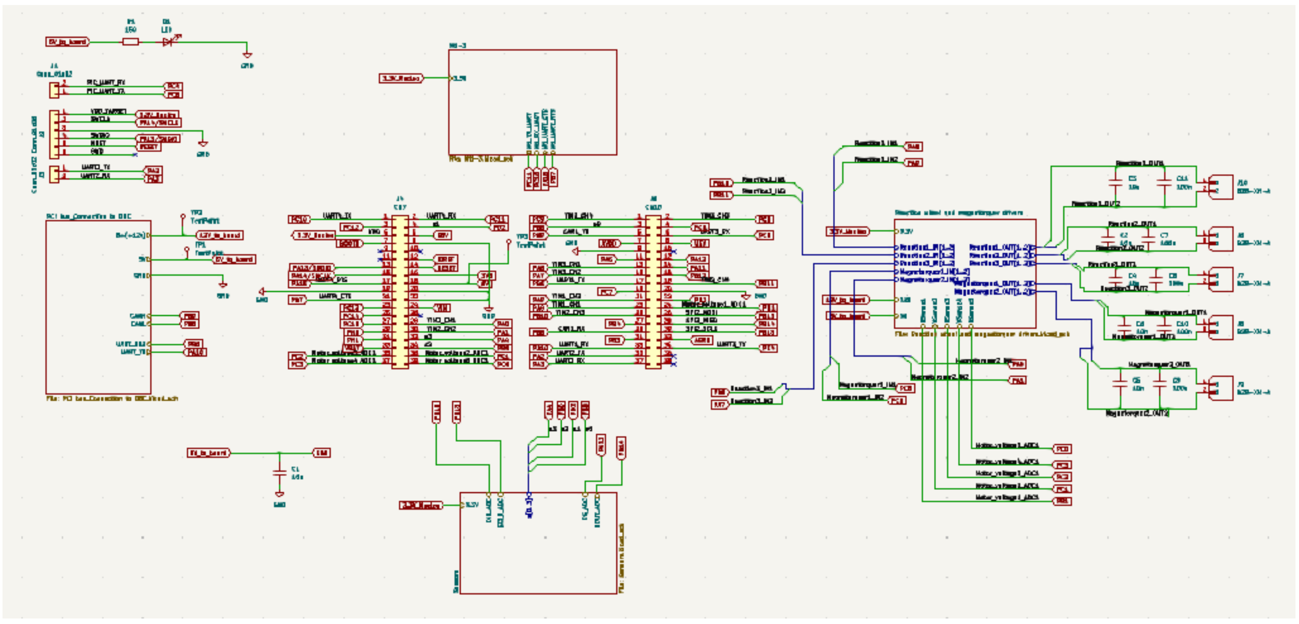


Figure 3.30: ADCS Board General Schematic

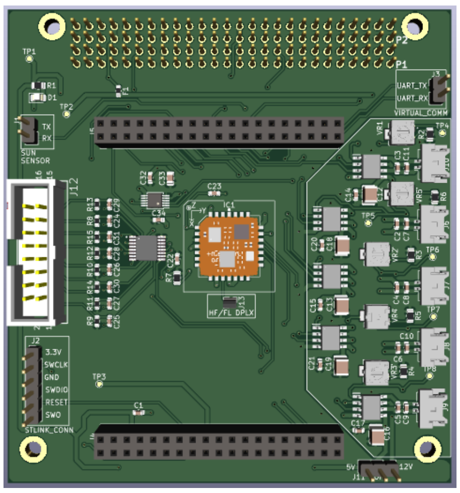


Figure 3.31: PCB front face on Kicad

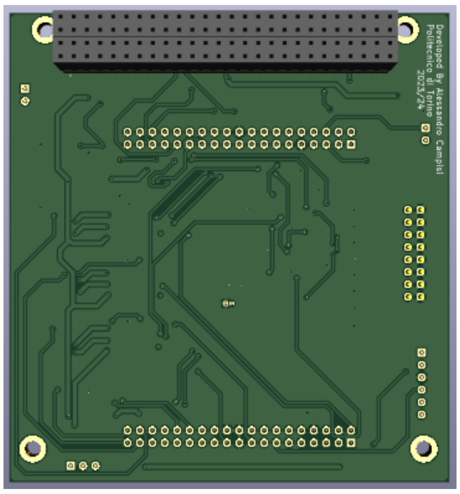


Figure 3.32: PCB back face on Kicad

In the picture below the main blocks discussed in 2.1 of the ADCS board are highlighted. The board is constituted of many components, however only the top layer is occupied, in fact, we can see that at the bottom part(figure 3.32) there's only the PCI bus connector.
In particular, the board is made up of(Figure 3.33):

- Blue part: Five Electronic Drivers, with for each driver a potentiometer in series with a resistor, a bulk capacitor, a connector for the actuator, and some decoupling capacitors;
- Pink part: One PCI bus connector;
- Yellow Part: Thermistors connector and its conditioning circuit with, and ADC and a mux, with other capacitors and resistors;
- Red part: Two connectors for the ST Morpho connectors.

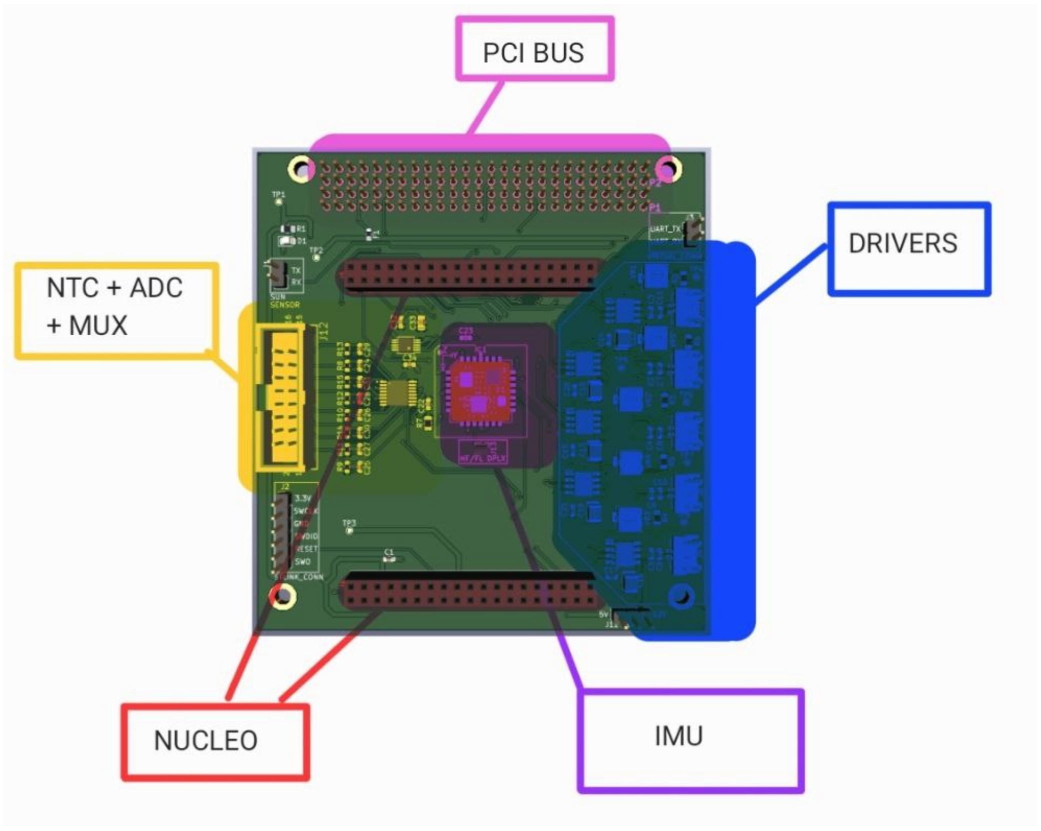


Figure 3.33: PCB Blocks

3.7.1 Connectors

The most important connectors of the board are the ones for the communication with the OBC(PCI Bus) and for the connection of the Nucleo Board.

PCI Bus and Nucleo Connectors

- **PCI Bus Connectors**: As described in the chapter 1 this bus is a standard for many applications and in particular is widely used for CubeSats. The OBC board works as EPS too so it provides all the power to the ADCS. There are 3 main lines: 3.3 v, 5 v, and 12 v. The 3.3v line is not used and motivations will be explained later, instead the 5v line is used to supply the Nucleo Board whereas the 12 v line(coming directly from the battery) is provided straight to the actuators(drivers). Since there are 3 different voltage regimes the choices made for their traces must take into account the amount of current they carry.

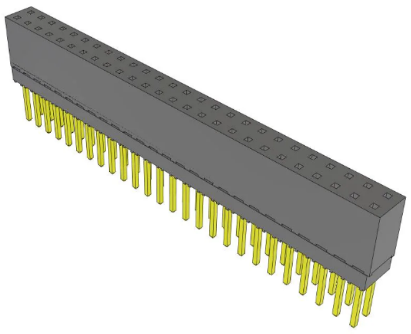


Figure 3.34: PCI BUS component

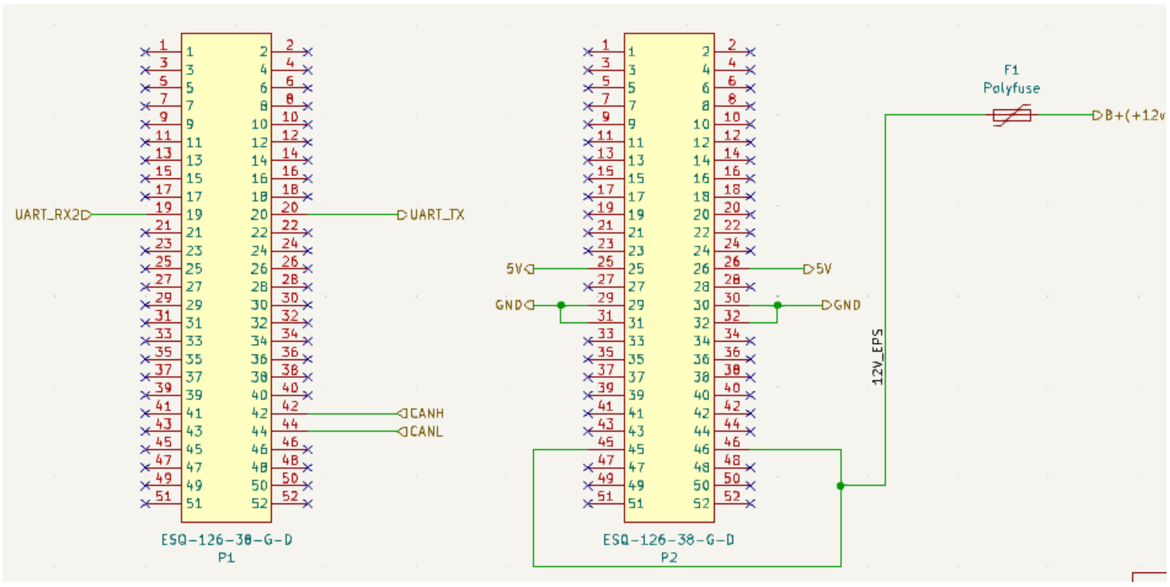


Figure 3.35: PCI BUS schematic

- **Nucleo Connectors**: For this PCB design, it was chosen for simplicity to connect the Nucleo Board provided by STMicroelectronics directly to the ADCS board. To do so it was needed to use 2 connectors to attach to the Nucleo **ST Morpho connectors** CN7 and CN10 (see figure 3.5). The connector used is the one in figure 3.37 and in the figure below instead are shown all the labels in every pin of the morpho connectors.

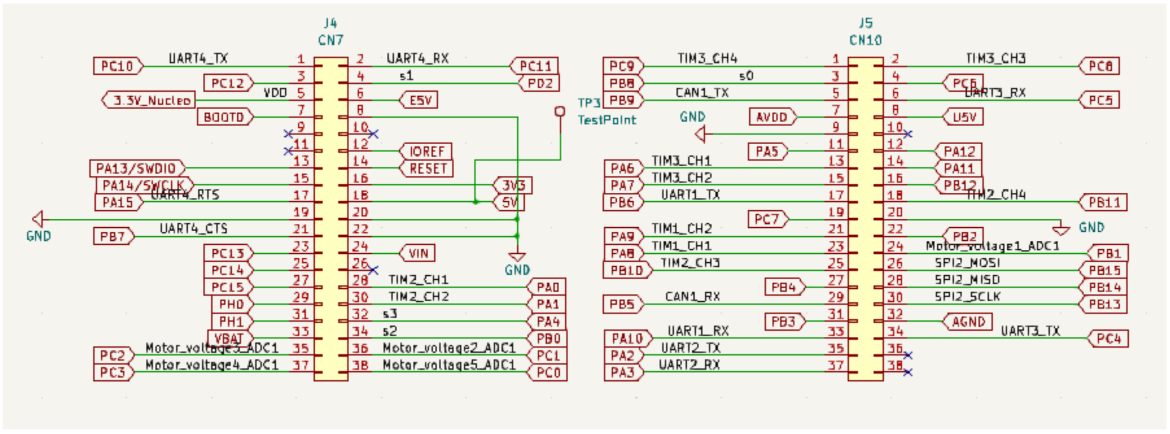


Figure 3.36: Nucleo Pin Sockets Schematic

Figure 3.37: Samtec 2x19 Pin Socket for the Nucleo

Thermistors and their connectors

Regarding the connector of NTC thermistors the SAMTEC one in figure 3.38 was used, it is the one soldered in the front-left side of the PCB. To connect it to all the 8 thermistors a female connector(figure 3.39), produced by Amphenol FCI was chosen. This connector can be opened from the bottom side in the image and it has a system that allows to embed wires inside it, after the removed part is attached again, The result can be seen in the figure next to it.

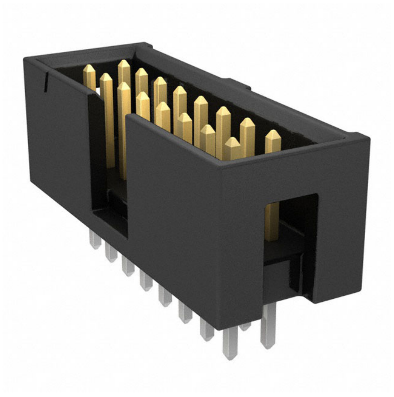


Figure 3.38: Male Connector for NTC thermistors

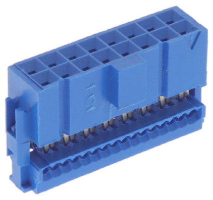


Figure 3.39: Female Connector for NTC thermistors

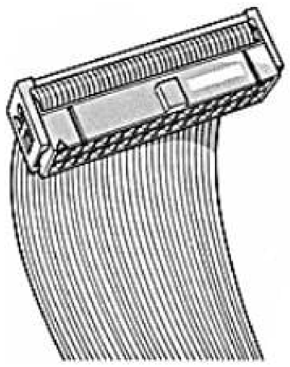


Figure 3.40: Female Connector for NTC thermistors + cable

The thermistors were soldered on Teflon wires that are strong enough to resist the enclosure of the connector on itself.

Other Connectors - Pin Headers

All the other connectors are, as indicated in figure 3.45, j1, j2, and j3 which are respectively the connectors for an external solar cell, the Nucleo debug interface, and Nucleo serial console interface(using one of the 4 UART available by the Nucleo):

- **Solar cell connector**: the idea is to use this connector to provide communication via UART to a microcontroller, for example, a PIC from a Microchip, that acquires the signal coming from a solar cell, appropriately conditioned, and sends them to the Nucleo Board to process them for the Attitude Control. However, this thesis work was not expected to use a Solar Cell;

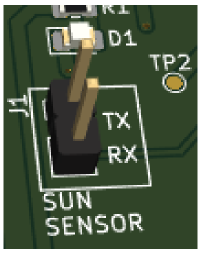


Figure 3.41: Sun Sensor Pin Header

- **Debug and Serial console interface**: This is an important part of the design of the PCB. As the Nucleo Board stands, it is cumbersome and occupies a big space considering a plane parallel to the PCB surface. Thanks to the Nucleo pin sockets it is lifted vertically and doesn't bother any component except for the male pin header of the NTC thermistors: in fact, the ST-Link part of the Nucleo is placed exactly in correspondence with that connector and this means that the Nucleo board cannot be plugged completely inside the pin sockets because it leans on the connector. So during the design phase, it was chosen to detach the St-Link from the Nucleo Board.
  This means that the connection to the USB port of the computer is disabled since the St-Link is the means to upload/debug the firmware using the CUBEIDE software and use the serial port to "visualize" printfs on the PC monitor. To overcome this problem some precautions were taken:

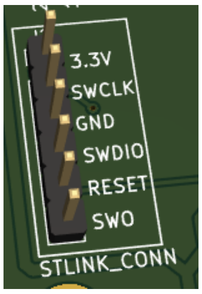


Figure 3.42: Pin Header DEBUGGER interface

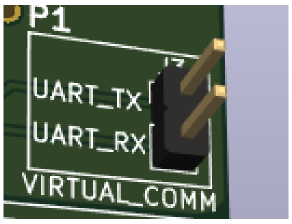


Figure 3.43: Pin Header Serial Console interface

To allow the connection of the detached ST-Link it was decided to add two pin headers, one for the debugging interface and one for the serial port. In figures 3.42 and 3.43 are

presented the two pin headers.
These pin headers are connected to the Nucleo connectors in the following way(the Debugger Pin Header and Serial Pin Header enumerations refer to the figure 3.45):

Debugger Pin Header	**Nucleo Connector Pin**
PIN 1 - 3.3V	CN7 - PIN 5
PIN 2 - SWCLK	CN7 - PIN 15
PIN 3 - GND	CN7 - PIN 8
PIN 4 - SWDIO	CN7 - PIN 7
PIN 5 - RESET	CN7 - PIN 14
PIN 6 - SWD	-

Table 3.7: Connection of Debugger pin header to the Nucleo Connectors

Serial Console Pin Header	**Nucleo Connector Pin**
PIN 1 - UART_TX	CN10 - PIN 35
PIN 2 - UART_RX	CN10 - PIN 37

Table 3.8: Connection of Serial Interface pin headers to the Nucleo Connectors

To connect the external ST-Link to the Nucleo board the configuration is the one described in the Table below and in the image 3.44 it is shown the connection between the ST-Link and The Nucleo connectors when they are separated.
The Nucleo connectors are internally connected to the Pin headers in the ADCS board as in table 3.7, which then are the ones actually to connect to the ST-Link using external cables:

ADCS - Debugger Pin Header	**ST-Link CN4 Pin Header**
PIN 1 - 3.3V	CN4 - PIN 1
PIN 2 - SWCLK	CN4 - PIN 2
PIN 3 - GND	CN4 - PIN 3
PIN 4 - SWDIO	CN4 - PIN 4
PIN 5 - RESET	CN4 - PIN 5
ADCS - Serial Console Pin Header	**ST-Link CN3 Pin Header**
PIN 1 - UART_TX	CN3 - PIN 1
PIN 2 - UART_RX	CN3 - PIN 2

Table 3.9: Connection of pins between ST-Link and Pin Headers on ADCS board

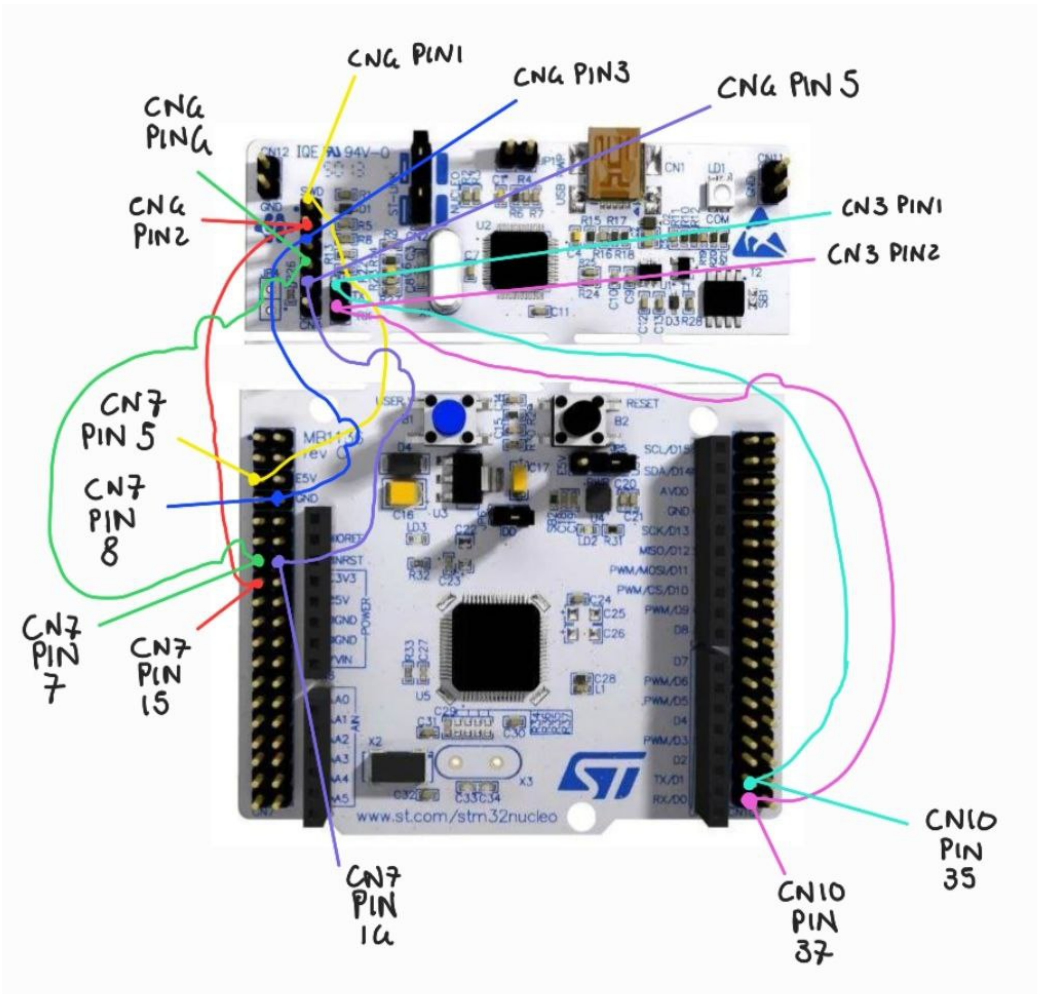


Figure 3.44: Connections between detached ST-Link and Nucleo Board

To conclude other two modifications must be taken:

- External Power Supply: Since, with the ST-Link attached the Nucleo was powered via the USB cable when it was plugged into a PC, now that it is detached the Stand-alone Nucleo board doesn't have access to the power supply. To provide a 5 v power supply there's a 3-pin jumper in the connector JP5 normally attacked to the U5V(USB5v) configuration, because that connection takes the supply directly from the USB in the ST-Link. Now the jumper must be put in E5V(External5V) since the Nucleo board will be powered via the E5V pin(PIN 6 of the CN7 morpho connector on the Nucleo Board);
- Connection of UART pin to the MCU in the Nucleo Board: with the ST-Link connected normally the Serial interface is created using the UART in the MCU present in the ST-Link. Now that the ST-Link is detached it must be provided the connection between PA2 and PA3 connectors of CN10 morpho Pin Header and the MCU of the Nucleo board this time. To do so two solder jumpers, the SB62 and the SB63 must be soldered.

In the picture below the LED diode is shown too, it is only a red LED to indicate that the board receives correctly the 5v supply voltage from the OBC board.

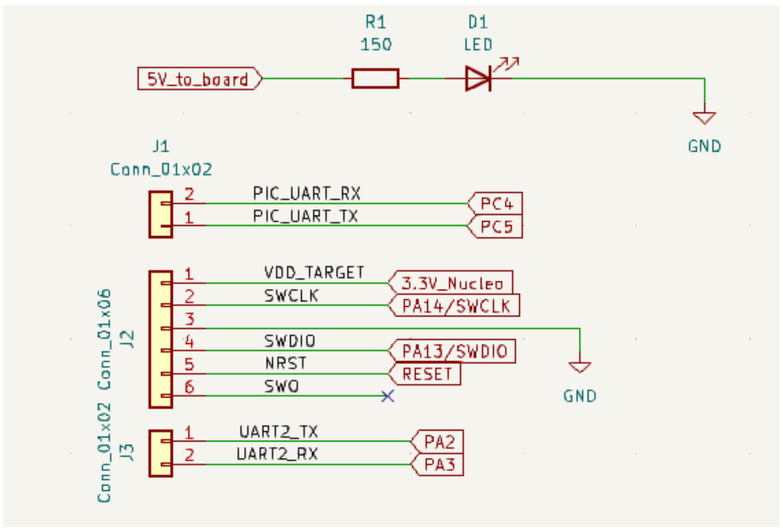


Figure 3.45: Other Connectors+diode schematic

3.7.2 Power Lines

Power in the ADCS board is distributed following the scheme in figure 3.46:

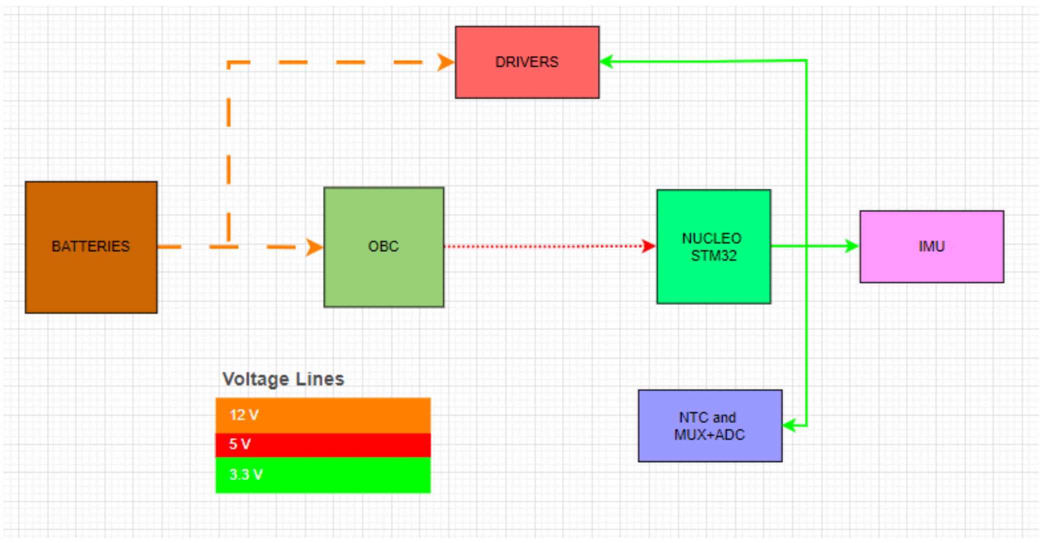


Figure 3.46: Power Lines schematic

- **12 V line**: Starting from the battery, it is a Lithium-Ion one composed of 3 cells ([18]) and is capable of delivering current up to 2Ah.
  Its wires are connected through a connector to the OBC board. The connector is the one shown in Figure below 3.47, which is a screw connector with 4 screws to block the wires from one side and the other side once they make contact. From one side the connector blocks the soldered wires coming from the OBC board, on the other side it is possible to attach or detach the wires of the battery using a screwdriver.

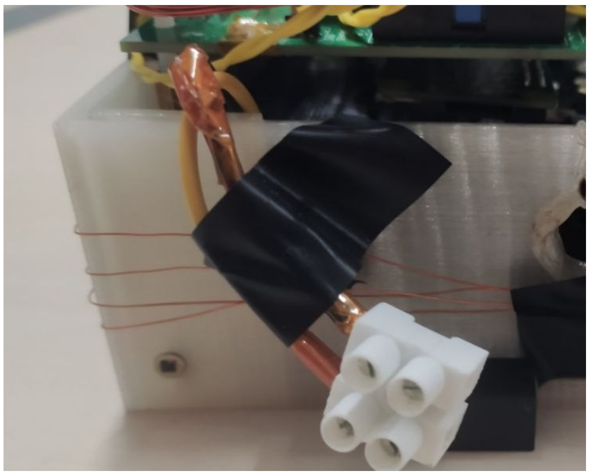


Figure 3.47: Battery connector

The 12 v line then goes directly to the ADCS board through the PCI bus and supplies all actuators from Drivers. Before being distributed among the drivers it passes through a fuse, which is the one on the top right in the figure 3.35. It is a fuse from LittleFuse [19], with a nominal current of 5A and a nominal voltage of 32 V. It was chosen, with that nominal current value, to suppose the case when all drivers would absorb 1 A of current. This is a remote case but since the board could be used to drive even Reaction Wheels other than Magnetorquers this is not a so borderline scenario.

- **5 V line**: the 5v line is obtained through a voltage regulator in the OBC board that converts the 12 v line. The regulator is a MAX20404AFOC/VY+ from Analog Devices [20], and as indicated in the datasheet it's a buck switching regulator capable of delivering up to 6A with an input voltage range from 3v to 36v. This line goes directly to the Nucleo Board E5V pin.
- **3.3 V line**: The 5V line comes directly to the Nucleo board regulator that provides the 3.3 v level to all the ADCS board components. This last regulator, as stated in the datasheet [21], is a quiescent current and low noise voltage regulator capable of delivering up to 500 mA of current when the input voltage ranges between 1.5 v and 5 v. These last lines power actually all the components present in the ADCS board: IMU, Drivers block, and NTC + Conditioning Circuitry Block.

3.7.3 IMU

In 3.48 is shown the circuitry for the IMU. As described in [22] the MTi has two supply pins: V_{DDA} and V_{DDIO}. They can be supplied independently or tied together to adapt to the application:

- Main supply voltage (V_{DDIO}) : The V_{DDIO} pin is the main supply of the device. This pin is connected to all the digital IO's and powers the processor.
- Analog supply voltage (V_{DDA}): The V_{DDA} pin of the MTi is connected to all the power supply pins of the sensing elements. There is no low-dropout regulator (LDO) on this type of device.
- Single power supply configuration: the V_{DDA} and V_{DDIO} supply pins can be connected to the same power supply. When the device is supplied with a single power supply source,

in this case with a 3.3 v supply, it is strongly recommended to decouple the V_{DDA} and V_{DDIO} supply pins, for example with a Resistor, for the best sensor performance. This way the digital circuitry will not influence the analog sensing part. Considering the minimum operating voltage for VDDA, the single supply voltage VDD should be at least 2.2 V, due to the voltage drop across the resistor.

In this case, a low pass filter circuit is used, with a 27 Ω resistance and a capacitor of 470 nF which allows a cutoff frequency of 10 kHz(considering the internal 100 nF capacitor). Moreover, a 100 nF capacitor is put near the MTi V_{DDA} and V_{DDIO} pin on the PCB to improve the decoupling.

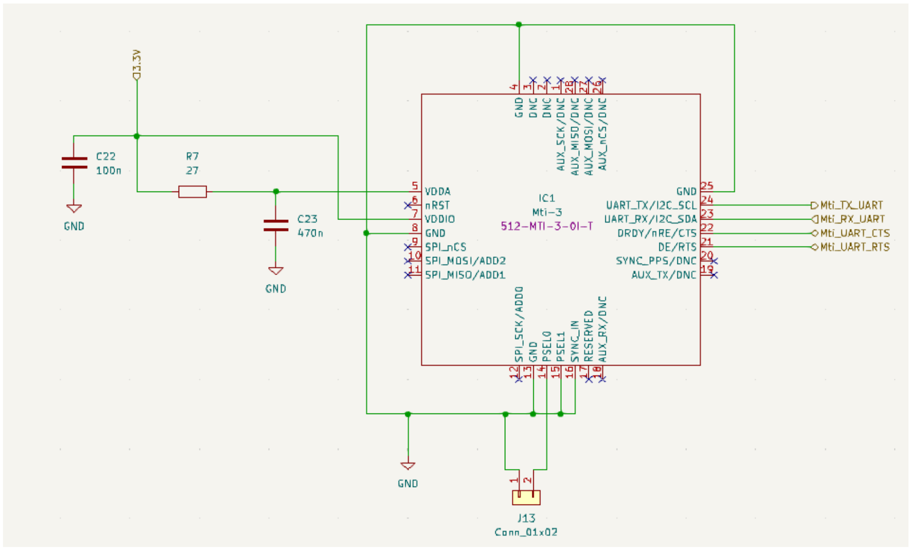


Figure 3.48: IMU Schematic

Regarding the PSEL1 and PSEL0, as described in the table 3.14 at least PSEL1 must be put to Ground always, instead PSEL0 can be high or low depending on the type of UART you want to use. Since both the pins are pulled up if nothing is attached to them their state is put to High. So PSEL1 is put directly to GND instead PSEL0 is connected to a pin header that can create a path to GND using a jumper or left as it is to keep a High state. Due to this last consideration, it can be seen that the signals taken for the UART communication are either the usual Tx and Rx and the two signals CTS and RTS for the full-duplex mode.

3.7.4 Drivers

Here the schematic of the actuator's driver is presented. In particular, the implementation of the design choices previously discussed in the 3.6.6, is shown. The most important things considered for this schematic are:

- 1) Connection between the resistor and potentiometer
- 2) Capacitors connected to V_M pin.

For the first point, it must be possible to vary the potentiometer value between 0 and its maximum value. To do so two of the three pins are tied together so that the potentiometer becomes a two-poles device and then connected in series with the resistor. So when the potentiometer has a 0 value across the two elements the corresponding value is the one of the resistor and otherwise it is the sum with the resistor.

For the second point, two capacitors are connected to the V_M pin, which is the actuator supply voltage line (12 v). Due to the high current path (orders of 1-2 A), a 47 uF capacitor is the **Bulk Capacitor**, whose design choice will be discussed later, and the other one, a 100 nF capacitor, is the one for the decoupling.
In the figure 3.50 the overall schematic of the Drivers Block is shown, in particular, specific attention must be paid to the capacitors put between the two output driver pins. In fact, it is suggested to have always at least one decoupling capacitor ranging between 0.01÷0.1uF as close as possible to the actuator wire if the mechanical load is a DC Motor for example, to protect the motor from undesired radio frequency EMI caused by arcing between the brushes and commutator.
To have the best effect the capacitors should be placed on the actuator, otherwise this is less effective at higher frequencies because the wires from the board to the motor will still be able to radiate EMI.
In any case, it's better to have them even for Magnetorquers that, as we have seen have long wires.
So that's why two parallel 10 nF and 100 nF are placed.

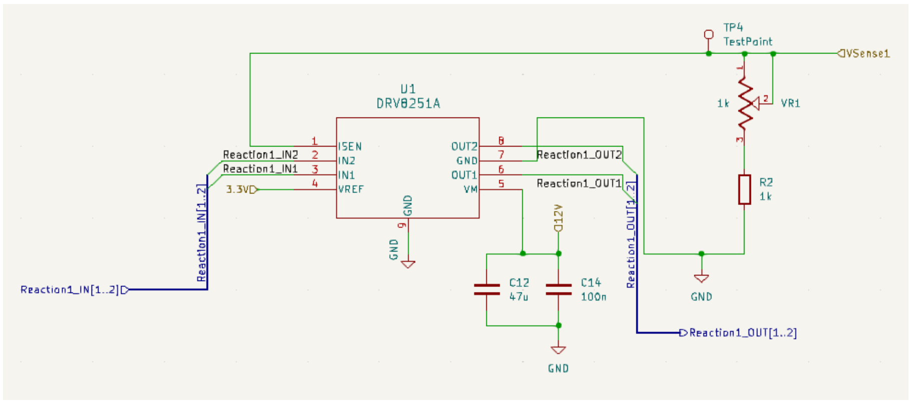


Figure 3.49: Single Driver Schematic

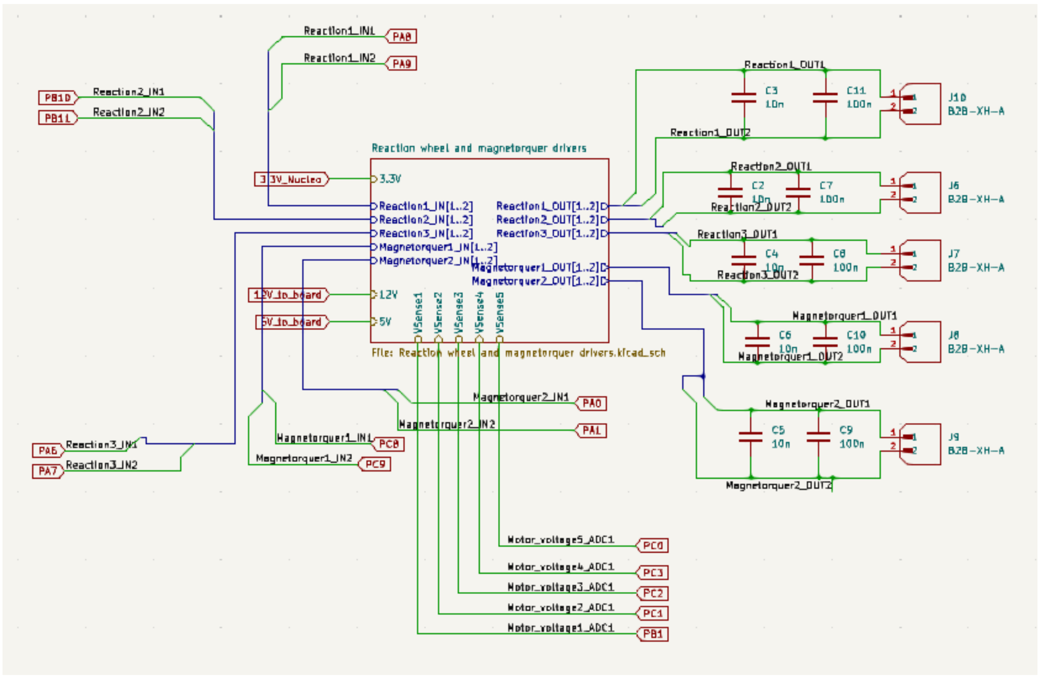


Figure 3.50: Drivers Block Overall Schematic

3.7.5 ADC+MUX

The most important thing to say about the chain MUX + ADC is that since the selected signal of the MUX comes from a simple voltage division conditioning circuit(for the NTC sensors), the signal is referred directly to the ground and no Differential Input configuration is needed for the ADC. So, as can be seen in the figure 3.51 the ADC mux has:

- The REFIN(+) pin to Vcc and the REFIN(-) pin to GND to VREF of the ADC ranges between 0v and 3,3v;
- The AIN(+) pin connected directly to the output of the MUX and the AIN(-) pin to GND: these two pins represent the actual analog input signal of the ADC, coming from the conditioning circuitry of the selected NTC;
- Two decoupling capacitors for the supply voltage connected to the VDD pin;
- Chip Select(CS*),Serial Clock(SCLK), DOUT and DIN connected to the Nucleo Board. All of these signal lines form the lines for the SPI communication.

The 4 select signals of the MUX are connected to 4 digital output pins of the Nucleo board but only 3 of them are used to select one of the input pins(Yi) from the first to the eighth: in figure 3.52 the functional table of the mux from its datasheet [10] tells that the forth select signal must be used only to disable the outputs and so it is not used in the application.

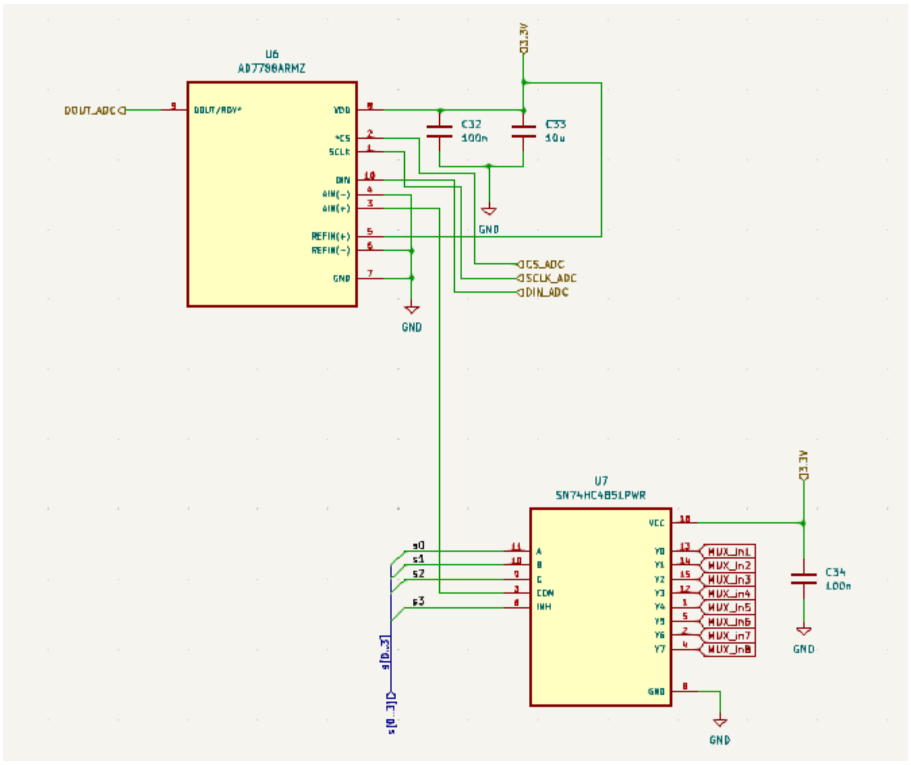
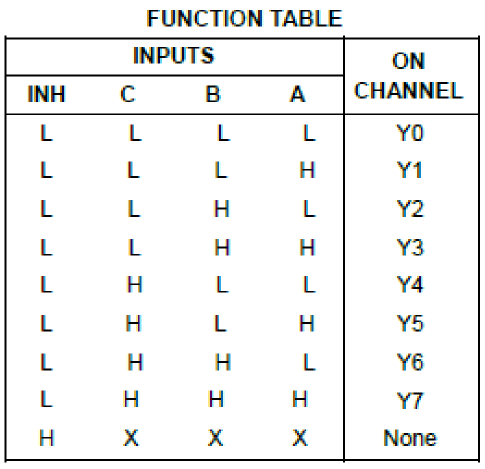


Figure 3.51: ADC+MUX schematic



FUNCTION TABLE

INPUTS				ON CHANNEL
INH	C	B	A	
L	L	L	L	Y0
L	L	L	H	Y1
L	L	H	L	Y2
L	L	H	H	Y3
L	H	L	L	Y4
L	H	L	H	Y5
L	H	H	L	Y6
L	H	H	H	Y7
H	X	X	X	None

Figure 3.52: Mux Function Table

3.7.6 PCB Layout

In this subsection, it will be described all the choices made in terms of signal traces, power traces, and power plane size. The PCB is made of two layers, the top and the bottom. On the top, there are all the components (SMD and Through Hole), instead on the bottom only the pin socket for the PCI bus(figures 3.53 and 3.54).
Two power planes are present in both faces, the top has the 3.3v plane plus a section dedicated to the ground plane and the bottom has the ground plane. The two ground planes are connected through vias.
The figures showed before 3.31 and 3.32, which are the ones of the last PCB project on KIcad, have the 4 holes near the top and bottom corner with a greater diameter with respect to the actually printed board: in fact, due to an error the PCB was printed with the holes of **M2**

type(2 mm), instead, the last images are modified by the project to have **M3** holes(3mm) which is the standard to stack one over the other all the electrical boards of a CubeSat.
To overcome the problem a driller was used to enlarge the 2mm holes.

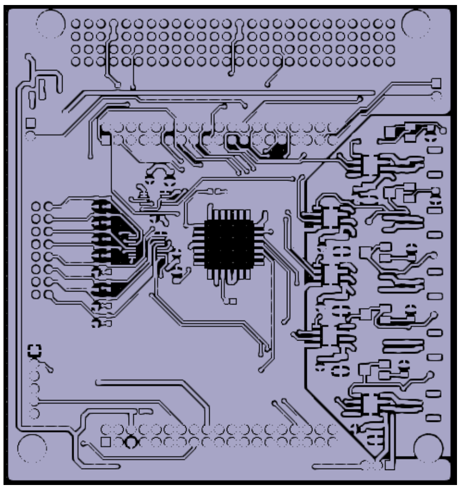


Figure 3.53: Top layer

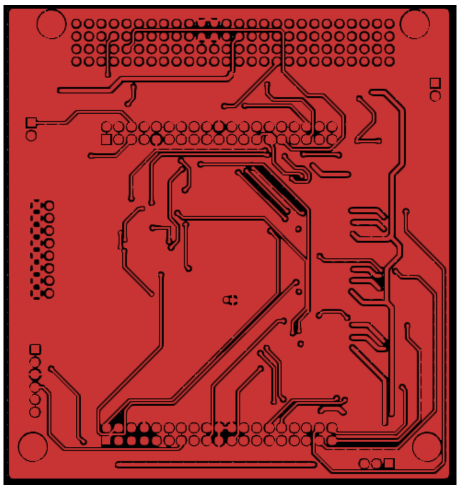


Figure 3.54: Bottom layer

Traces and Planes

Since on the board are present signal and power lines, their sizes differ due to the amount of current they carry.
In table 3.10 are shown all the widths of the traces: as expected the one for the signal is thinner than the power traces.
To calculate trace width an industry standard was considered, the IPC 2221 [23]: this standard provides some guidelines for the design of rigid printed circuit boards, so it is taken as a point of reference for the design of this PCB. It provides a formula for the calculation and a single graph to compute the current of traces. Moreover, a site is available([24]) for the calculation online of the trace width according to some parameters, following the IPC 2221 guidelines.
The formula for the current (in Ampere) is:

$$I = K \cdot \Delta T^B \cdot A^C, \tag{3.14}$$

where K = 0.0048, ΔT measures the Temperature Change in Degree Celsius[°C], the values of B and C parameters are respectively 0.44 and 0.725, adimensional. K, B, and C are constants resulting from IPC-2221 curves. Finally, the area of the trace's cross-section is represented by the symbol A, measured in $mils^2$.
Knowing the amount of current I flowing through the line it is possible to invert the 3.14 to obtain the area of the trace cross-section A:

$$A = \sqrt[c]{\frac{I}{K \cdot \Delta T^B}} \tag{3.15}$$

From this last value, knowing the weight of the copper the trace width is then calculated as:

$$W = \frac{A}{Thickness \cdot 1.378}, \tag{3.16}$$

where the Thickness of the PCB,realized by **JLCPCB**, is 1 oz and the number 1.378 is measured in $mils/oz$. The found width value is then converted to mm. It is an indication of the minimum width required to limit the temperature change under the set ΔT value.

Referring to the cited site for the calculation of the Trace width for the power lines a minimum trace width of 0.781 mm was found. Since it is a minimum value it was fixed to 0.8 mm and used for all power lines. The reason is that, as explained in subsection 3.7.2, the 12v battery is the only line that carries all the current to the boards: the OBC converts the 12v line to the 5v line with a voltage regulator. This line is then brought to the Nucleo voltage regulator which brings the 3.3v regime to all the ADCS circuitry. For the signal traces the width should oscillate between 0.17 and 0.30 mm depending on the case. However, we choose the use the highest value to carry the most possible current amount.
Here a summary of the trace widths is shown:

	Signals	**3.3v line**	**5v line**	**12v line**	**Gnd**
Width	0.3 mm	0.8 mm	0.8 mm	0.8 mm	0.8 mm

Table 3.10: Traces widths

For the 3.3v line, the width is the same as the other voltage regimes but as can be seen in figure 3.57 only a few traces were used, which were the ones to connect the 3.3v line to the drivers since they are surrounded by the side ground plane section. Moreover having a power plane simplifies the routing since all the pins connected to that line in the schematic are automatically connected by Kicad once the plane is designed.

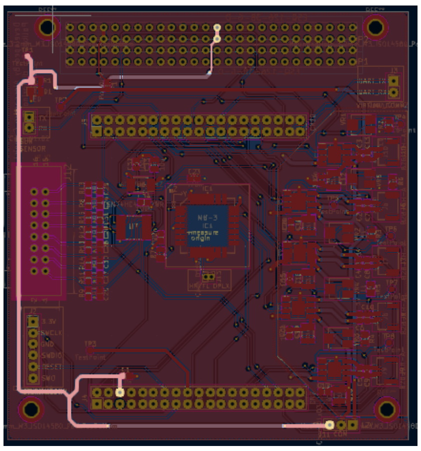


Figure 3.55: 5v line

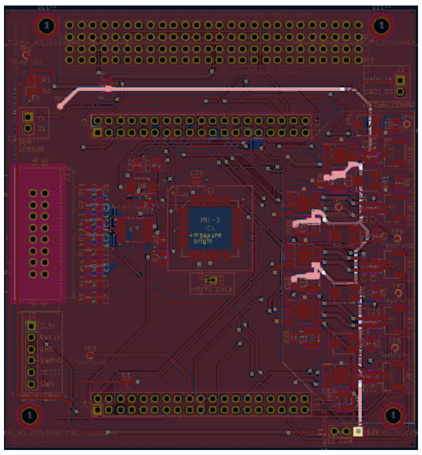


Figure 3.56: 12v line

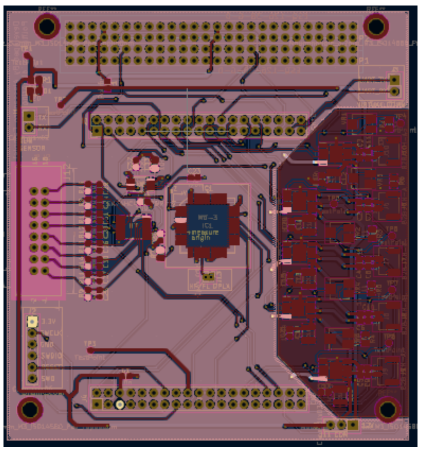


Figure 3.57: 3.3v Top Power plane and Traces

The same situation of the 3.3v applies to the GND plane which is spread throughout the bottom layer and in the side right portion of the top layer(figures 3.58 and 3.59) and only a few traces were needed to connect some pins to ground.

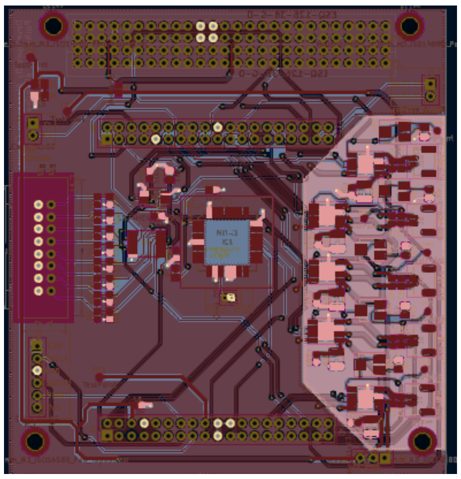


Figure 3.58: GND Top Plane e Traces

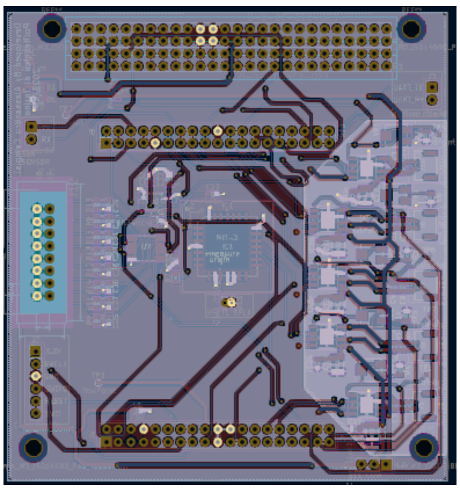


Figure 3.59: GND Bottom Plane e Traces

Drivers

- **Potentiometer dimension** : Dimension of resistor and potentiometer forming the R_{IPROPI}
  For the first thing, the problem is that potentiometers are usually not so small due to technological process choices to guarantee good performance. However as will be explained later the space available for both the resistor and potentiometer is limited, considering a total of 10(two per driver) must be soldered.
  So the chosen potentiometer must be small enough to be easily placeable on the board and easily accessible to be able to change the resistance value
- **Bulk Capacitance Design**: in the datasheet [17], in the Application information section(Typical Application) is suggested to use a bulk capacitance of 47 uF if the V_M ranges in the 4.5÷48 V interval.
- **Layout choices** : In the datasheet [17] some suggestion are given about the layout choices. The driver integrates MOSFETs capable of driving high currents, so particular attention should be paid to the layout design and external component placement. Some of the suggestions are:
  - 1) Low ESR ceramic capacitors should be utilized for the VM to GND bypass capacitor. X5R and X7R types are recommended;
  - 2) The VM power supply capacitors should be placed as close to the device as possible to minimize the loop inductance.
  - 3) The VM power supply bulk capacitor can be of ceramic or electrolytic type, but should also be placed as close as possible to the device to minimize the loop inductance.
  - 4) V_M, OUT_1, OUT_2, and GND pins carry high currents. So thick metal routing should be utilized for these traces as is feasible.
  - 5) The device thermal pad should be attached to the PCB top layer ground plane and internal ground plane (when available) through thermal vias to maximize the PCB heat sinking.

- 6) The copper plane area attached to the thermal pad should be maximized to ensure optimal heat sinking.

All of these suggestions have been applied to the PCB section of the driver to reduce risks of damage or overheating. The result is shown in the zoomed image below.
As mentioned before this plane, which is a ground one connected to the Thermal Pads(Figure 3.60) of the drivers is connected through vias to the bottom ground plane so the space for correct thermal dissipation should be enough.

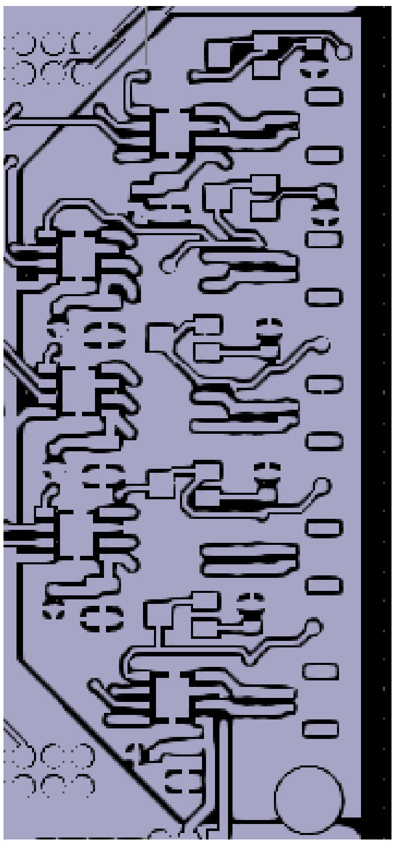


Figure 3.60: Drivers Top Layer Section

IMU

About the layout of the IMU, the only caution to be taken was to leave the space under the component free, as stated in the document [22], to prevent the current flowing underneath from altering the performance of the Device.

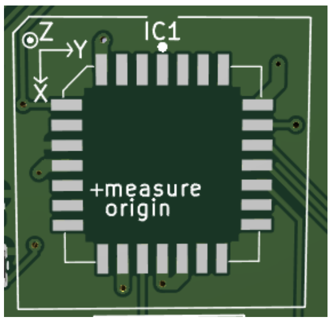


Figure 3.61: IMU Footprint

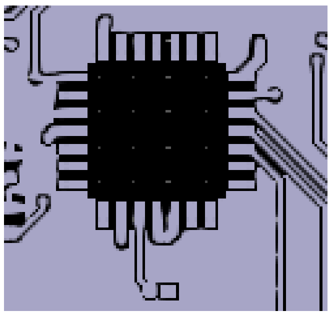


Figure 3.62: IMU Top Layer zoom

Chapter 4

Firmware Description

In this chapter will be presented the Environment used to develop the firmware of ADCS board, the main flow charts and an insight about the choices made on tasks management and synchronization.

4.1 CubeIDE Environment

STM32Cube [25] is an STMicroelectronics original initiative to significantly improve designer productivity by reducing development effort, time, and cost. STM32Cube covers the whole STM32 portfolio,it includes a set of user-friendly software development tools to cover project development from conception to realization, among which are:

- **STM32CubeMX** : a graphical software configuration tool that allows the automatic generation of C initialization code using graphical wizards
- **STM32CubeIDE**, an all-in-one development tool with peripheral configuration, code generation, code compilation, and debug features

4.2 FreeRTOS

FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller or microprocessor but it is used widely in many other applications.

4.2.1 FreeRTOS memory management

In Freertos,[26] it is possible to decide how to use memory: whether it is preferable to use static or dynamic memory allocation depends on the application, and the preference of the application writer:

- **Dynamic Memory** : The possibility to create objects dynamically allows to reduce complexity(fewer function parameters are required) and RAM usage. The memory allocation is done automatically and its scheme can be chosen to fit the specific applications needings.
- **Static Memory** : Create objects allocated statically gives more control to the application writer and he must not worry about memory allocation fealures. RTOS objects can be placed at specific memory allocations.

There are 5 implementations of memory allocation scheme and depending on the necessity the user can use one of these schemes or create its own:

- **HEAP 1**: used in small embedded systems, it only creates tasks and other kernel objects before the scheduler starts. **Fragmentation** is not possible and memory remains allocated until the applications expires its lifetime. Each created task requires a task control block (**TCB**) and a stack to be allocated from the heap.
- **HEAP 2**: widely used for backward compatibility, but its use is not recommended for new designs.As in HEAP 1 an array is allocated at run-time. It does not combine adjacent free blocks into a single larger block, so it is more susceptible to fragmentation. It is suitable for applications that create and deletes tasks repeatedly, provided the size of the stack allocated to the created tasks.
- **HEAP 3**: it uses the standard library **malloc()** and **free()** functions. Its size is defined by the linker configuration. It makes **malloc()** and **free()** thread safe by temporarily suspending the FreeRTOS scheduler.
- **HEAP 4**: It is an improvement of HEAP 2 but with tow differences:
  - It uses the first free block of memory that is large enough to hold the number of bytes requested
  - Combines portion of free memory to eliminate **fragmentation**.

  The heap contains three blocks of free memory that, in the order in which they appear in the array, are 5 bytes, 200 bytes, and 100 bytes, respectively.
- **HEAP 5**: it uses the same algorithm of HEAP 4 to allocate and free memory and it is not limited to allocating memory from a single statically declared. Useful when the RAM provided by the system on which FreeRTOS is running does not appear as a single contiguous (without space) block in the system's memory map array

4.2.2 Tasks Management

Every application built using FreeRTOS consists of many tasks. Each task that runs can exists in 2 main states as shown in Figure 4.1:

- **Running**: only one task can be in the Running state;
- **Not Running**: all the other tasks that aren't running stay in this state.

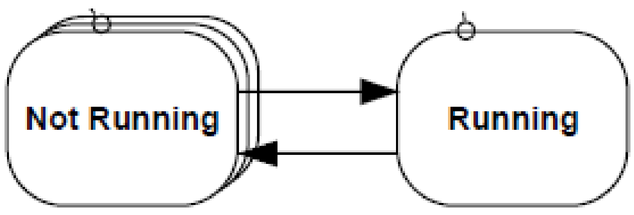


Figure 4.1: The two main states of a Task

When a Task is in the Running state the processor executes its code,instead in the other state the Task is in "Idle",waiting to get its turn to be runned.
The Not Running Task status is saved to be ready to resume execution the next time the scheduler decides it should enter the Running state,so when a Task resumes execution, it does from the instruction it was about to execute before it last left the Running state.

The FreeRTOS scheduler is the only entity that can switch a task in and out. In FreeRTOS priorities among tasks are important and 9 levels of priority are defined (es: **osPriorityNone** = 0,**osPriorityIdle** = 1,**osPriorityLow** = 8, **osPriorityBelowNormal** = 16, **osPriorityNormal** = 24, **osPriorityAboveNormal** = 32, **osPriorityHigh** = 40,etc...),each level than has 7 sub-levels.

When the scheduler gives the "context" to a specific Task, between the execution of a Task and the other there's a time interval to wait. This time interval is so small in real life that it seems like every task is running concurrently but at cpu time it is not. This time interval is obtained relaying on a "tick interrupt" which is a periodic interrupt: the length of the time slice is set by the tick interrupt frequency,configurable in FreeRTOS.
The Scheduler decides which Task through tasks **priorities**,in particular the highest prority Task occuring after that time slice expires is the one that will get processor attention to be executed. So it is important,depending on the application and on what that Task will perform in terms of operations, to determine which are the priorities of each task. This can lead to a problem called **Task Starvation** : if a task with higher priority never stops , a lower priority task will never be executed.
In order to avoid Task Starvation usually tasks are designed to be **Event Driven**:they perform their operations based on an event and after the execution they will wait for the next event.
A task that is waiting for an event is said to be in the **Blocked state**: it is a Sub state of the Not Running state Tasks can enter the Blocked state to wait for two different types of events:

- **Temporal(time related) events**: the event being either a delay period expiring, or an absolute time being reached. For example, a task may enter the Blocked state to wait for 10 milliseconds to pass;
- **Synchronization events**: where the events originate from another task or interrupt. For example, a task may enter the Blocked state to wait for data from a peripheral.

It is possible for a task to block on a synchronization event with a timeout. For example, a task may choose to wait for a maximum of 10 milliseconds for data to arrive from a peripheral. The task will leave the Blocked state if either data arrives within 10 milliseconds, or 10 milliseconds pass with no data arriving.
Suspended is also a Sub-state of the Not Running state. In the **Suspended state** Tasks are not available to the scheduler. The only way to enter the Suspended state is through a call to **osThreadSuspend()**.The only way out is a call to the **osThreadResume()**.
Most applications do not use the Suspended state.
Tasks that are in the Not Running state but are not Blocked or Suspended are said to be in the **Ready state**. They are able to run, and therefore "ready" to run, but are not currently in the Running state.
In figure 4.2 a general Block scheme of all the states is shown:

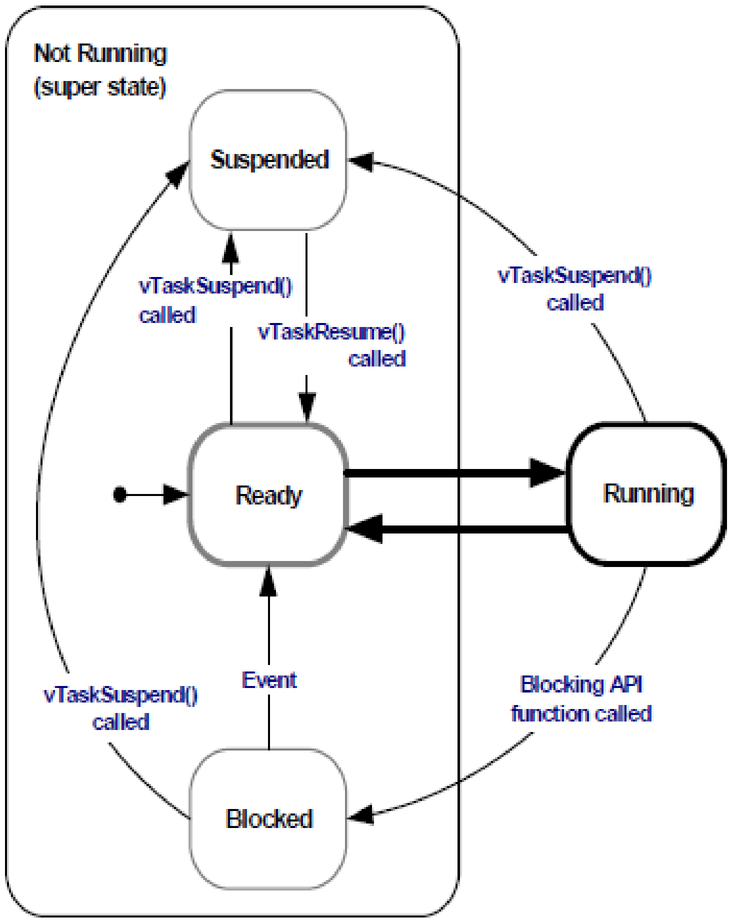


Figure 4.2: State Diagram

It is possible to place a task in the Blocked state for a specific time to avoid Starvation due to Busy-waiting(and polling): **osDelay(uint32_t ticks)**can be used to put the task in the blocked state for a specific amount of ticks(1 tick = 1 ms).
As it can be imagined it is not possible to not have at least one task running, so there must be always one Task in the Running state: FreeRTOS always creates by default a Task called **Idle Task** which will run after every actually implemented task finish its execution. It is possible to use the Idle Task to perform some operations anyway:Idle task performs some operation related to the OS. For example, it is responsible for cleaning up kernel resources after a task has been deleted.

4.2.3 Tasks Synchronization mechanisms

RTOS provides various mechanisms of concurrency and synchronization, such as **mutexes**, **semaphores**, **queues**, **events**, **signals**,and **message passing**. In this discussion we will concentrate only on the first three. These mechanisms allow tasks or threads to communicate, share data, synchronize execution, and avoid race conditions or deadlocks.

Mutexes

In a multitasking system there is potential for error if one task starts to access a resource, but does not complete its access before being transitioned out of the Running state.
If the task leaves the resource in an inconsistent state, then access to the same resource by any

other task or interrupt could result in data corruption, or other similar issue.
To ensure data consistency of a resource that is shared between tasks, or between tasks and interrupts, must be managed using a **mutual exclusion** technique.
Once a task starts to access a shared resource that is NOT **thread safe**(safe for use from multiple threads; operation is protected by an RTOS primitive and can be used from multiple threads, but will cause problems if used from an interrupt service routine), the same task has exclusive access to the resource until the resource has been returned to a consistent state. The best way to avoid this issues is to not share resources, but it is not always possible.

Critical sections are regions of code that are executed blocking the scheduler:in this way it is sure that this code is not interrupted by the scheduler and task transition is avoided. Critical section MUST be very short and during their execution Interrupts are not disabled.
If an interrupt requests a context switch while the scheduler is suspended the request is held pending and is performed only when the scheduler is resumed. However Suspending the scheduler is not the optimal solution: specific kernel objects can be used to perform **Mutual Exclusion(Mutex)** on data or resources.

The mutex can be thought of as a token that is associated with the resource being shared. A task can access the shared resource only after taking the token it must give back the token when the resource is no longer needed. Only when the token has been returned, another task can successfully take the token, and then safely access the same shared resource.
Infact mutexes are very similar to binary semaphores: the main difference is that the task that takes a mutex have to give it back. With semaphores, normally, another task will release the semaphore, and the one that took it will simply wait to be able to take it again.

A potential problem using mutexes could be: there are 2 Tasks running in an application and the higher priority Task 2 must wait for the lower priority Task 1 to give up control of the mutex. A higher priority task being delayed by a lower priority task in this manner is called **priority inversion**.
To overcome this problem Mutexes include a basic **priority inheritance** mechanism that is not present in Semaphores. Priority inheritance is a scheme that minimizes the negative effects of priority inversion: it ensures that the inversion is always time bounded. However Priority inheritance complicates system timing analysis, and it is not good practice to rely on itfor correct system operation.

It works by temporarily raising the priority of the mutex holder to the priority of the highest priority task that is attempting to obtain the same mutex. The low priority task that holds the mutex "inherits" the priority of the task waiting for the mutex. The priority of the mutex holder is reset automatically to its original value when it gives the mutex back. It's prerogative fo the application writer to enable priority inheritance, from the attributes structure.

Queues

Queues provide a **task to task**, **task to interrupt**, and **interrupt to task communication** mechanism. A queue can hold a finite number of fixed size data items. The **lenght** of a queue is the maximum number of elements a queue can carry.
Queues are normally used as **FIFO**(First In First Out) buffers,It is also possible to write to the front of a queue, and to overwrite data that is already at the front of a queue.
There are two ways in which queue behavior can be implemented:

- **Queue by copy**: the data sent to the queue is copied byte for byte into the queue

- **Queue by reference**: means the queue only holds pointers to the data sent to the queue, not the data itself

FreeRTOS uses the queue by copy method.

Queues can be accessed by any task or ISR that knows of their existence: any task can write to the same queue and likewise any task can read from the same queue. However it's more common for a queue to have multiple writers than multiple readers.

A task can block for a timeout when trying to read from a queue,instead a task in the Blocked State is automatically moved to the Ready State when another task or interrupt places data into the Queue. The task will be moved automatically from the Blocked State to the Ready State if the specified block time expires before data becomes available.

A task can optionally specify a block time when writing to a queue. If the queue is already full, the task will go in the Blocked State. The block time is the maximum time the task should be held in the Blocked State to wait for space to become available on the queue.

An important thing to consider is that a queue can have multiple writers, so it is possible for a full queue to have more than one task blocked on it waiting to complete a send operation: only one task will be unblocked when space on the queue becomes available and that task will be always the highest priority task that is waiting for space.If the blocked tasks have equal priority, then the task that has been waiting for space the longest will be unblocked.

Semaphores

Queues are used in case of data transmission. A more efficient way for synchronization when no data transfer is needed is available: Binary and Counting semaphores.

Semaphores are abstract type used to control access to resources by different threads. From the practical point of view they consist in a shared variable that is incremented and decremented, or toggled.
Semaphores which allow an arbitrary resource count are called **counting semaphores**.
Semaphores which are restricted to the values 0 and 1 are called **binary semaphores**.
The action of requesting access to the resources is normally referred to as "**taking**" or **acquiring**" the semaphore: take a semaphore means decrement its value. The action of freeing the resources is normally referred to as "**giving**" or "**releasing**" the semaphore: give a semaphore means increment its value.

Binary Semaphore can be used to unblock a task each time a particular interrupt occurs:the interrupt event processing is implemented within the synchronized task(only a very fast and short portion of code is in the ISR). The deferred task will try to take the semaphore, blocking if it is not available. The ISR will give the semaphore, moving the deferred task in the ready state.

It is possible to specify a timeout to take or give a semaphore. The thread will exit from the blocked state if the semaphore is released before the timeout or when it is reached. Indefinite timeouts are normally bad practice in real applications, because they make it difficult to recover from an error. It is possible to check for the return status to detect if a timeout occurred.

When a semaphore is created it is possible to set a parameter for the maximum count it can reach to a value greater than one In this case it is a counting semaphore. A counting semaphore can be taken more than one time, until it gets to 0. Counting semaphores are typically used for two things:

- **Counting event**: Counting events: an event handler will "give" a semaphore each time an event occurs. A task will " take" a semaphore each time it processes an event.
- **Resource management**:Resource management: the count value indicates the number of resources available. To obtain control of a resource, a task must first obtain a semaphore. When the count value reaches zero, there are no free resources. When a task finishes with the resource, it "gives" the semaphore back.

4.3 ADCS FreeRTOS Tasks Management and Code

In this section it will be discussed the choices made about Tasks Management. In particular the role that every Task has and the type of synchronization used between tasks.

It was decided to divide the application among 4 different tasks:

- 1)**CHECK/SAFETY Task** : this is the most important task, it is used to acquire measures of temperature from NTC sensors and to collect data about the currents in the actuators. Than it will monitor,depending on the values, to let the scheduler continue to work or interrupt it in case of high values of currents or overheating of some ICs. In this case an interrupt system will be used to manage all emergencies ;
- 2)**OBC-COMM Task** : this is the Task for communication with the OBC through the UART line ; It is used for data reception and sampling from the OBC board(Raspberry Pi) and data trasmission to the OBC board in order to visualize them on Grafana.
- 3)**Control Task** : this is the Core Task, where the alghoritm for the attitude control is executed ;
- 4)**IMU Task** : this is the Task for reading and collecting data about Angular Speed and Magnetic Field measurements from IMU.

4.3.1 ADCS Flow Chart

In this sub section the details about Tasks Flow Chart will be showed:

CHECK TASK

As explained at the beginning of this section this is the most critical Task,used to monitor the housekeeping parameters. In case of non-safety values some action must be taken in order to bring the systems in a safer state. In particular as the ADCS board stands, two kind of problems could overcome:

- **Actuator Overcurrent**: the task monitors all the currents of the actuators attached to the board, in particular in this application we have 3 Magnetorquers. Due to the fact that we have limited power resources(the battery cannot generate all the current that we want but it is limited and its capacity is not high) we must ensure that enough current is driven to every driver ,considering that we have a 3D application,so all the three Magnetorqeurs are active.

- **IC Overheating**: this is an extremely important question. Over heating of electronics parts is a huge problem in space electronics application and if not managed well it can lead to damages potentially irreversible to the board functioning. Given that, the task monitors the temperature of critical ICs too:
  - Nucleo board Processor(1): the processor is the most important part in this case because it is the core of the board, the scheduler cannot stop working due to an hardware damage since it is not possible to replace the damaged processor while the Cubesat is working. So if somehow the processor exceeds a critical temperature the board and so the whole system must be put in a temporary IDLE/SAFE state;
  - Actuator Drivers (5): as mentioned earlier of the 5 drivers available, 3 are actually used so 3 over 5 NTC sensors values are monitored. The overheating of the drivers is strictly linked with the overcurrent problem, the more current flows though an electronic devices the higher the temperature rises. This problem is managed reducing the amount of current given to a specific driver ,controlling the Duty Cycle of the PWM signals to that driver.

However,as mentioned here 3.6.7 the device enters a safe state every time there's an overcome of the junction temperature or an overcurrent, but we must ensure to not let the device enter in this mode because the CubeSat could be moving in the space and so the movement could be compromised due to this "auto-protection" of the drivers.
In figure 4.3 a simple Flow Chart of the code of the Task is showed. In particular the two problems of Overheating and Overcurrents are named respectively "Problem A" and "Problem B".
If no problem occurs or they are solved, the Tasks sends data(through queues) to the OBC task. These data are the **Local HouseKeeping** Data of the ADCS.

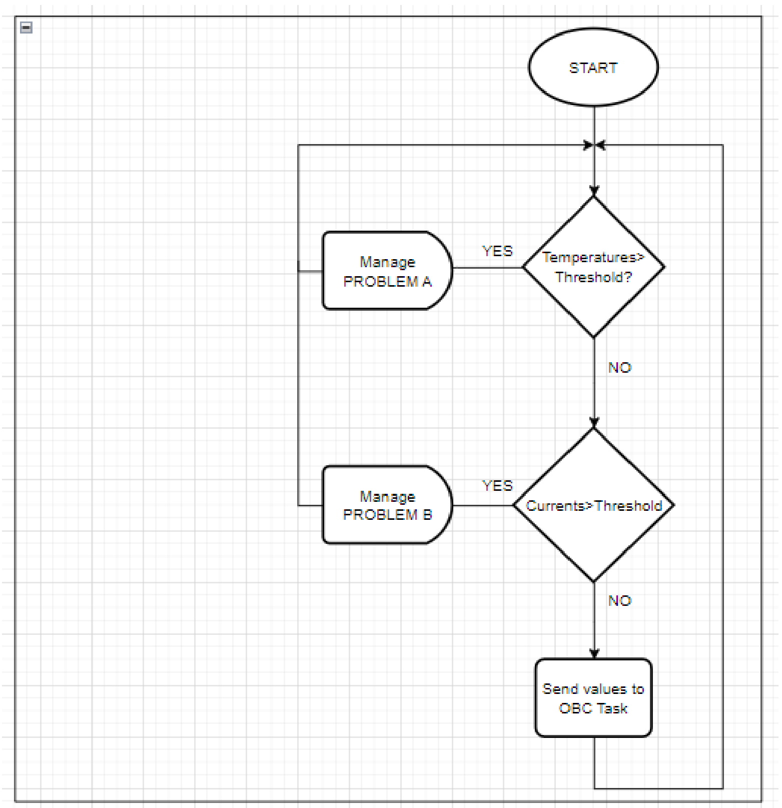


Figure 4.3: CHECK Task Flow Chart

In 4.4 is shown the flow chart of the management of the problem of overheating.

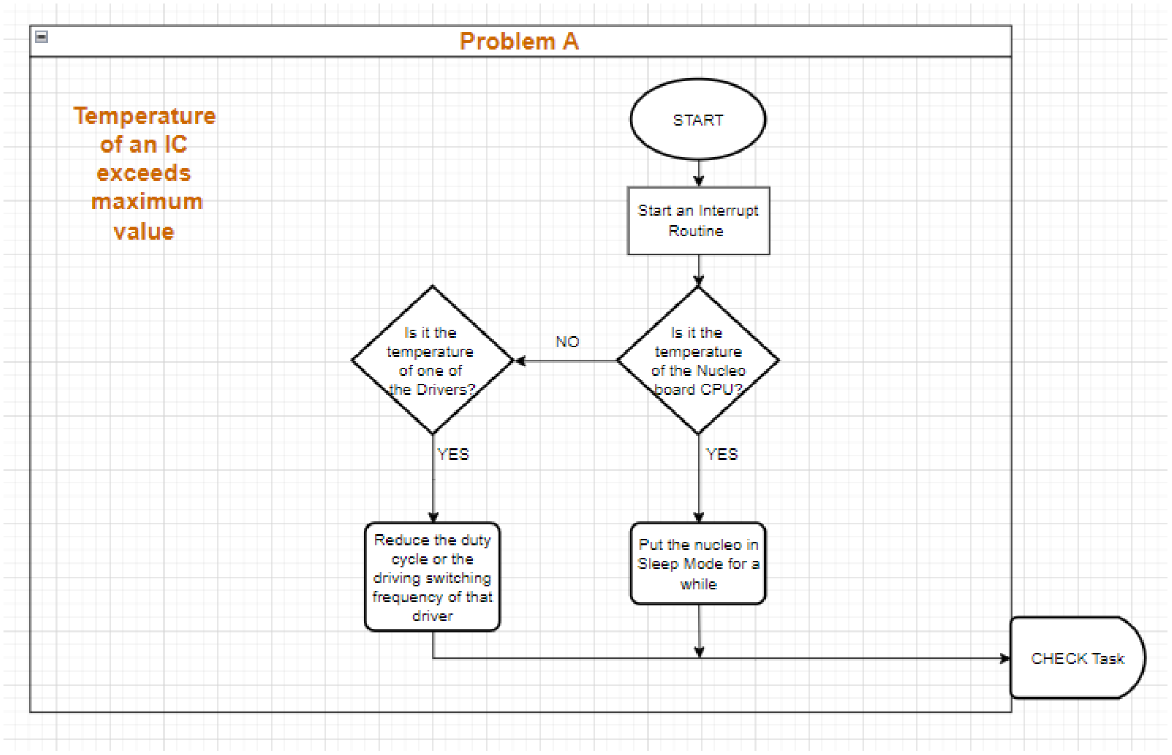

Problem A
Temperature of an IC exceeds maximum value
START
Start an Interrupt Routine
Is it the temperature of the Nucleo board CPU?
NO
Is it the temperature of one of the Drivers?
YES
YES
Reduce the duty cycle or the driving switching frequency of that driver
Put the nucleo in Sleep Mode for a while
CHECK Task

Figure 4.4: Problem A Flow Chart

Regarding the Problem A,as mentioned before there are 2 type of ICs whose temperature is monitored.

- For the Nucleo, to manage the overheating problem, putting the Board in a safe state, the board offers 7 options depending on the needing[8]:**Sleep mode**,**Low-power run mode**,**Low-power sleep mode**,**Stop 0**, **Stop 1** and **Stop 2** modes,**Standby mode**, **Shutdown mode**.
  Of all the options the one considered is the **Sleep mode**:In Sleep mode, only the CPU is stopped. All peripherals continue to operate and can wake up the CPU when an interrupt/event occurs.
  In this case the interrupt coincides with the return of the temperature to a normal value.

- For the Drivers,we know that the power dissipation is realted to the temperture. From its datasheet [17] the power dissipation is calculated summing up the contributions: P_{VM}, P_{SW}, P_{RDS}. The first contribution is essentially constant because depends on the nominal voltage and current specification of the motor power supply(in this case the battery), the second depends on the motor power supply,switching frequency and average output current and the last on the average output current. but this entails that the produced magnetic dipole and so the resulting torque will decrease, or by reducing the switching frequency. This last choice doesn't change the average output current but could increase the ripple on the current if the switching frequency is not close to the resonant frequency of the magnetorquer[16]. The safer choice so seems to be the frequency reduction, relaying on the fact that the control could overcome the added noise on the current.

Considering the Overcurrent problem(Problem B) in figure 3.57 we have a maximum current that can be given to the drivers, which is divided among the three actuators depending on their equivalent resistance (3.5),considering the maximum current deliverable by the battery. So the task checks that none of each driver takes too much current from the battery and in case of exceeding the current threshold the action to perform is reducing the Duty Cycle since it is the only way to reduce the current amount. After the current comes back to a safe value we can again increase it accordingly.

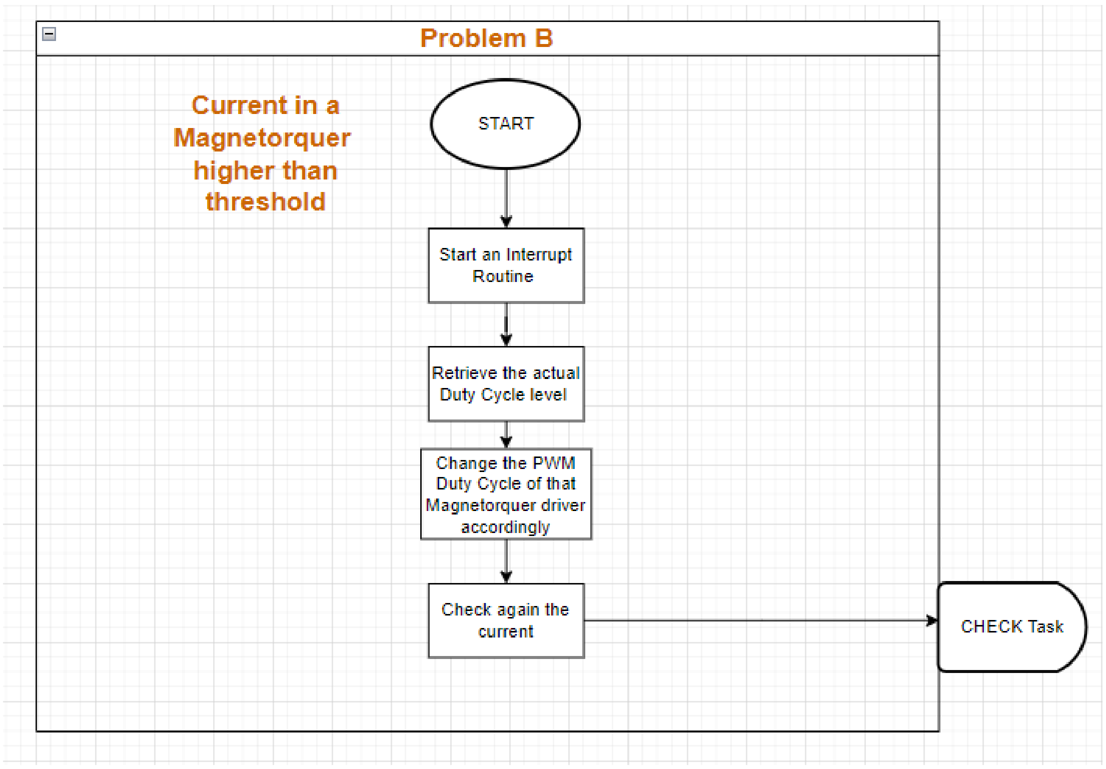

Problem B
Current in a Magnetorquer higher than threshold
START
Start an Interrupt Routine
Retrieve the actual Duty Cycle level
Change the PWM Duty Cycle of that Magnetorquer driver accordingly
Check again the current
CHECK Task

Figure 4.5: Problem B Flow Chart

OBC TASK

As explained the OBC task is responsible of acquiring all the Data sent by the OBC board and at the same time send to it local Housekeeping and Telemetry Data. Here's showed in figure 4.6 a simple Flow Chart of the Task. The data exchanged via UART by the OBC board are managed thorugh specific libraries and they will be explained later,instead the data received by the IMU Task are sent to the OBC Task via Queue.

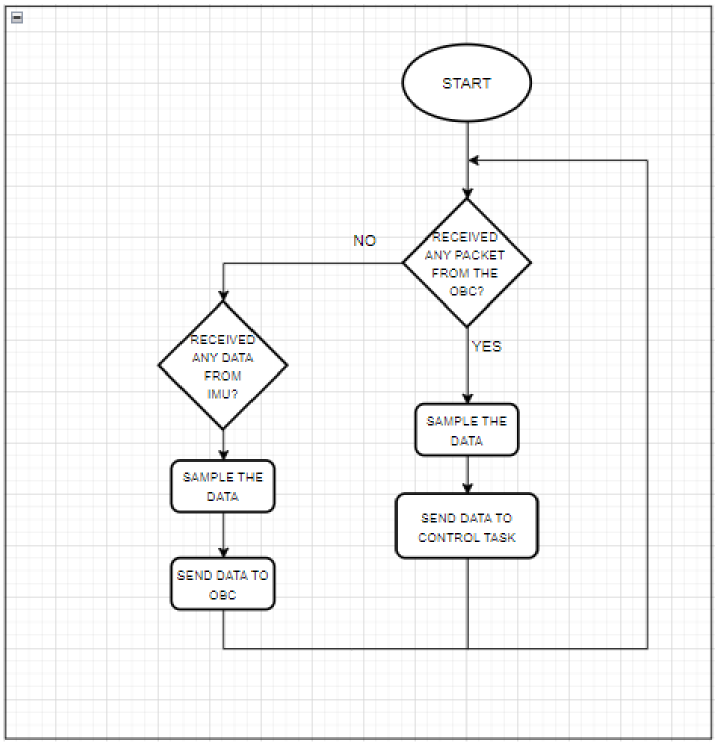

START
RECEIVED ANY PACKET FROM THE OBC?
NO
YES
RECEIVED ANY DATA FROM IMU?
SAMPLE THE DATA
SEND DATA TO CONTROL TASK
SAMPLE THE DATA
SEND DATA TO OBC

Figure 4.6: OBC Task Flow Chart

CONTROL TASK

The Control Task is the one that practically controls Magnetorquers and so it is able in the first place to drive them or not when it is needed.At the beginning it waits for any OpMode data coming from the OBC Task and depending on the value it manages 3 cases:

- **OPMODE = 0**: This is the IDLE case,where the Cubesats isn't moving because it doesn't receive any target speed. It can be seen as the initial state and the state to STOP the control.
- **OPMODE = 1**: This is the case where the control code is actually executed, in this Opmode the ADCS receives the target Speed data from the OBC Task and drive the Magnetorquers in order to reach the Target.
- **OPMODE = 2**: This Opmode can be seen as the "Initial test" state,where magnetorquers can be driven separately one after the other to control that they are responding to the PWM command from the IC drivers and for testing their **Polarity**. This is a crucial part since if it goes all well it allows to be sure that the control can be actually implemented.

Below is shows the Control Task Flow Chart:

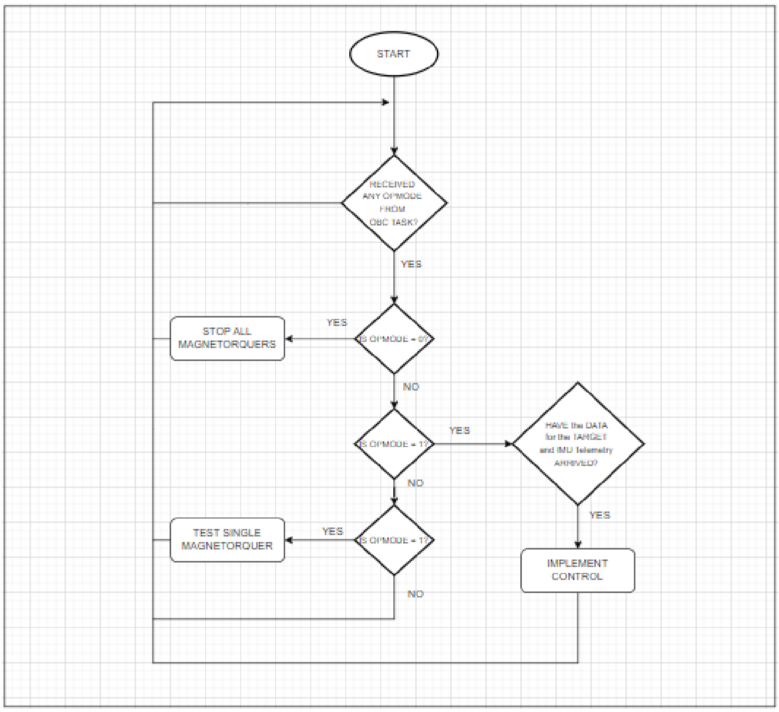


Figure 4.7: Control Task Flow Chart

IMU TASK

The Task that reads IMU values is extremely simple(in figure 4.8 the Flow Chart). After sampling the measured angular speed(Gyroscope) and magnetic field intensity(Magnetorquers) it sends them to the Control Task and the OBC Task via 2 distinct queues.

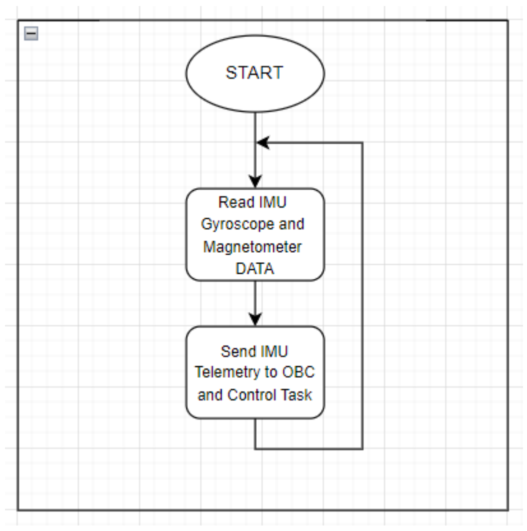

START
Read IMU Gyroscope and Magnetometer DATA
Send IMU Telemetry to OBC and Control Task

Figure 4.8: IMU Task Flow Chart

4.3.2 ADCS Code

In this section it will be explained the main firmware architecture and its elements.

Firmware Architecture

The firmware architecture that will be described includes libraries for the nucleo Drivers management,for the acquisition of temperature values from NTC sensors, the reading of angular speed and magnetic fields from IMU and for the driving of the Magnetorquers. Looking at the figure 4.10 we can define 2 layers of libraries:

- The top blue layer which includes all the library related to drivers of the Nucleo such as the timers library,the ADC and SPI one;
- The red layer which is related to the code that deals with the components of the PCB and not only the Nucleo board,so the NTC thermistors,the IMU, the electronic drivers and so on;
- The central freertos.c block is related to the library where all the tasks code is written.

In figure 4.9 the CubeMx software Nucleo Pinout is showed, in it we can notice the number of pins dedicated to timers peripherals, ten because we have 2 pwm signals per electornic driver, the five ADC1_INx pins dedicated to the acquisition of voltages on the Rsensing of the electronic drivers, the 8 UART Tx and Rx signals(two per each UART), the signals for the SPI protocol and some GPIO outputs for the Multiplexer selection.
To notice the fact the the CAN pins are not used in the application.

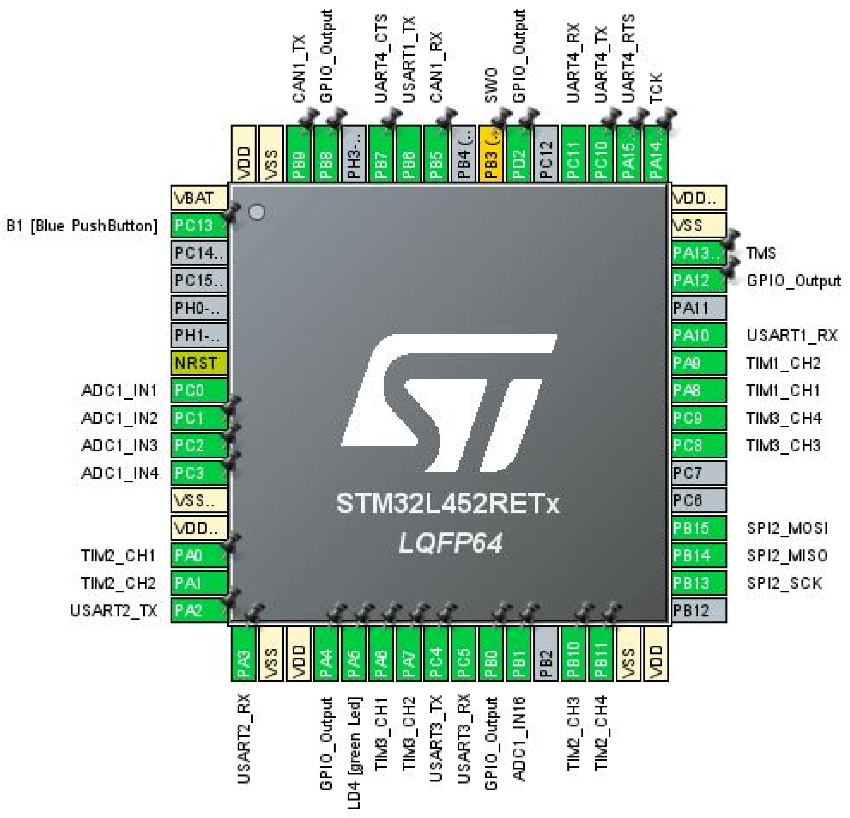


Figure 4.9: CubeMX Pinout of Nucleo

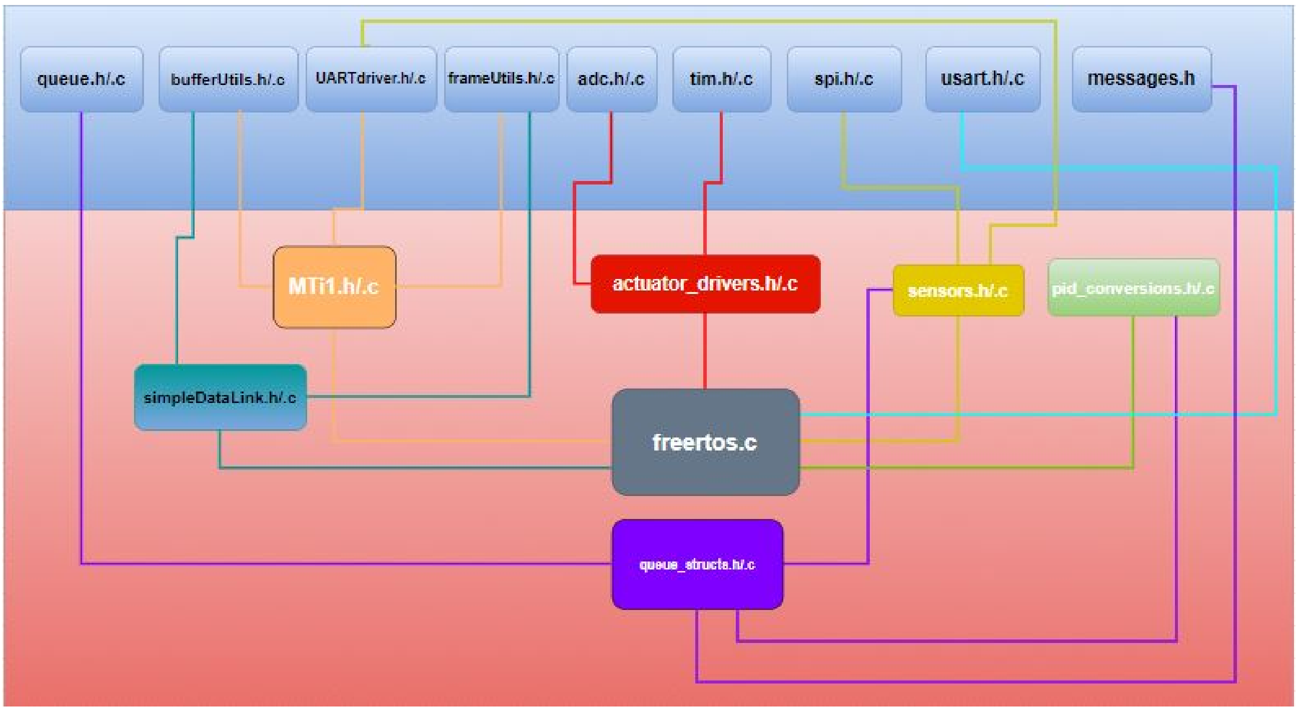


Figure 4.10: Firmware Architecture

Now we'll se them in details.

Blue Layer

- **queue.h/c**,**adc.h/c**,**tim.h/c**,**spi.h/c** and **usart.h/c**: are library created by the ST Nucleo software that manages rispecively the code for the usage of queues in Freertos, the internal A/D converter, the timer peripheral and the SPI and UART/USART protocols ;
- **UARTdriver.h/c**: it is a custom created library to enhance the pre-existing usart library ;
- **bufferUtils.h/c** and **frameUtils.h/c**: the first is a support library to manage all the buffers used in the UARTdriver library and the second implements functions to search frames insider serial buffers.
- **messages.h/c**: is a library containing all the opmode frame structure that are send between the OBC and the ADCS board.

Red Layer

- **MTi.h/c**:this library uses the UARTdriver library to implement the code for the UART communication with the Nucleo
- **simpleDataLink.h/c**: this library uses the libraries bufferUtils and frameUtils to implement the Tx and Rx function of the UART protocol that enhances the ones created by the Nucleo Software;
- **actuator_drivers.h/c**: this library implements the code to manage the control signals to the elctronic drivers and to read the actuators currents;
- **sensors.h/c**: this is the library to manage the communication via SPI with the external ADC for the A/D conversion of temperatures;
- **queue_structs.h/c**: this library implements custom general functions to work with complex structures that manages the reception of data via queue in the Tasks;
- **pid_conversions.h/c**: this library implements the code needed to realize the Control alghorithm for the Dethumbling maneuvre of the Cubesat, see 2.3;

The libraries **UARTdriver.h/c**,**simpleDataLink.h/cbufferUtils.h/c**, **frameUtils.h/c** and **MTi.h/c** comes from a previous thesis work [11] and were adapted to this Application.

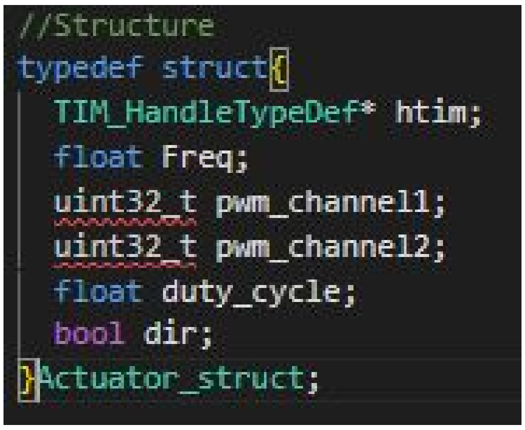
Now we'll see in details the main functions of the libraries related to the Red layer in the figure 4.3.

actuator_drivers.h

The main functions of the actuator_drivers library are:

- **void get_actuator_current(ADC_HandleTypeDef *hadc,volatile float voltagebuf[],volatile float currentbuf[],uint8_t Channels_mask[])**: takes as input an ADC handler, a buffer to upload the converted values of the ADC channels, obtains the currents (see 3.6.6) and uploads them in a buffer. The last parameter allows to select only certain channels over the five availables;

- **void update__pwm__Frequency(Actuator__struct *act)**:this function allows to update a timer's internal register to obtain a square wave with a certain frequency. It accepts as parameter a custom struct of type **Actuator__struct** that contains all the variables related to a specific driver and so to its two timer PWM channels;



```
//Structure
typedef struct{
  TIM_HandleTypeDef* htim;
  float Freq;
  uint32_t pwm_channel1;
  uint32_t pwm_channel2;
  float duty_cycle;
  bool dir;
}Actuator_struct;
```

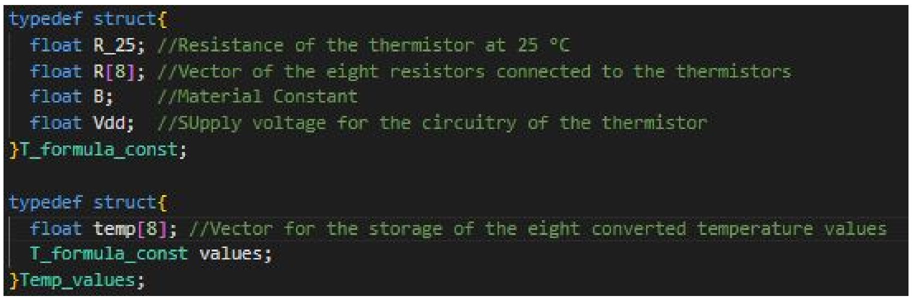
Figure 4.11: Handler for an actuator electronic driver

- **void update__duty__dir(Actuator__struct *act,float duty,bool dir)**: allows to change the Duty Cycle of the PWM square waves of a specific electronic driver(accepts the electronic driver handler **act**) and the direction of the current of that actuator by setting or resetting the third parameter(the logic behind is explained here 3.6.3):
  - dir = 1 -> Forward current direction;
  - dir = 0 -> Reverse current direction;

sensors.h

- **void select__input(uint8__t sel)**:allows to select which channel of analogic MUX send to the external ADC for the conversion;
- **float ADC__Conversion(SPI__HandleTypeDef *spi__struct,uint8__t mode)**: allows to perform the A/D conversion with the external ADC and returns the conversion of the single channel. The second parameter allows to select the conversion mode:
  - mode = 0 -> Continuous conversione mode
  - mode = 1 -> Single conversion mode
  - mode = 2 -> Continuous Read mode.

  See 3.3.1 for details about conversion modes.
- **void voltage__to__temperature__conv(float value,Temp__values *s1,uint8__t i)**: allows to convert a voltage values from the A/D conversion into a temperature value in Celsius degrees.
  See 3.3.4 for the calibration function.



```c
typedef struct{
  float R_25; //Resistance of the thermistor at 25 °C
  float R[8]; //Vector of the eight resistors connected to the thermistors
  float B;    //Material Constant
  float Vdd;  //SUpply voltage for the circuitry of the thermistor
}T_formula_const;

typedef struct{
  float temp[8]; //Vector for the storage of the eight converted temperature values
  T_formula_const values;
}Temp_values;
```

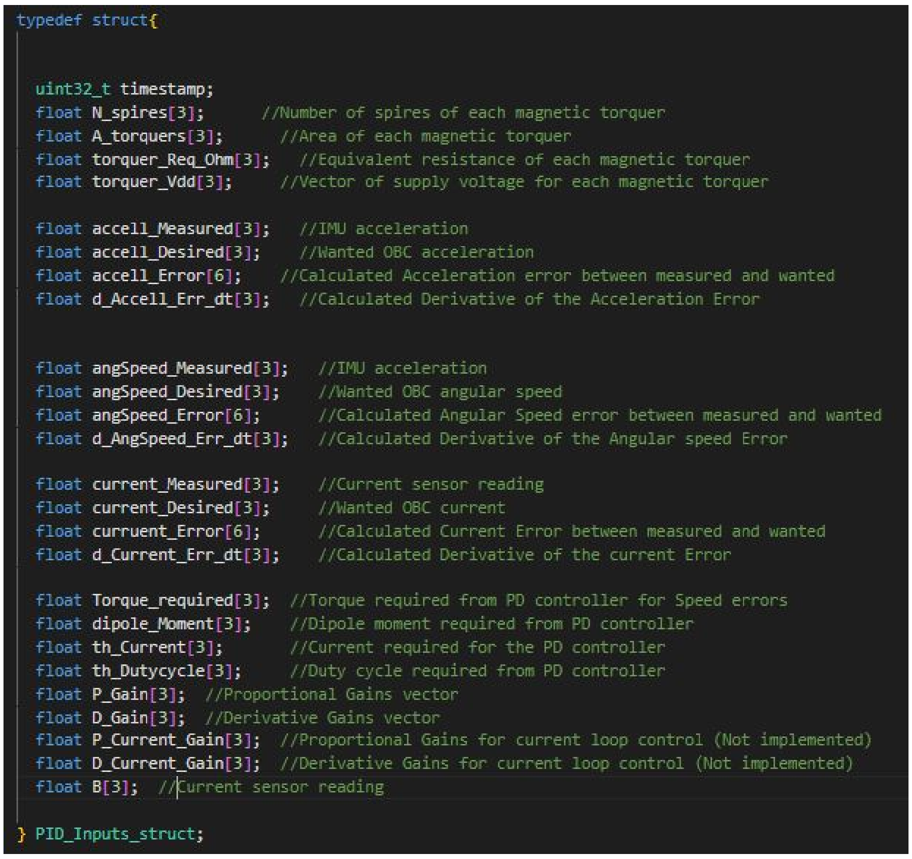
Figure 4.12: Handlers for the temperature sensors acquisition

- **void get__temperatures(SPI__HandleTypeDef *spi__struct,Temp__values *temp__struct, uint8__t counter)**: allows to obtain the temperatures of a selected NTC channel(parameter **counter**).
  It makes use of the functions **select__input**,**ADC__Conversion** and **voltage__to__temperature__con** previously described.

pid__conversions.h

Even if the variables suggests that a PID control was developed at the moment only a PD one was actually implemented:

- **void PID__error__calculation(PID__Inputs__struct *PID__Inputs)**: calculates the error between two attitude measurements. It has as input a structure for the management of PID control and below the handler is showed:



```c
typedef struct{


  uint32_t timestamp;
  float N_spires[3];      //Number of spires of each magnetic torquer
  float A_torquers[3];      //Area of each magnetic torquer
  float torquer_Req_Ohm[3];   //Equivalent resistance of each magnetic torquer
  float torquer_Vdd[3];     //Vector of supply voltage for each magnetic torquer

  float accell_Measured[3];   //IMU acceleration
  float accell_Desired[3];    //Wanted OBC acceleration
  float accell_Error[6];    //Calculated Acceleration error between measured and wanted
  float d_Accell_Err_dt[3];   //Calculated Derivative of the Acceleration Error


  float angSpeed_Measured[3];   //IMU acceleration
  float angSpeed_Desired[3];    //Wanted OBC angular speed
  float angSpeed_Error[6];      //Calculated Angular Speed error between measured and wanted
  float d_AngSpeed_Err_dt[3];   //Calculated Derivative of the Angular speed Error

  float current_Measured[3];    //Current sensor reading
  float current_Desired[3];     //Wanted OBC current
  float curruent_Error[6];      //Calculated Current Error between measured and wanted
  float d_Current_Err_dt[3];    //Calculated Derivative of the current Error

  float Torque_required[3];  //Torque required from PD controller for Speed errors
  float dipole_Moment[3];    //Dipole moment required from PD controller
  float th_Current[3];       //Current required for the PD controller
  float th_Dutycycle[3];     //Duty cycle required from PD controller
  float P_Gain[3];  //Proportional Gains vector
  float D_Gain[3];  //Derivative Gains vector
  float P_Current_Gain[3];  //Proportional Gains for current loop control (Not implemented)
  float D_Current_Gain[3];  //Derivative Gains for current loop control (Not implemented)
  float B[3];  //Current sensor reading

} PID_Inputs_struct;
```

Figure 4.13: Handlers for PD control alghoritm

- **void PID_attitude_Derivative_calculation(PID_Inputs_struct *PID_Inputs)**: calculates the derivative of the error to give as input to the PD controller
- **void PID_error_2_torque(PID_Inputs_struct *PID_INPUTS)**: converts the error calcuated in **PID_error_calculation** into a torque value;
- **void PID_torque_2_dipole(PID_Inputs_struct *PID_Inputs)**: converts the torque calculated in **PID_error_2_torque** into a magnetic dipole value;
- **void PID_dipole_2_current(PID_Inputs_struct *PID_INPUTS)**:converts the magnetic dipole calculated in **PID_torque_2_dipole** into a current value;
- **void PID_current_2_DutyCycle(PID_Inputs_struct *PID_Inputs)**:converts the current value calculated in **PID_dipole_2_current** into a Duty Cycle value;

For further details about the control law implemented see 2.3.

queue_structs.h

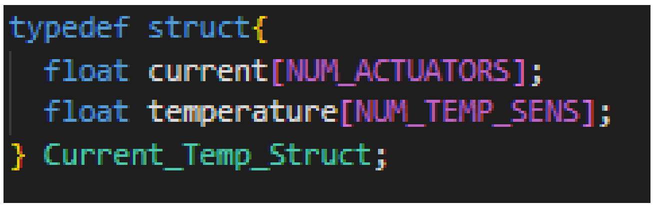
In the following list will be described the functions used two manage data arriving from queues inside tasks.

- **typedef void (*CombinedDataProcessor)(void *event,void *data1)**: this is a special function that allows to handle whatever type of struct object: as first and second parameters accepts two struct pointer of void type and as third parameter a function that processes the data contained on the two structs objects. The third function is customized depending on the case;
- **void processCombinedData(void *event,void *strct1, CombinedDataProcessor processor)**: this function allows to handle pointers to void(first two parameters) and as third one a pointer to void struct function;
- **void receive_OpModequeue_control(void *event,void *opmode)**: this function allows to store in a local variable the OpMode received from the OBC;
- **void receive_IMUqueue_control(void *event,void *PID_struct)**: this function allows to store in a local struct, dedicated to the PID control, the IMU data recevied via a Queue dedicated to the IMU;
- **void receive_IMUqueue_OBC(void *event,void *attitude)**: this function allows to store in a local struct,dedicated to the attitude received from the OBC, the IMU data received via a Queue dedicated to the IMU;
- **void receive_Current_Tempqueue_OBC(void *event,void *current_temp_struct)**: this function allows to store in a local struct all the data related to A/D conversions on the ADC, so the temperature values and the Current values data received via a Queue dedicated to the Houseekeeping(**HouseekeepingADCS**) data of the ADCS;
- **void receive_Attitudequeue_control(void *event,void *attitude_adcs)**:allows to store in a local struct all the data related to the target Attitude received via a Queue containing the Attitude data(**setAttitudeADCS**) sent by the OBC board to the ADCS;

ADCS Tasks Data Exchange

Here a brief explenation of the code relative to the freertos.c library will be given.
To synchronize the Tasks the Queue mechanism was used due to the architecture at system level that required continuous exchange of data starting from the GS arriving to the ADCS.

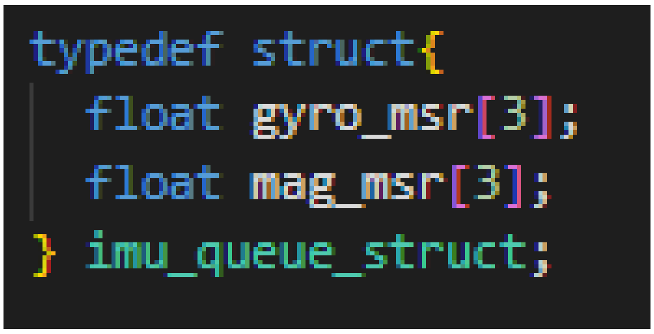
- **ADCSHouseKeepingQueue** : this is a Queue used to send from the Check/Safet Task to the OBC Task the Local HouseKeeping data. Before beeing sent the Queue is filled with a structure pointer of type **Current_Temp_Struct** organized like this:



```
typedef struct{
  float current[NUM_ACTUATORS];
  float temperature[NUM_TEMP_SENS];
} Current_Temp_Struct;
```

Figure 4.14: Currents and Temperatures struct

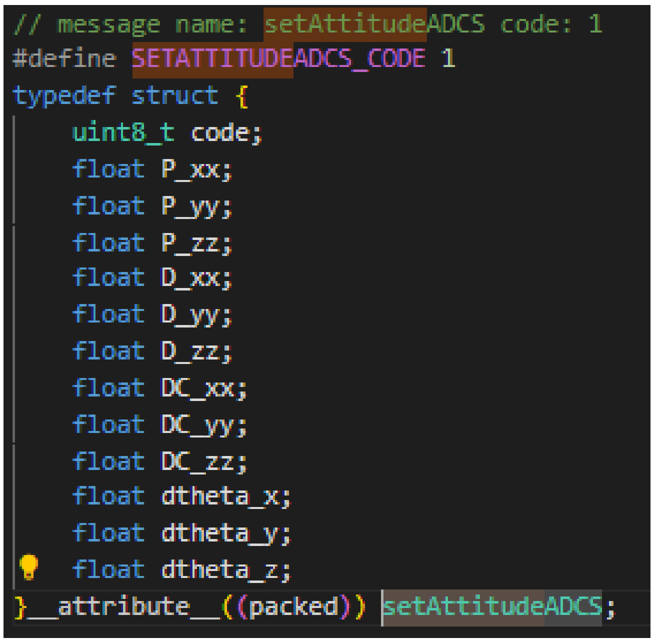
- **IMUQueue1** and **IMUQueue2** : Once the IMU Task reads the values of Angular Speed and Magnetic field it samples them with a struct of type **imu_queue_struct**(4.5) and fills the Queue which is sent to the Control Task and the OBC Task: the first needs them to have an update on the state of the System and close the control Loop,the second to send them back to the GS which shows them in the Dashboard on Grafana.



```
typedef struct{
  float gyro_msr[3];
  float mag_msr[3];
} imu_queue_struct;
```

Figure 4.15: IMU data struct

- **setAttitudeADCSQueue** : once the OBC receives the new target Attitude Data from the GS it samples the data in a local struct of type **setAttitudeADCS**(4.16), declared in the library message.h because these data comes after the change of OpMode to 1 on the part of the GS, and puts them in the Queue ready to be sent to the Control Task;



```
// message name: setAttitudeADCS code: 1
#define SETATTITUDEADCS_CODE 1
typedef struct {
    uint8_t code;
    float P_xx;
    float P_yy;
    float P_zz;
    float D_xx;
    float D_yy;
    float D_zz;
    float DC_xx;
    float DC_yy;
    float DC_zz;
    float dtheta_x;
    float dtheta_y;
    float dtheta_z;
}__attribute__((packed)) setAttitudeADCS;
```

Figure 4.16: Struct in message.h library to manage received target Attitude Data from OBC

- **setOpModeADCSQueue** : this queue is sent by the OBC Task to communicate to the Control Task chenges on the OpMode,once it is received succesfully by the OBC Task;
- **intOpModeADCSQueue** : this queue has the only purpose of sending back to the OBC the updated OpMode data so that it can be updated on the Dashboard of Graphana.

In figure 4.17 there's a scheme who summarizes the Queue exchange between tasks.

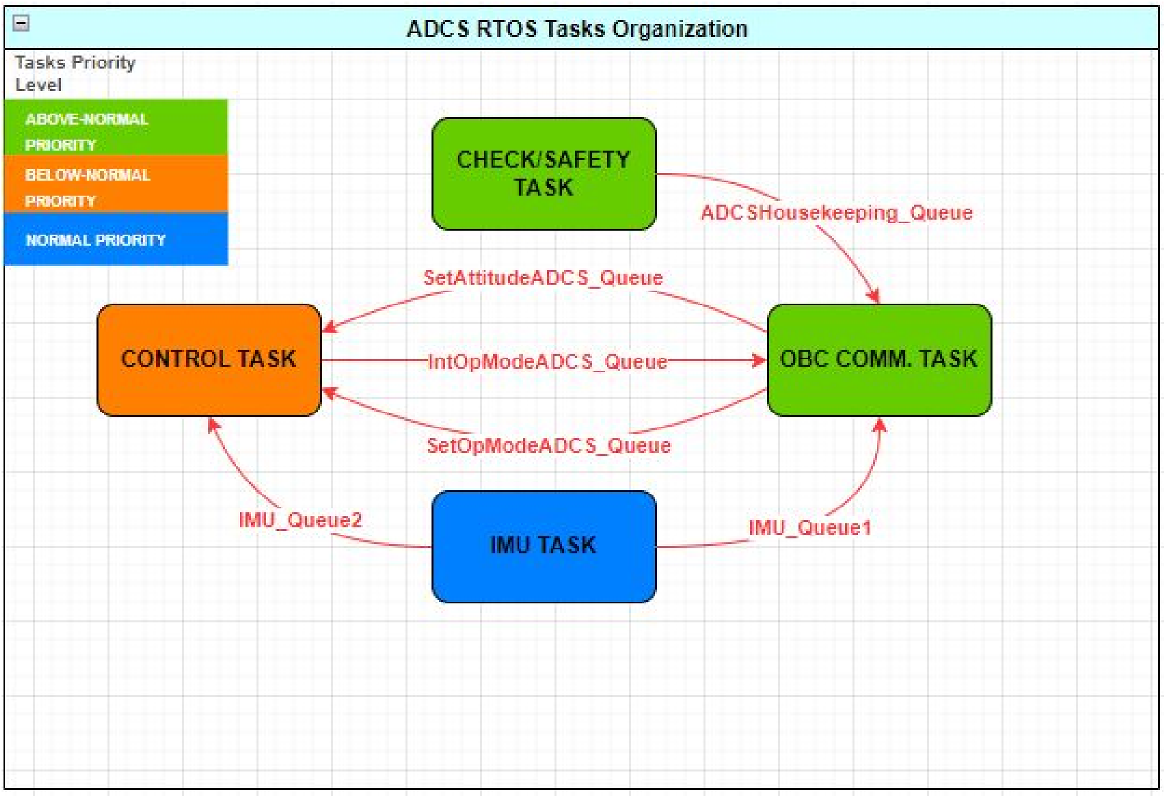


Figure 4.17: ADCS RTOS Tasks Organization

Chapter 5

Test Campaign

This chapter will introduce the facility present in Forlì laboratory [27],called CubeDynA (CubeSat Dynamic Attitude simulator testbed) Facility , by describing the included components,experiments and final results.

5.1 Facility Description

[28],The increasing number of launches of nanosatellites and cubesats over the years lead to an increasing needing in performing quality test to assure the reliability of all the sub-systems and mechanical structure. Looking at this actual scenario, developing appropiate test facility becomes of remarkable importance.

Due to these reasons dynamic simulation facilities have been built to emulate the space conditions and in particular the microgravity. This last topic becomes extremely important considering that tests must be performed in the earth with completely different conditions compared to the space. It becomes essential to reproduce in the best way possible those harsh conditions. The parameter that must be minimized is the **Disturbance Torque** of the facility where the tests are performed: this torque is a consequence of the different methodologies used to recreate microgravity.
To minimize it the mini-satellite must be the largest possible.

Of all the methods implemented to recreate microgravity on earth the Facility of Forlì applies the **Air-Bearing**: a pressurized air flow is used to generate a film between two corresponding surfaces. This film will act as a lubricant between the surfaces,reducing friction.
In Figure 5.1 the CubeDynA Facility is showed. To simulate the Attitude Dynamics Environment encountered by a satellite in Low Earth Orbit (LEO) the Facility embeds mainly 4 components:

- 1) A **Triaxial Helmholtz Cage** to simulate the Earth's magnetic field;
- 2) A **Sun simulator**;
- 3) An **Air Bearing Table** (**ABT**) equipped with an **Automatic Balancing System** (**ABS**) that provide three-degree-of-freedom rotational motion with low disturbance;
- 4) A metrology system for generating **Ground-Truth Attitude Data**.

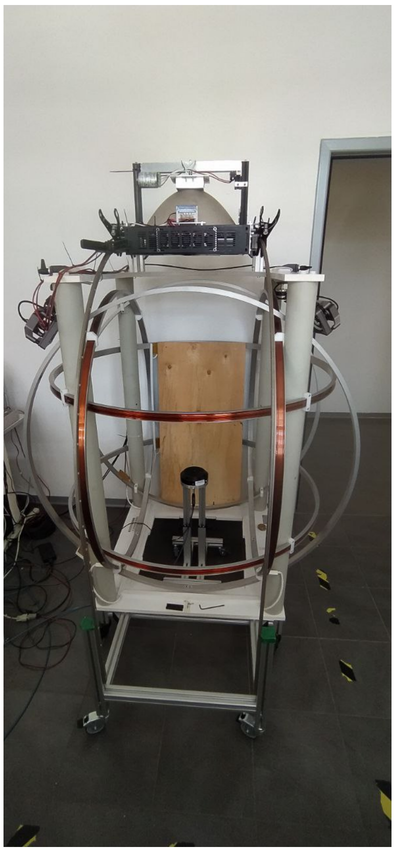


Figure 5.1: CubeDynA Facility in Forlì

5.1.1 Helmotz Cage

A magnetic field can be generated to test the Attitude Determination and Control System of a cubesat or to simulate the magnetic field generated by the earth experienced byt the satellite once it is in orbit.
One of the possible methods to reproduce these conditions is the Helmotz Cage, a device capable of generating **Uniform** Magnetic Fields in all the 3 directions.
To reproduce the field in the three axis the so called **Helmotz coils** are used: they consist of a pair of coils parallel to each other,each single coil has N turns of wires. If a current is lead passing through one of the coils a magnetic fields is created. It's expression is:

$$B = \frac{\mu_0 \cdot i \cdot N \cdot R^2}{2 \cdot (x^2 + R^2)^{\frac{3}{2}}} \tag{5.1}$$

The Helmholtz cage in the Facility is composed of some hardware components:

- 3 orthogonal pairs of coils with 1300 mm diameter, which correspond to the x, y and z axes of the reference system, respectively;
- A fluxgate magnetometer;
- A Programmable power supply;
- An Arduino Uno for closed-loop control

Thanks to these component the Cage is able to generate fields of arbitrary direction in the range +/-10G(1 Gauss = 10^{-4}) In the center of the HC it's installed a cilindrical support, which has the function to keep the base platform of the air bearing at the center of the cage, and so at the center of the generated magnetic field.

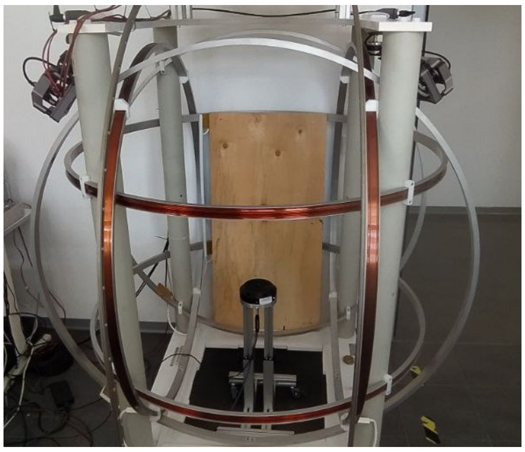


Figure 5.2: Facility Helmotz Cage

5.1.2 Sun Simulator

As the name suggests a Sun Simulator is a system able to reproduce in the best way possible the solar radiation characterising the satellite's orbit.
It serves essentially two scopes:

- Use it to implemet an attitude determination: for Cubesats attitude determination systems that exploit solar radiation are often employed;
- Allow the simulation of disturbance torque due to solar radiation: this torque must be compensable by the CubeSats itself.

The simulator consisting of a **COST LED Studio light**, with a 300W phosphor-coated LED as light source.
The collimation lens is a 400mm diameter Fresnel. The purpose of the simulator is to collimate the light of the lamp to replicate sunlight.
Through a focus test, the distance between the Fresnel lens and the light source has been tuned to maximise beam collimation.

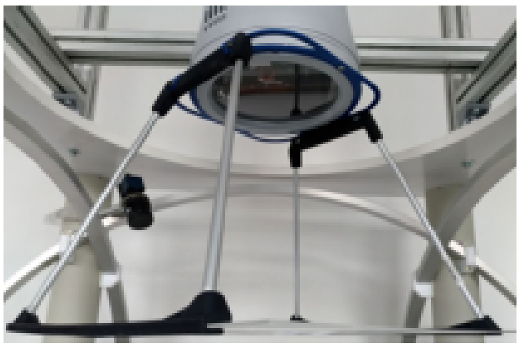


Figure 5.3: Facility Sun Simulator

As explained in section 3.7.1 the ADCS board has the arrangment to use the Sun Sensor but it's circuitry wasn't designed,in fact for this Test Champaign it is not provided to test this part of the ADCS.

5.1.3 Air Bearing Table

The Air Bearing system in figure 5.4 mentioned before, consists of two parts: the base which is fixed at the pedestal, and the hemispherical one which is movable.
Between the two a thin layer of air is generated to minimize friction. Ideally, the disturbance torques affecting the platform should be kept in the same order of magnitude as those in orbit: in case of LEO orbit, at 700 km altitude, with a 1U satellite, these orbital disturbance torques are in the order of $10^{-6} Nm$.
The disturbance torques affecting an air-bearing platform can have different causes but in this specific case, the torque due to gravity is the main torque acting on the platform,resulting from misalignment between the CM and CR.

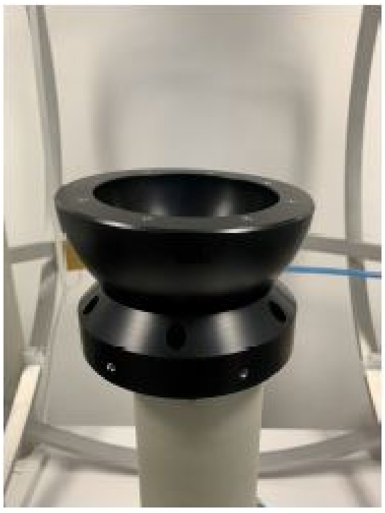


Figure 5.4: Facility Air Bearing Pedestal

The correction of this misalignment is performed by adjusting the relative position of the CR and CM through a balancing process.
Two types of balancing must be applied:

- **Coarse balancing** : is accomplished by evenly distributing the mass of platform components and by manual placement of counterweights on the rotating platform;
- **Fine balancing** : is achieved using a set of three mutually orthogonal sliding masses moved by stepper motors thanks to a two-steps procedure.
  First, the in-plane masses positions are adjusted through a PID feedback control; afterwards, the out-of-plane mass displacement is computed through a system identification algorithm by sampling the ABT free oscillations, as explained in details in [29].

In total two ABTs have been developed, capable of hosting payloads of different mass ranges up to 4 kg.

Mechanical interface with the ADCS

The CubeDynA platform allows hosting satellites of the CubeSat form factor with mechanical contact only with the Cubesat's rails. In the case of the CuebeSat Test Platform, the mechanical interface happens with lower part of the platform, while the top part is left open for an easy access to the electronic board as shown in Figure 5.5.

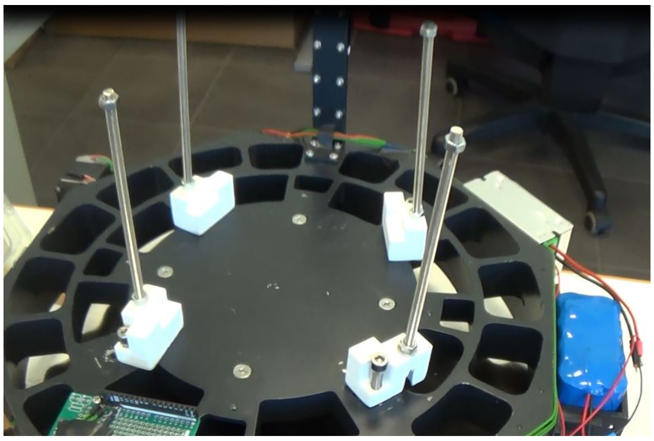


Figure 5.5: Mechanical interface with the CubeSat

5.2 Integration and Verification

The Champaign consisted in three days of work where the objective where:

- Succesfully perform the **Balancing procedure**;
- Perform the Tests: in particular the Tests to perform had the target of:
  - 1) Verifing the actual **polarization** of the magnetorquers activating the HC magnetic fields in all of the three directions separately and so, testing the magnetorquer positioned in that direction independently of after the other;
  - 2) Testing the Control Law to implement the **Dethumbling maneuvre**.

5.2.1 Fully Assembled Cubesat

The structure that supports the OBC and ADCS boards together with the battery is made with additive manufacturing techniques in PLA, a material that provides sufficient structural strength with low density, resulting in the minimum structural mass. The total mass of the totally assembled system results in 430 g.

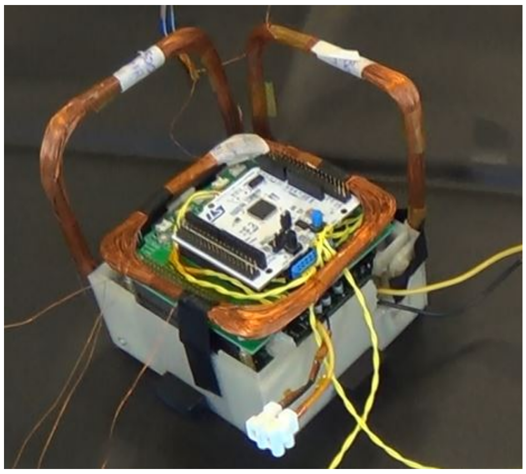


Figure 5.6: The Cubesat Test Platform

5.2.2 Preliminary Setup

The first day the work was the following:
in order to put the CubeSat under test on the ABT some prelimiary checks should have been taken, considering that we brought with us all the necessary material,so te risk was that some hardware connection were weak or other problems could occur.

The same passages explained in the following where done in laboratory at the STAR Lab so they were tried and tested before.
To prevent this we followed a precise outline:

- 1) **Setup Raspberry and Ground Station**: it was important to configure again the Raspberry to connect to a new IP addres since the WIFI port was different with respect to the Star Lab at Politecnico of Turin. Once done this we had to check that the Raspberry could launch the InfulxDB and Telegraf services with no problem and that it was able to send data to the Database to be than visualized in Graphana, meaning the Ground station was working properly.

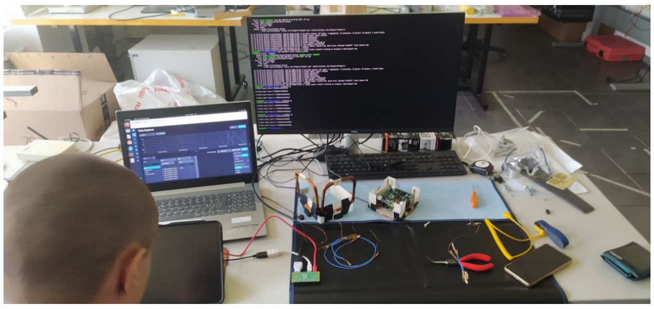


Figure 5.7: Setup Raspberry and Ground Station in the Laboratory

- 2) **Hardware Check**: after the Rasp and GS worked properly we had to connect together the two boards. In this phase we checked that every connection was ok,specifically the lines for the UART communication and for the power supply.
  During the power supply check we had problems with the battery connection and we had to repair it as showed below;

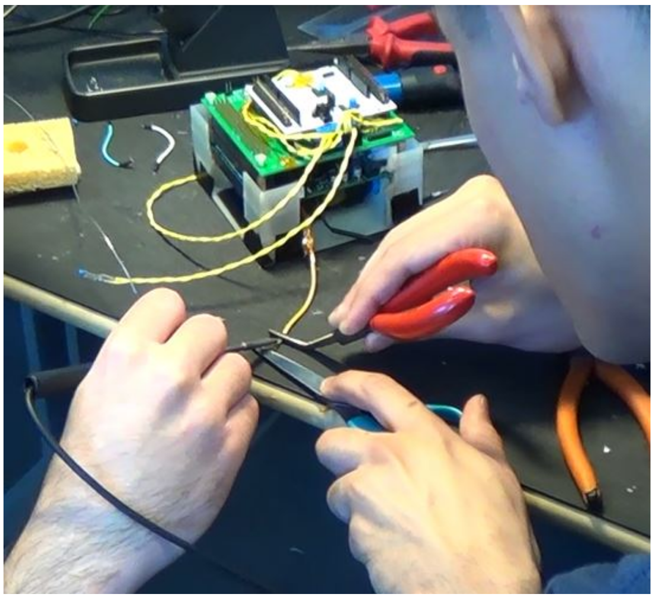


Figure 5.8: Battery connection repair

- 3) **Firmware Check**: once the connections were ok we had to upload the latest version of the Firmware and assure it was correctly delivered to the board.

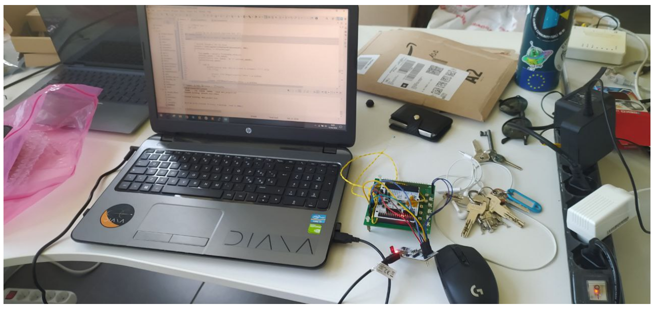


Figure 5.9: Firmware upload

Finally the integration between the ATBand the Cubesats was done and we were ready to perform the Balancing Procedure. In figure 5.10 is reported the integrated ATB + Cubesat structure.

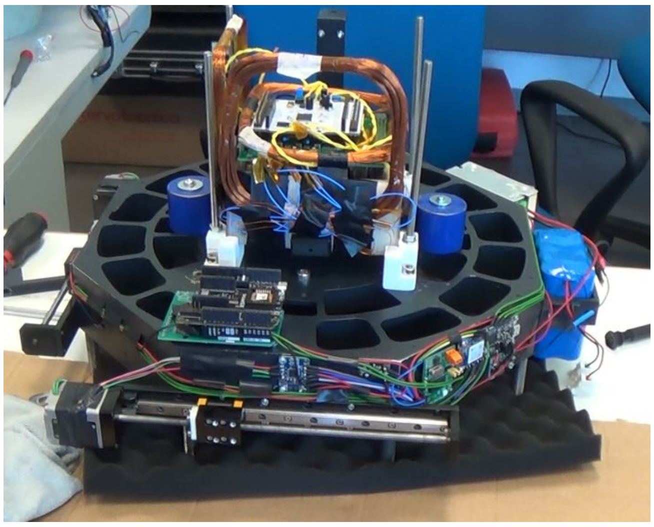


Figure 5.10: Integrated platform ready for the Balancing procedure

5.3 Balancing Procedure

As just mentioned the Balancing procedure consisted of two phases.

5.3.1 Coarse Balancing

Coarse balancing is needed to compensate for the macroscopic offset between the CM and the CR of the platform. In this particular case, coarse balancing required distributing 0.8 kg of counterweights(fig 5.11) on the platform. In figure 5.12 the moment where coarse balancing was performed application is reported.

Overall, the total weight of the platform resulted to be equal to 7.404 kg.

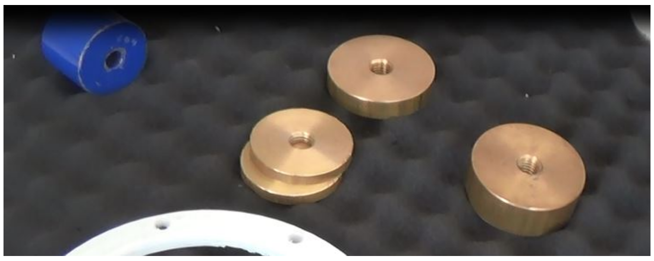


Figure 5.11: Some counterweights used for the Coarse Balancing

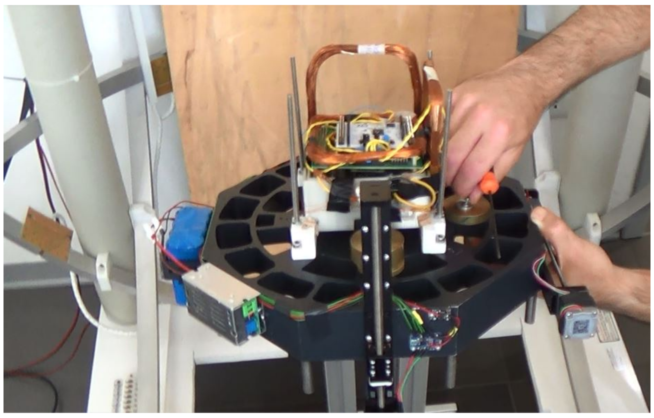


Figure 5.12: Coarse Balancing procedure

5.3.2 Fine Balancing

Fine balancing consists of two steps:

- **In-plane balancing** :where the horizontal component of the CM to CR offset is compensated;
- **Vertical offset estimation and compensation**.

The two steps are iterated until the offset estimated value does not decrease further.

Figure 5.13 shows the normalized horizontal components of the gravitational acceleration, horizontal components of angular velocity and stepper motors' carrier position during the in-plane balancing process. The in-plane balancing process is considered completed when the horizontal components of the gravitational converge to zero.

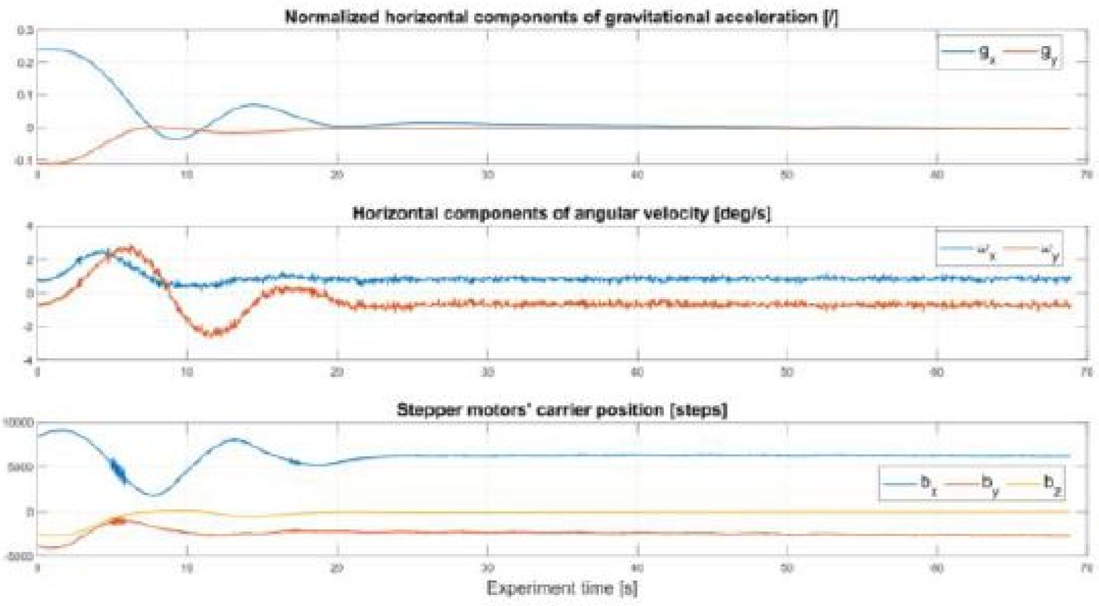


Figure 5.13: Normalized horizontal components of the gravitational acceleration, horizontal components of angular velocity and stepper motors' carrier position during the in-plane balancing process.

Figure 5.14 shows the residual torque estimation and the CM to CR offset estimation. The residual torque RMS results to be equal to $6 \times 10^{-5} Nm$ and the peak value $1.5 \times 10^{-4} Nm$.

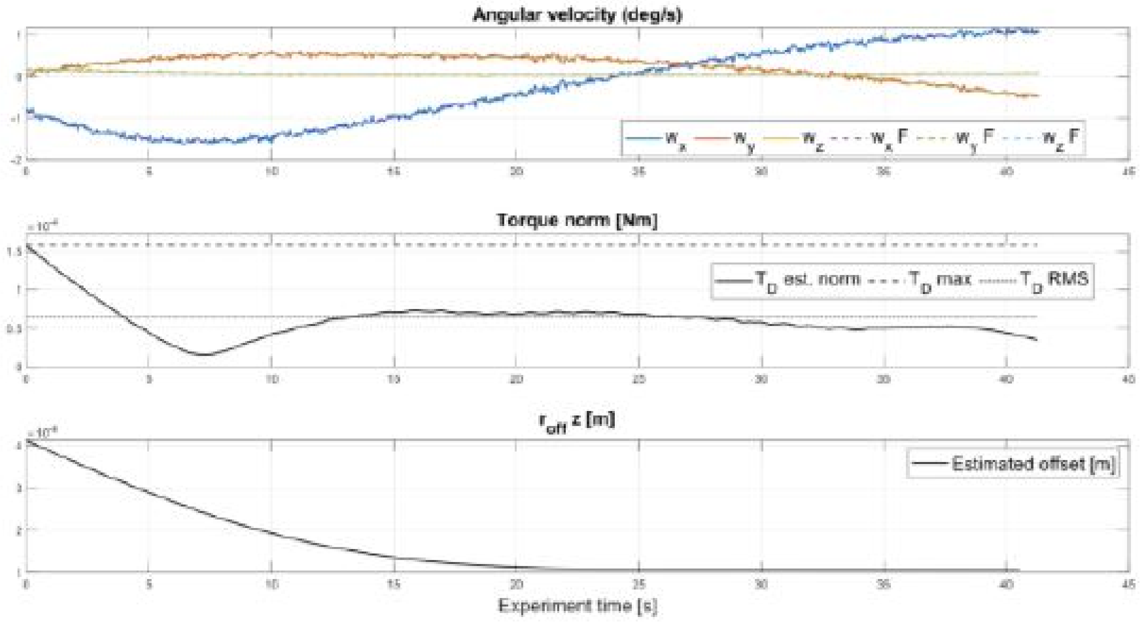


Figure 5.14: Angular speed, residual torque estimation and offset estimation

5.4 Final Tests

In figure 5.15 the Cubesat ready to be tested was integrated in the Facility structure.

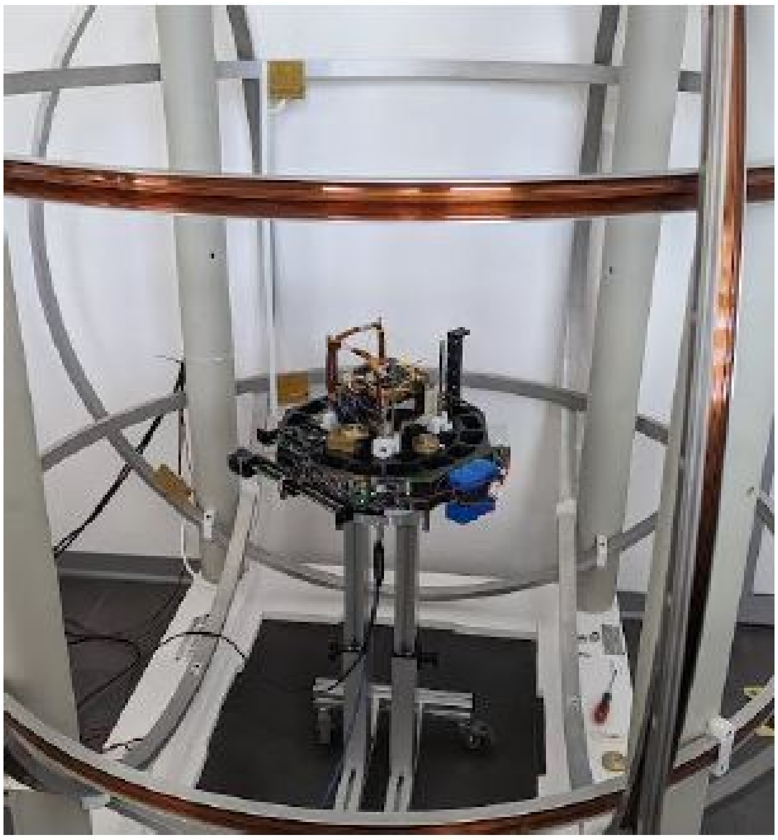


Figure 5.15: Cubesat succesfully integrated in the Facility structure

As previously explained,after the Balancing was completed it was time to perform the Tests. The first one is the **Polarity Test** : when the Magnetorquers were attached to the ADCS board we didn't check for the correct direction of the current so this test is a good way to control if the Magnetorqers where mounted on the contrary with the help of the HG external magnetic field. At the end the measurements in terms of Magnetic field and angualr velocity were validated,comparing them to the ones of the IMU which is an element of the ground-truth instrumentation present in the Facility.
Before the Test the IMU of the Facility were already calibrated

5.4.1 IMU Characterization

The execution of the test consisted of moving the CTP starting from a random initial attitude to an attitude for which one axis of the CTP is aligned with the direction of the magnetic field generated by the Helmholtz cage.

Due to the different sampling rate, a mask with values collected at the same time tag was matched. The difference (Δw) between the reference values and the IMU measurement are shown in figure 5.16. Red, blue, and green bars report the difference of measurements of angular velocities along x, y and z respectively. The mean error was [-0.00273, -0.0021, 0.00466] rad/s, with standard deviation of [0.0155, 0.0412, 0.0231] rad/s and a distribution resembling a Gaussian one, so we can consider the selected IMU accurate enough for the future application.

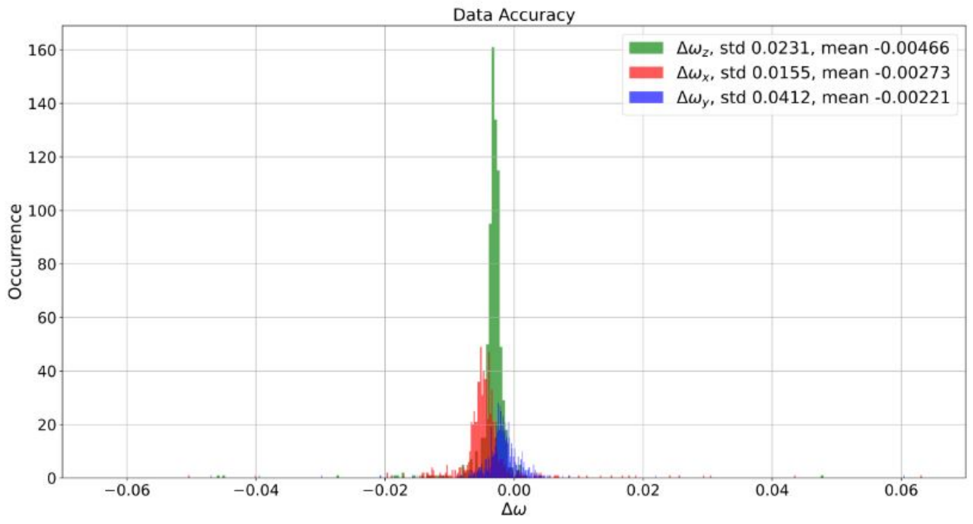


Figure 5.16: Mean and Standard Deviation of the error on the angular rate measurements.

5.4.2 Polarity Test

As described in a previous section, at 4.3.1, it was introduced a further OpMode to allows the testing of magnetorquers singularly. In fact during these Tests the Opmode 2 was sent by the Ground Station after an initial ideal state at Opmode 0.

The test consisted in checking that the dipole commanded to the magnetorquers is actuated in the correct direction: using the Helmholtz cage, a magnetic field in a direction perpendicular to the local vertical is created. If the magnetic dipole is actuated in the correct direction, it is expected that the CubeSat aligns the positive dipole direction to the magnetic field lines.

This test foresaw that the currents flowed in the actuators first in clockwise and then in anticlockwise directions. CTP started to rotate from a random attitude and a duty cycle of 90% is applied via PWM driver and circuits up to the alignment with the Magnetic Field direction. Then, a duty cycle of –90% is applied and CTP started rotating in the opposite direction.

Magnetorquer on X axis

Figure 5.17 shows the results of actuation of the magnetic torquer mounted on the positive x(IMU reference frame) face of the CubeSat. The top graph shows the evolution of the angular velocity. It is possible to see that there is an increment of the velocity then a slowing down (with an overshoot) followed by a stable attitude condition (no rotation). Then, the inversion of polarisation is applied, and a similar trend, but with opposite sign, is observed with a lower overshoot up to reach the stable attitude.

The graph in figure 5.18 instead shows the current flowing inside the magnetic torquers, which exhibits a non-negligible variation of 50 mA of amplitude.
However it was confirmed that the current that flows inside the magnetorquers was constant and the significant variation of the measurements was due to the acquisition circuit, requiring an review of the latter.

On the contrary, no ripple on the measured values was observed, confirming that the filtering stage worked properly.

These results suggest that the magnetic actuators embedded in the CubeSat platform mock-up can surely be exploited for future Attitude and Angular Velocity control testing.

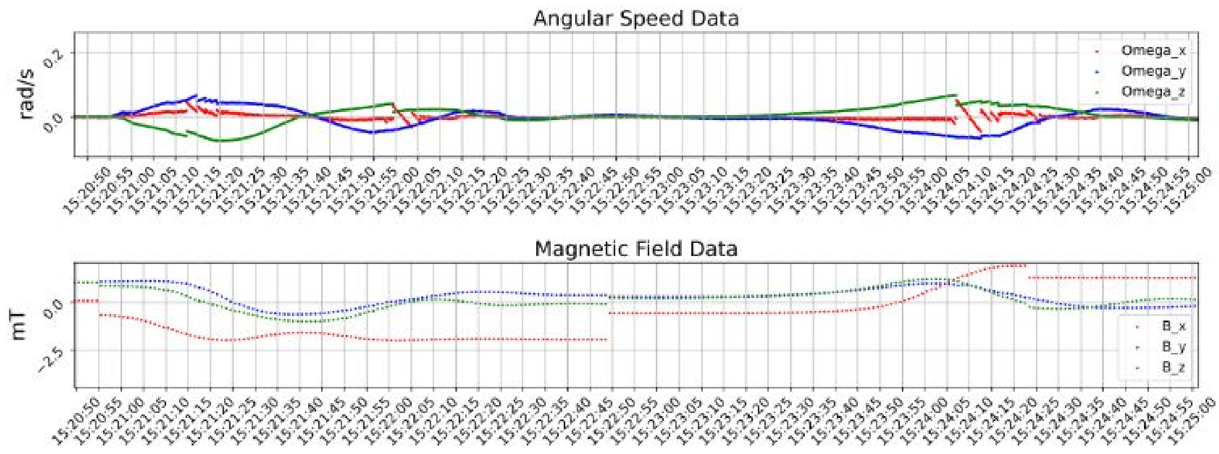


Figure 5.17: Angular Speed and Magnetic Field measurement when Magnetorquer on X axis is activated

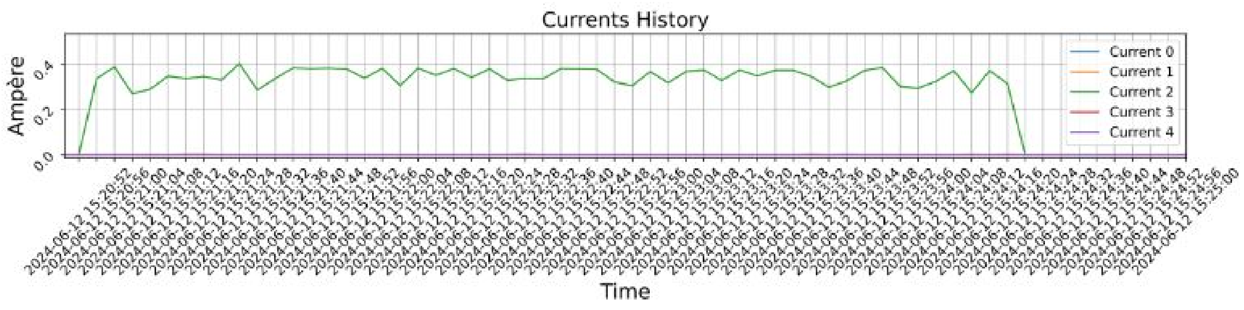


Figure 5.18: Current flowing in X Magnetorquer

Magnetorquer on Y axis

The same experiment was performed for the Magnetorquer placed in the positive y axis(IMU reference frame) allowing to conclude that even here the current exhibited a non-negligible ripple due to the circuitry behind.

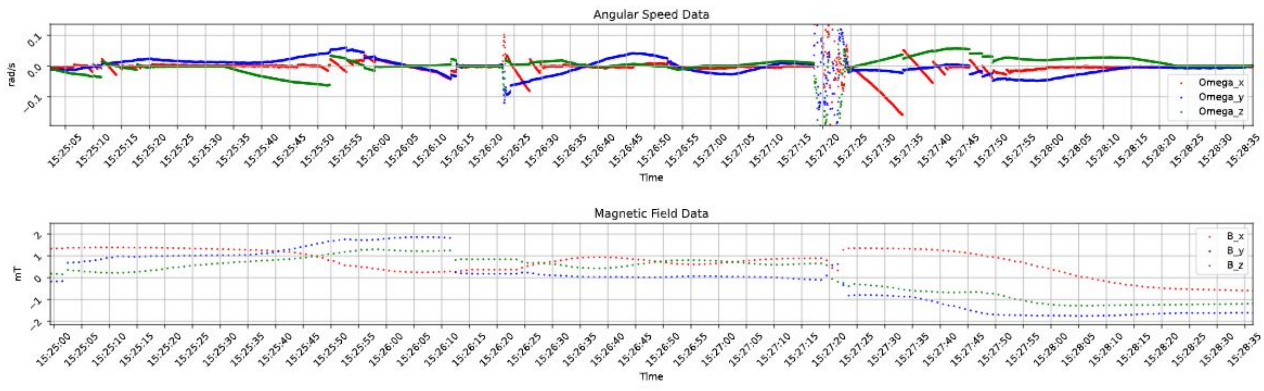


Figure 5.19: Angular Speed and Magnetic Field measurement when Magnetorquer on Y axis is activated

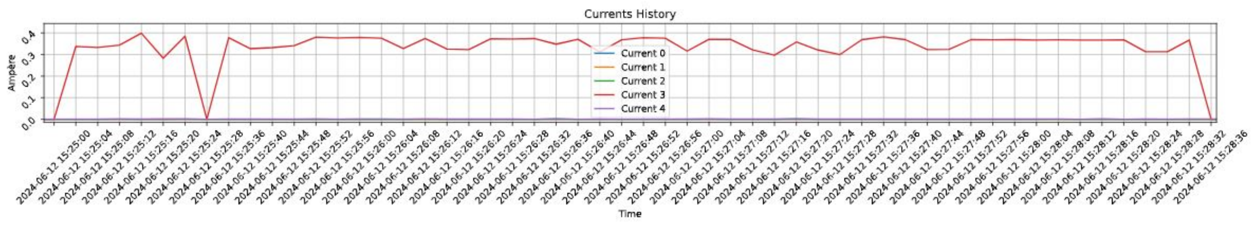


Figure 5.20: Current flowing in Y Magnetorquer

Below is showed the Facility frame reference to be compared with the reference of the IMU in figure 3.19:

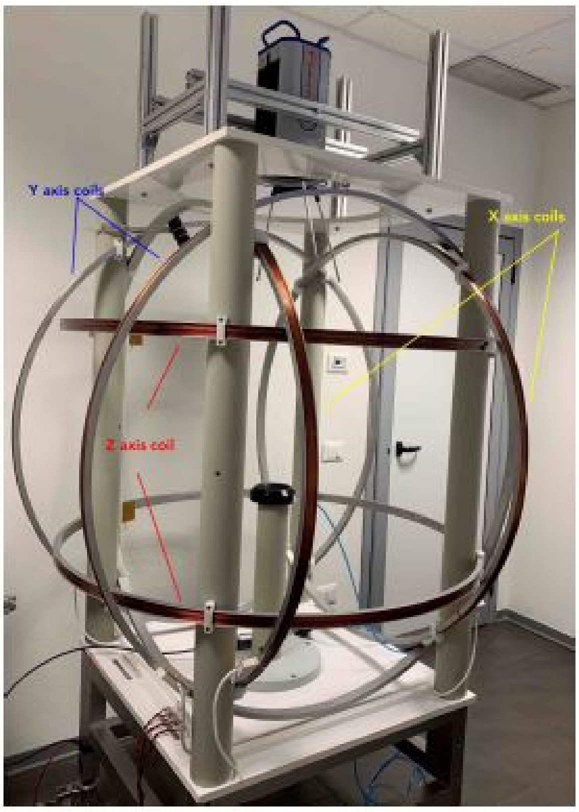


Figure 5.21: Facility frame reference

As expected it was not possible to perform the same polarization experiment on the z axis since as explained in the section on Magnetorquers(3.5) its Magnetorquer had not sufficient turns(210) to generate a significant mechanical torquer able to make the system rotate on the z axis when the HC generated a magnetic field in the z axis,even having a Duty Cycle of 90% on that Magnetorquer's electronic driver. However replacing it with a proper actuator it will be possible to perform all the tests as for x and y axis.

Chapter 6

Conclusions

This thesis work had as its primary objective to review an old ADCS board project developed some years ago to provide the first version of an ADCS board that will be integrated into a more complex 3U structure Cubesat to launch 3 years later in space.
The work carried out in this thesis reached the goal of designing, testing, and validating a simple ADCS board integrated into a CubeSat of reduced dimension concerning the standards.

The system successfully demonstrated to be able to communicate at different levels(Ground Station to OBC and OBC board to ADCS board) and to drive correctly hands-made Magnetorquers, producing a good torque to provide rotation considering that this is a system completely developed by few students of Politecnico of Turin.
The integration of the CubeSat system in the CubeDynA Facility at the Forlì Campus of the University of Bologna was successfully completed, leaving room for optimism in future test sessions.
Some mistakes were made in the process that inevitably stole time for improvements and in particular this was crucial for the Test Campaign since it wasn't possible to test the Control of the Attitude. However, the results collected demonstrated that future studies will surely allow us to obtain a good control strategy regarding the Dethumbling maneuver and more in general to a complete attenuation of the Cubesat integrating even the Reaction Wheels.

In the process of the project development a previous thesis work [11], where a PCB with the name of Singer was developed for the project of **SPEA Satellites**, was taken as reference especially for the Hardware design.
However, this work has to be intended as a starting point for a future 2.0 version of the ADCS board, in fact during the thesis work the author identified some aspects that could be improved or added for the next version of the ADCS board:

Hardware improvements

- **Complete removal of the Nucleo Board**: the Nucleo Board attached to the ADCS one was an easy and fast solution to provide a microcontroller unit to the board. However, this steals space in vertical for other boards especially if the ADCS board must be integrated into a bigger Cubesat. For this reason, it was thought to embed directly in the ADCS board the MCU of the Nucleo Board with all the necessary electronic components plus the electronics for the ST-Link to reduce vertical clutter and allow easier access to the USB port for programming and Debug;
- **On-site IMU calibration**: some solutions were taught to allow performing IMU calibration directly on board before every Test session to reduce the risk of noisy measurement;

- **Sun sensor and Reaction Wheel integration**: as widely explained the ADCS board has the arrangement for collecting data from a Sun sensor. However, to provide its measurement another PCB should have been developed entirely dedicated to the conditioning of the sensor signal and transmission of the data to the ADCS board. As previously said this can be done using a simple microcontroller like a PIC from Microchip or other similar microcontroller.
  Moreover Reaction Wheels must be integrated too to provide propulsion to the Cubesat and it will be important to understand their characteristics and adapt eventually the new ADCS board to them;

- **OBC Board improvements**: the OBC board had two main serious problems: the circuitry for A/D conversion of the EPS module voltages and currents couldn't work, moreover the connector for the battery gave a lot of problems, especially during the Final Tests. So, surely, these are the main problems to fix regarding the board.

Firmware improvements

- **Improvement of the code and specifically of the Control Law Library**: to develop and improve a good control strategy the code of the library for the PD control implementation could be improved as well as the code for the safety state of the ADCS board;

- **EEPROM usage**: it is possible to make use of the Flash memory of the Nucleo board to emulate an EEPROM behavior. This could be done in the optic of collecting and saving data directly on the Nucleo for further post-processing actions after every Test session, reducing eventually the overhead of data to send to the OBC board.

Technical documents

[1] Alicia Johnstone. *CubeSat Design Specification,(1U - 12U).* Feb. 2022. URL: https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/62193b7fc9e72e0053f00910/1645820809779/CDS+REV14_1+2022-02-09.pdf (cit. on p. 1).

[2] Nanosats Database. *CubeSat constellations, companies, technologies, missions and more.* URL: https://www.nanosats.eu/ (cit. on p. 2).

[5] PC/104 Embedded Consortium. *PCI-104 Specification Version 1.0.* Nov. 2003. URL: https://resources.winsystems.com/specs/PCI-104Spec_v1_0.pdf (cit. on p. 5).

[8] STMicroelectronics. *STM32L452xx.* Nov. 2020. URL: https://www.st.com/resource/en/datasheet/stm32l452cc.pdf (cit. on pp. 15, 60).

[9] Analog Devices. *Low Power, 16-/24-Bit,Sigma-Delta ADCs.* URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ad7788_7789.pdf (cit. on p. 16).

[10] Texas Instruments. *SN74HC4851 8-Channel Analog Multiplexer/Demultiplexer With Injection-Current Effect Control.* Sept. 2003. URL: https://www.ti.com/lit/ds/symlink/sn74hc4851.pdf?ts=1719177074357&ref_url=https%253A%252F%252Fwww.google.com%252F (cit. on pp. 19, 46).

[12] Vishay. *NTC Thermistors, Radial Leaded, Accuracy Line.* URL: https://www.vishay.com/docs/29048/ntcle203.pdf (cit. on p. 19).

[13] Vishay. *NTC Thermistors Application Notes.* URL: https://www.vishay.com/docs/29053/ntcappnote.pdf (cit. on p. 19).

[14] Movella. *MTi-3.* URL: https://www.xsens.com/hubfs/Downloads/Leaflets/MTi-3.pdf (cit. on p. 21).

[17] Texas Instruments. *DRV8251A 4.1-A Brushed DC Motor Driver with Integrated Current Sense and Regulation.* Jan. 2022. URL: https://www.ti.com/lit/ds/symlink/drv8251a.pdf?ts=1712058941747 (cit. on pp. 28, 32, 50, 60).

[18] RS Pro. *Pacco batterie ricaricabile RS PRO.* URL: https://it.rs-online.com/web/p/pacchi-batterie-ricaricabili/1449413?gb=s (cit. on p. 42).

[19] Littlefuse. *467 Series 0603 Fast-Acting Fuse.* URL: https://www.mouser.it/datasheet/2/240/media-3320582.pdf (cit. on p. 43).

[20] Analog Devices. *Automotive 36V, 4A/5A/6A, Fully Integrated Synchronous Silent Switcher Buck Converters.* Apr. 2024. URL: https://www.mouser.it/datasheet/2/609/max20404_max20406-3128015.pdf (cit. on p. 43).

[21] STMicroelectronics. *500 mA low quiescent current and low noise voltage regulator.* Dec. 2019. URL: https://www.mouser.it/datasheet/2/389/ld39050-1849494.pdf (cit. on p. 43).

[22] Xsens. *Hardware Integration Manual.* URL: https://www.xsens.com/hubfs/Downloads/Manuals/Hardware_Integration_Manual_MTi_1-series.pdf (cit. on pp. 43, 51).

[23] IPC. *IPC-2221A Generic Standard onPrinted Board Design.* May 2003. URL: `https://www-eng.lbl.gov/~shuman/NEXT/CURRENT_DESIGN/TP/MATERIALS/IPC-2221A(L).pdf` (cit. on p. 48).

[24] AdvancedPCB. *Trace Width Calculator.* URL: `https://www.advancedpcb.com/en-us/tools/trace-width-calculator/` (cit. on p. 48).

[25] STMicroelectronics. *Integrated development environment for STM32 products.* URL: `https://www.st.com/resource/en/data_brief/stm32cubeide.pdf` (cit. on p. 52).

[26] FreeRTOS. *The FreeRTOS™ Reference Manual.* URL: `https://www.freertos.org/fr-content-src/uploads/2018/07/FreeRTOS_Reference_Manual_V10.0.0.pdf` (cit. on p. 52).

Bibliography

[3] Anton Rassõlkina, Toomas Vaimanna, Peeter Orgb, Alar Leibakc, Rauno Gordond, and Eiko Priidelb. «ADCS development for student CubeSat satellites – TalTech case study». In: (Aug. 2021) (cit. on p. 3).

[4] Foletti A. and Kaewkerd P. *SwissCube Phase A ADCS Report*. Technical Report 86-194. EFPL Lausanne, Mar. 2006 (cit. on p. 3).

[6] Wen-Chi Lu, Lili Duan2, and Yi-Xian Cai. «De-tumbling Control of a CubeSat». In: (2018) (cit. on p. 10).

[7] Jozef C. Van der Ha and Kikuko Miyata. «Attitude Control by Magnetic Torquer». In: (Jan. 2009) (cit. on p. 11).

[11] Simone Bollatino. «Development of a versatile on board computer for small satellites». MA thesis. Torino: Politecnico di Torino, Oct. 2023 (cit. on pp. 19, 20, 65, 85).

[15] Aziz EL FATIMI, Adnane ADDAIM, and Zouhair GUENNOUN. «Study and design of an active magnetorquer actuator model for nanosatellites». In: (2022) (cit. on p. 23).

[16] Jovanovic N., Riwanto B., Niemelä P., Mughal M. Rizwan, and Praks J. «Design of Magnetorquer-Based Attitude Control Subsystem for FORESAIL-1 Satellite». In: (Dec. 2021) (cit. on pp. 23, 26, 60).

[27] Dario Modenini, Anton Bahu, Giacomo Curzi, and Andrea Togni. «A Dynamic Testbed for Nanosatellites Attitude Verification». In: (Mar. 2020) (cit. on p. 71).

[28] Chiara Lughi. «Development and validation of a test bench for Attitude Determination and Control System for small satellite». MA thesis. Torino: Politecnico di Torino, Mar. 2023 (cit. on p. 71).

[29] Bahu A. and Modenini D. «Automatic mass balancing system for a dynamic CubeSat attitude simulator: development and experimental validation». In: (2020) (cit. on p. 74).