UNIVERSIDAD DE GRANADA

Facultad de Ciencias

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

TRABAJO FIN DE GRADO

Air Bearing Platform for Zero-Gravity Simulation with Reaction Wheels controlled by Espressif ESP32

Presentado por:

D./Dª. Francisco José Llave Iglesias

Tutor:

Prof. D. Andrés María Roldán Aranda

Curso académico 2020/2021

UNIVERSIDAD
DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

Air Bearing Platform Simulation for Reaction Wheels controlled by Espressif ESP32

Autor: Francisco José Llave Iglesias

Director: Andrés María Roldán Aranda

Departamento: Electrónica y Tecnología de Computadores

Palabras clave: CubeSat, Altium Designer, SolidWorks, Electronics, PCB Design, I2C, Inertial Disk, Reaction Wheel

Resumen: El proyecto en cuestión tiene como objetivos principales el desarrollo de una plataforma Air Bearing para la simulación de un ambiente en gravedad-cero, así como el uso de discos de inercia que ayudarán en su control de giro y estabilización. Para el control del sistema y la comunicación con el usuario se empleará un microcontrolador de Espressif, el microchip ESP32, el cual nos permite establecer una comunicación inalámbrica mediante WiFi, así como trabajar de manera simultánea con distintas tareas a la vez gracias a su sistema operativo FreeRTOS y la presencia de dos núcleos en el procesador.

Este Trabajo Fin de Grado pretende mejorar las anteriores versiones de este sistema inercial, escogiendo por lo tanto un dispositivo de bajo coste económico capaz de ser controlado a distancia. El auge de los dispositivos móviles con pantalla hace viable la creación de una interfaz HTML amigable que permita la conexión con el dispositivo.

UNIVERSIDAD
DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

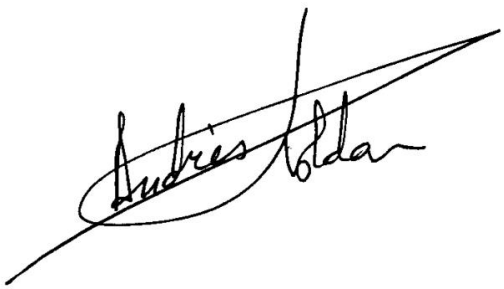
AUTORIZACIÓN DE LECTURA DE
TRABAJO FIN DE CARRERA

D. Andrés María Roldán Aranda, profesor del Departamento de Electrónica y Tecnología de Computadores de la Universidad de Granada, como director del Trabajo Fin de Grado titulado "**Air Bearing Platform Simulation for Reaction Wheels controlled by Espressif ESP32**" y realizado por el alumno D. Francisco José Llave Iglesias

CERTIFICA/N: que el citado Trabajo Fin de Grado, ha sido realizado y redactado por dicho alumno y autorizan su presentación.

Granada,

Fdo. Prof. D. Andrés María Roldán Aranda



UNIVERSIDAD
DE GRANADA

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

AUTORIZACIÓN DE DEPÓSITO EN LA BIBLIOTECA

Yo, D/Dña. Francisco José Llave Iglesias con DNI 76664621D, autor del Trabajo Fin de Grado titulado "**Air Bearing Platform Simulation for Reaction Wheels controlled by Espressif ESP32**" realizado en la Universidad de Granada,

DECLARO: explícitamente que asumo la originalidad del trabajo, entendida en el sentido de que no ha utilizado fuentes sin citarlas debidamente.

AUTORIZO: al depósito de dicho Trabajo en la Biblioteca de la Universidad de Granada, y de la visualización a través de Internet.

Granada,

Fdo. D/Dña. Francisco José Llave Iglesias

Agradecimientos:

Lo primero de todo es agradecer a mis compañeros de laboratorio. Gracias a Sharif por ayudarme tanto con la fabricación de la PCB, como por hacer más amenas las mañanas en el laboratorio. A Flo, por ser tan amable y atento (y por gustarle tanto los coches). A Irene, con la que nos hemos echado risas (o llantos) a la par. Cómo no, agradecer a Andrés esa labor inhumana que tiene de aguantarnos y enseñarnos. Estos meses has sido como un segundo padre, te echaré de menos.

Luego agradecer de corazón a mi nueva familia inesperada este año. Juan, Juanca, Soriano, Alexa, Joserto, Mangel y Mario. Muchísimas gracias por vuestras magníficas ideas, por vuestra atención y dedicación hacia mí, y por vuestro cariño. Este trabajo no hubiera sido posible sin vosotros.

A Claudia. A mi hermana, compañera, amiga y alma gemela. No seguiría vivo de no ser por ti.

Agradecer cómo no, a toda mi familia. A mi tía, por creer siempre en lo que he hecho y apoyarme cuando estaba peor. A mis primas, en especial a mi prima Azucena por sacarme del agujero mental en el que me metí cuando estuve en Madrid, y a Amabel, por esas innumerables horas estudiando matemáticas en cuarto. A mis tíos, Sergio e Irene, por los planes que siempre están organizando. A mis primos, primas y hermanos, que siempre me sacan una sonrisa. A mi madre, no sólo por ser una madre ejemplar, si no por ser una profesional incansable, en la cual me he inspirado siempre. Gracias, mamá.

Por último, agradecer a mis abuelos. Mis padres. Mis mentores en la vida, incansables y fuertes. Os quiero muchísimo.

Index

List of Figures

List of Tables

Glossary

Altium Designer® 19 EDA software used to design PCB from schematics. It allows 3D Design, as well as electronics simulation.

BUNGARD RoutePro 2000 © is a German program used to control the CNC PCB production machine and it was chosen due to its free availability and easy-to-use interface as well..

CircuitCAM 5.0 © CAM system for PCB Gerber files transformation into manufacturing files..

CubeSat Miniaturized satellite normally for space research, with dimensions of $1\,\mathrm{dm}^3$ and mass lower than 1.33 kg per unit.

ESP32 Advanced microchip with wireless communication capabilities like WiFi and Bluetooth, as well as a Real Time Operating System.

GranaSAT GranaSAT is an academic project from the University of Granada originally consisting in designing and developing a picosatellite (CubeSat). Coordinated by the Professor Andrés María Roldán Aranda, GranaSAT is a multidisciplinary project with students from different degrees, where they can acquire and enlarge the knowledge necessary to face an actual aerospace project.

Housekeeping Telemetry data associated to the health state of the instruments and devices of the spacecraft, in contrast the so-called **space telemetry** or simply telemetry, which gathers the real observations.

Simplify 3D © Program which converts STL files into GCODE files.

SolidWorks® CAD Software from Dessault Systèmes for 3D Mechanical Design.

Acronyms

ACK-NACK Acknowledgement- Negative Acknowledgement.

EDA Electronic Design Automation.

GA Gripping Arms.

I2C Inter-Integrated Circuit.

ID Inertial Disk.

IMU Inertial Measurement Unit.

PCB Printed Circuit Board.

PFM Pulse Frequency Modulation.

PWM Pulse Width Modulation.

SLLM Simple Light Load Mode.

Chapter 1

Introduction

This Bachelor's Thesis is presented as a guide for the next generation of students in GranaSAT laboratory, which comprehends a compilation of the knowledge acquired during my electronics engineering degree, as well as all of the information received during this intense last year's degree. This project aims to help future partners to understand, design and test new air bearing platforms relying on the advances of technology present at that time. The overall goal of my Bachelor's Thesis was developing an air bearing platform controlled by an ESP32 microcontroller, supported by older thesis.

I chose this ambitious project chasing the opportunity to work hand in hand with electronic components and CNC machines, the design & manufacturing of PCB, microcontroller programming, CAD modelling, etc. This Thesis has supposed a hard design process, as many of the programs used were new for me, and the GranaSAT equipment management took a lot of time to comprehend and operate. This means a wide variety of knowledge was necessary in order to make a fully integrated project. Electronics, telecommunication, mechanics or even IT engineering skills were needed for the developing of this job. Moreover, the fact that this project was developed with a mentor who has a lot of experience in private sector jobs has allowed me to feel how the situation changes when you have a deadline, production problems arise, safety tests are mandatory, etc. and how to try to solve all those issues.

In order to keep an structure, the project has been developed considering the different parts of the design, i.e., **Electronic, Mechanical and Software Design**. This document also follows that division, which eases understanding and allows a natural progress to the reader.

This Bachelor's Thesis fits within GranaSAT, a multidisciplinary project which gathers people from a variety of fields who are committed to acquiring new knowledge related

1

to Electronics and Aerospace Engineering. Since its origins, one of its main purposes has been getting a CubeSat in orbit, one of the reasons of this project; however, today its goals go far beyond, and a wide range of devices and projects are being developed in collaboration with different students and companies.

Figure 1.1 – *GranaSAT Logo*

1.1 Motivation

In 2018 I met my current mentor Andrés Roldán in one of the many courses he bestows, the SolidWorks® one. He used to explain how his work at GranaSAT was and how many things we would be able to learn with him. I was impressed of the quantity of projects he was involved in, so I decided to join him in some of them.

First, the summer of 2019 he guided me through the first steps in what would turn out to be this Thesis. While I was doing a summer job in a hotel in Guernsey, an island from United Kingdom, I started working with Altium Designer and the CAM package from SolidWorks during my spare time (Figure 1.2), in order to practice for future GranaSAT projects and to manufacture the components I had been designing during the SolidWorks course, in the laboratory CNC milling machines.

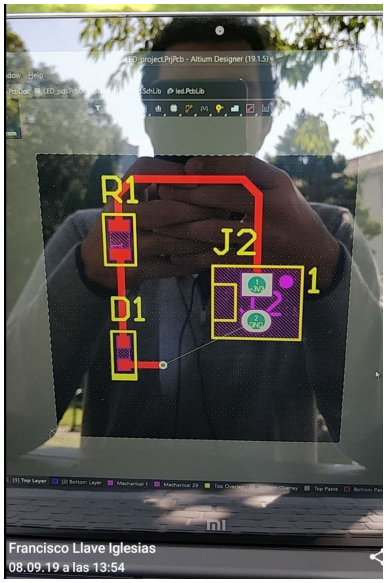
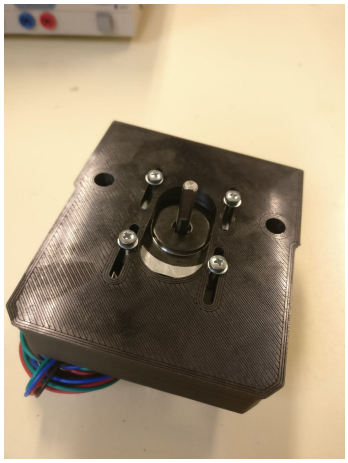
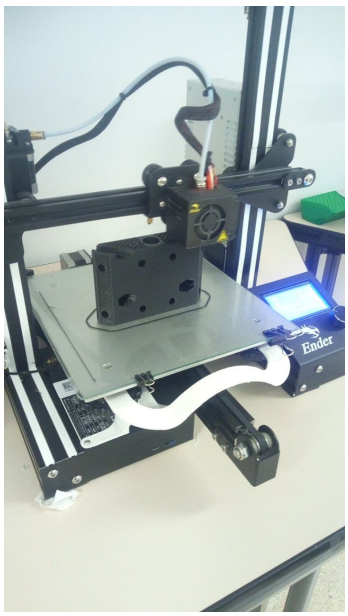


Figure 1.2 – *First PCB in Altium while working in a Guernsey park after work*

After that, he offered me to be the manager for a group project where we were designing a CNC machine made by 3D printed components (Figure 1.4), but due to the covid-19 world pandemic we had to cancel the project, as we needed laboratory tools and machines that we did not have available in any other place.



(a) *One of the lateral motors with its 3D printed support*



(b) *Printing the Z axis motor mount*

Figure 1.3 – *CNC 3D Printed Milling Machine Project*

1

On February 2020 he offered me a fulfilling experience: the opportunity to work for ESERO, an ESA subsidiary in Spain which promotes engineering and science between young people, from school to pre-university students. This job consists of communicating with pupils working on the CANSAT project, which basically entails manufacturing a can-size-satellite with all of the necessary electronic, telecommunication and landing equipment, and receiving from them the report of the mission. Report structure recommendations, help with circuitry or programming and an ongoing assessment were the requirements of this job. Then, the summer 2020 a launch event had been postponed due to covid19, but luckily ESERO was able to organize it, being a success. I happened to like it so much that this year 2021 I joined the team again.

(a) *CanSat Mentors Meeting at Parque de las Ciencias*

(b) *Rocket made by the Universidad Politécnica de Cataluña*

Figure 1.4 – *CanSat ESERO Competition 2020*

All of the former experiences made my choice of doing my Bachelor's Thesis with GranaSAT quite easy. Having organized my degree to leave my last year with barely no subjects aiming to develop a multidisciplinary thesis, which would resume all of the experiences (degree's and non-degree's ones) I have had during these years. The interoperability and cooperation with other partners at the lab was not only advisable but even necessary. Some of them did know how to operate the CNC or laser machines, and they would teach me their skills. All of these complexities with equipment, the large amount of information received every day and a nice work environment translates into a group of professionals working towards a same goal.

The *Air Bearing Platform controlled by an ESP32* which emerges from this Bachelor's Thesis is the result of a painstaking process of engineering, study and comparison of the cutting-edge technology used in space which I sincerely hope that can be useful for the GranaSAT Project in the future.

1.2 Prior art. Problem Statement

1.2.1 Simulating a Zero-Gravity Environment

[13] This simulation system provides a low friction load-bearing interface between surfaces, thanks to the use of a thin or pressurized gas. In our case we use an Aerostatic bearing with porous surface. This kind of bearing is externally-pressurized (our compressor-tank which appears in Figure 4.3) and injected in the clearance of the bearing. However, this system requires an external gas compression system, which induces costs in terms of complexity and energy.

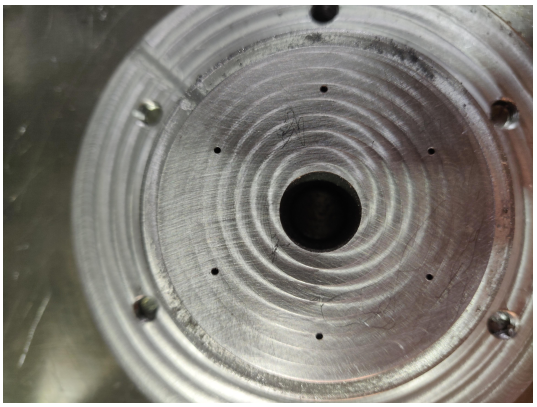


Figure 1.5 – *Detail of the Porous Surface*

1.2.2 Rotation in Space. Reaction Wheels.

[36] When the spacecraft must be rotated by very small amounts, such as keeping a telescope pointing at a star or keeping the overall system (satellite, spacecraft, ...) balanced, reaction wheels are used. These tools allow the control of the attitude of a satellite without the use of thrusters, which reduces the mass fraction needed for fuel.

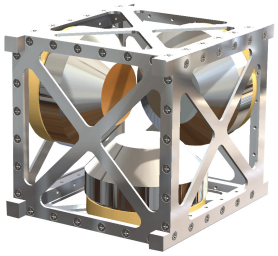


Figure 1.6 – *Example of reaction wheels in a CubeSat structure [37]*

1

It consists of a flywheel used primarily in spacecraft or satellites for three-axis attitude control. It's characteristic for providing a high pointing accuracy, being useful when spacecraft must be rotated by very small amounts.

In our case it is operated as a momentum wheel. Its inertial mass and the angular acceleration allow us to obtain a torque, which is used to balanced or rotate the simulation platform. Usually, the wheels will be rotating at a constant velocity, and accelerating when needed to create a force towards a direction, resulting in a movement of the system against that direction (explanation in Subsection 3.1.2)

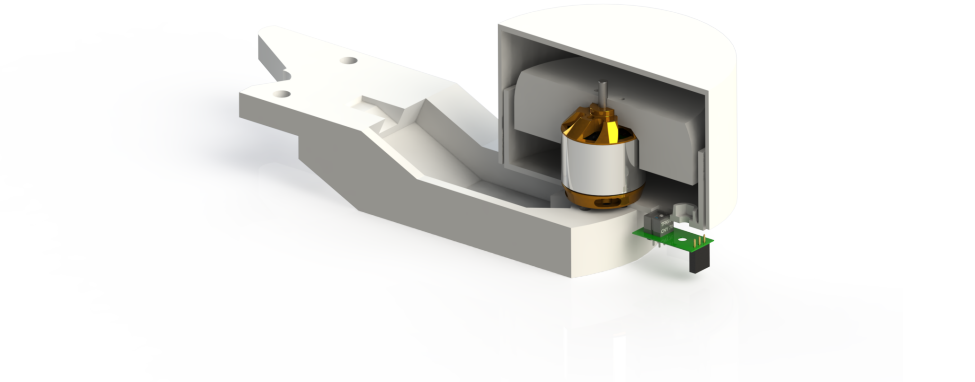


Figure 1.7 – *Render from the sphere and the pneumatic bearing system in SolidWorks®*

1.3 Project Goals & Objectives

1.4 Project Structure

The project is structured as follows:

These chapters are:

- **Chapter one**. This chapter will show the project's introduction and our main purposes with its realization, motivational aspects of myself and the productivity organization of the working time with a Gantt Diagram.
- **Chapter two**. The second chapter addresses the Mission Engineering step of a project. Requirements, technical objectives and functional aspects of the project will be defined.
- **Chapter three**. According to the system engineering methodology applied, in this chapter we will address the analysis of our project's elements, keeping the requirements in mind.
- **Chapter four**. The fourth chapter translates the client's requirements from the

mission engineering and the technical approaches from the system engineering into real-world solutions. This is extensively written for the purpose of understanding how the project has been developing during the last months.

- **Chapter five**. Validation is a mandatory chapter if we want to reassure the project's integrity and viability. Here we are able to review if the project meets the requirements and how the output was.
- **Chapter six**. Finally, conclusions and future approaches are seen in this chapter.
- **Addendum**, Appendices and Budget are located in this chapter.

1

Chapter 2

Mission Engineering

Determining the client's needs is the goal of this chapter. Client's initial needs are not likely to change, so they must be taken as the main goals of the project.

This chapter will address top level objectives, defining the Functional Requirements in order to establish the necessary steps to follow.

2.1 Introduction to Mission Engineering

Tasks to perform, how they have to be performed and how much budget we have to make it work properly. These are usually the steps we have to follow, in order to determine the base of the project. We have to be aware of the capabilities and the constraints from the equipment, as well as the workers.

Excessive optimism or ambitious objectives may not be the answer to get paramount results. First, we need to stay focused on the client's requirements, and if by any chance we are able to enhance those requirements (without increasing the budget, unless the client allows it) and not compromise the deadline imposed, that will be great.

2.2 System Objectives & Functional Requirements Definition

2.2.1 Mechanical Platform

The mechanical platform must fulfill the necessary requirements in order to make the simulation plausible. Not only the performance and proper functioning of the system will be discussed, but also these will be certified to work fine in harsh environments, such as the space.

Technical Objectives	
Ref.	**Primary**
MP.Obj.1	Allow the test of balancing and rotating the whole system, emulating space conditions.
MP.Obj.2	Integrate external elements to the platform, such as the arms for the reaction wheels, allowing vertical positioning for PCB or battery placement.
MP.Obj.3	Make the platform center of mass as centered as possible , leaving the platform as balanced as feasible.
Ref.	**Secondary**
MP.Obj.4	Scalable, different weights should not alter the system a lot.
MP.Obj.5	Low cost philosophy.

Table 2.1 – *Mechanical Platform - Technical Objectives*

Fulfilling *Primary Objectives* we would be meeting the client's petitions, but the *Secondary Objectives* would lead to a complementary success. We could extrapolate from the former table the requirements:

Ref.	Functional Requirements
MP.FR.1	Enough pressure to move the whole platform, all along with the reaction wheels and the control elements, and to withstand vibrations provoked by the reaction wheels spinning.
MP.FR.2	Low friction for zero-gravity emulation.
MP.FR.3	Scalability. The system could vary its weight and the air bearing system must not suffer considerable changes.
MP.FR.4	Balanced. The system should stay as balanced as viable when static, so the energy required could be reduced.

Table 2.2 – *Mechanical Platform - Functional Requirements*

2.2.2 Control Unit

A control unit is the component or group of components in charge of making effective tasks, such as data sending or reception, and communication links, be it wireless or wire like. This unit must be able to perfectly integrate within the simulation platform, dealing with data handling, sensor reading and the capability of establishing stable wireless & or wired communications between the Control Unit (Subsection 2.2.2) and the Monitoring

Station (Subsection 2.2.3). This unit should be ready to work with I2C local protocol communication for sensors like the IMU, as well as digital and analog signal reading and PWM signal generation.

The complexity of this subsystem is the largest of the project. A wide range of the technical spectrum involved in the Thesis is displayed in here. From **Mechanical Engineering** (strong vibrations support capabilities, air bearing platform with a capable foundation to withstand the reaction wheels and their gripping arms) to **Electronics Engineering** (electronic design, communications) or **IT Engineering** ($C/C++$ programming, operating software development, HTML/CSS and JavaScript for server design).

This control unit is intended to be controlled, addressed and managed through an HTML interface, available for all kind of on-screen mobile devices with WiFi capabilities, making it easy for an inexperienced or novice user. Table 2.3 and **??** show us the *Technical Objectives* and *Functional Requirements* from the **Control Unit**.

Table 2.3 lists Technical Objectives for the Simulation CubeSat.

Technical Objectives	
Ref.	**Primary**
CU.Obj.1	Allow to test the required functions by the client, dealing with different communication protocols, data sending and reception, and also energy management.
CU.Obj.2	Being able to cope with critical situations related to vibrations and noise.
CU.Obj.3	Provide the necessary components and characteristics for the rotational movement, like wires, track lanes, etc.
CU.Obj.4	Allow position determination and control the system based on it.
CU.Obj.5	Allow wired communications, when used in the simulation platform, so microcontroller configuration or local testing is feasible.
Ref.	**Secondary**
CU.Obj.6	Measurement of ambient conditions.
CU.Obj.7	Wireless communication availability.
CU.Obj.8	Angular acceleration could be helpful to determine the torque, knowing exactly how fast we have to turn the inertial disks .
CU.Obj.9	Balance control with PID controller.
CU.Obj.10	It should contain every single needed component with the minimum possible weight, in the smallest size practicable and to place every component optimally, in order to reduce space.

Table 2.3 – *Control Unit - Technical Objectives*

Table 2.4 shows the Functional Requirements determined.

Ref.	Functional Requirements
CU.FR.1	Applicable interconnection standards for local communication protocol testing, such us digital or analog signal reading, I2C protocol, etc.
CU.FR.2	Sensors for orientation, position and rotation rate measures.
CU.FR.3	Autonomous capabilities for sensor data reading and data sending,.
CU.FR.4	Ensure energy management capabilities, like battery under-voltage protection.
CU.FR.5	Wireless communication standards like WiFi or Bluetooth for data management and control feasibility.
CU.FR.6	Fast response between the interface and the simulation platform control unit.
CU.FR.7	USB presence is mandatory, for local testint and programming.
CU.FR.8	The whole system shall provide Housekeeping telemetry to enable the verification of the nominal behaviour of sensors, actuators and on-board functionalities, on ground.

Table 2.4 – *Control Unit - Functional Requirements*

2.2.3 Monitoring Station

This could be designated as the central server of the operation. This station will comprehend an HTML interface for the end user to control and supervise the mission from an on-screen device. Wide availability of these devices make it easy for everyone to access the information panel from the simulation platform.

Wireless communications need to be addressed properly in order to get first-rate data, achieving as well a real time response of the Control Unit (Subsection 2.2.2) from the simulation platform. Table 2.5 and Table 2.6 show the technical objectives and functional requirements of this subpart, respectively.

Technical Objectives	
Ref.	**Primary**
MS.Obj.1	Allow inexperienced or novice users to effectively communicate with the platform.
MS.Obj.2	Allow sending control commands to the platform, in addition to receiving telemetry and payload data in a clear way from it.
MS.Obj.3	Allow the definition of new control commands.
MS.Obj.4	Allow the definition of new control commands.
Ref.	**Secondary**
MS.Obj.5	Graphics and animations for easier user reading.
MS.Obj.6	The presence of a capable software in order to manage transmitted and received data.

Table 2.5 – *Monitoring Station - Technical Objectives*

Ref.	Functional Requirements
MS.FR.1	Communication protocol which adapts to the needs of the simulation platform, in terms of frequency, payload capabilities and remote control abilities.
MS.FR.2	The subsystem must have a software with real time operating capacity, as well as enough processing potential to establish WiFi communications while multitasking at the same time.
MS.FR.3	A friendly interface is mandatory. Data reading or control commands should be easy-to-use and clearly exposed.
MS.FR.4	The subsystem management shall be able to be used in a wide variety of commercial devices.
MS.FR.5	Prepare fragile electric and noise-sensitive components to resist strong vibrations.

Table 2.6 – *Monitoring Station - Functional Requirements*

Chapter 3

System Engineering

3.1 Mechanical Platform Analysis

3.1.1 Air Bearing System

3.1.1.1 Multi-flow Pneumatic Sphere Design

Air bearing platforms [13] have been used for satellite rotational testing for decades. In order to replicate a minimal-torque environment like in space, this almost frictionless device allows us to partially simulate a zero-g environment. Paramount variables such as the pitch, roll or yaw (heading).

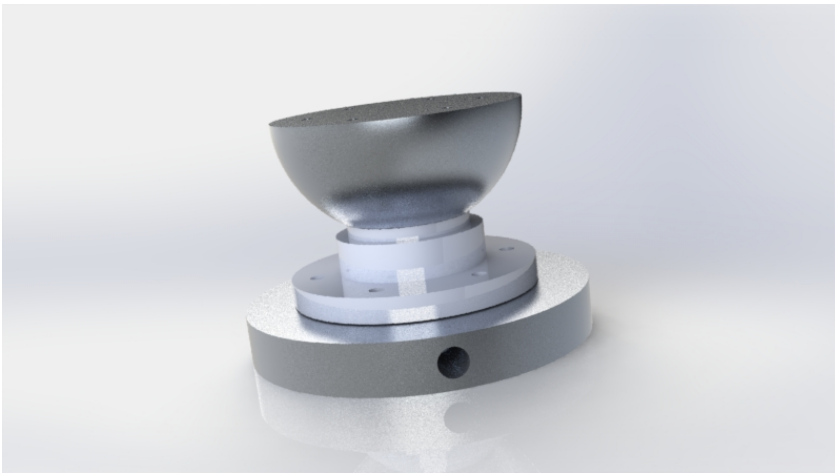


Figure 3.1 – *Render from the sphere and the pneumatic bearing system in SolidWorks*

To create the conditions present in space we must design an environment with similar characteristics. This pneumatic sphere allows us to simulate the lack of friction.

Usually, this type of design is made with only one air flow entrance directed to the sphere. Sending all of the air flow through one centered hole can be easier to design, but it is not a good option. The nearest parts of the sphere will work fine, but as we go far from the air flow hole the pressure will decrease, thereby not creating the ideal "non-friction" system.

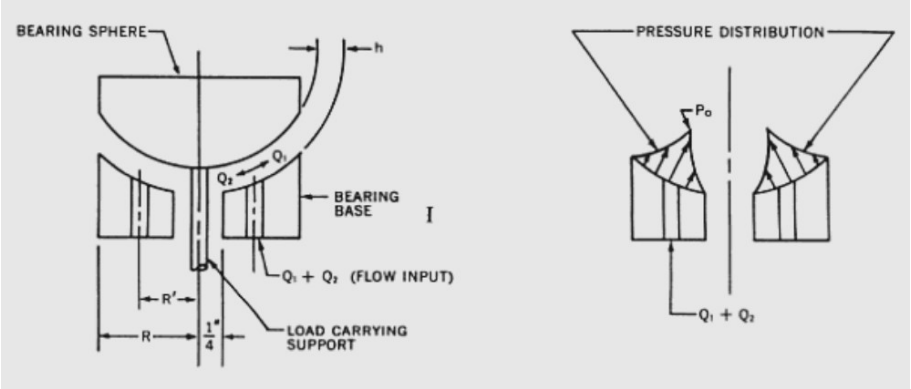


Figure 3.2 – *Pressure Distribution in the Cup Holder [48]*

The following equations were based on locating the capillaries so that the flow towards the bearing center equals the flow outwards. Figure 3.2 shows the flow conditions and pressure distribution on which the design was based.

3.1.1.2 Determination of the Radii [48]

$$Q = \frac{\Delta P \cdot b \cdot h^3}{12 \cdot \mu \cdot l} \rightarrow \begin{cases} \Delta P & \text{Pressure Gradient,} & \frac{kg}{m^2}, psi \\ b = 2\pi r & \text{Central hole circumference,} & (cm, inches) \\ h & \text{Distance between spherical surfaces,} & (cm, inches) \\ l = dr & \text{Differential length,} & (cm, inches) \\ \mu & \text{Air dynamic viscosity,} & \frac{N \cdot s}{m^2}, \frac{lb \cdot s}{in^2} \end{cases}$$

The air flow should be the same through all of the exit holes. ΔP is the same in both directions, being the drop from the initial pressure to the outside pressure, where h is a geometrical constant. The capillaries are equally spaced around the bearing center at a distance R'; hence the flows may be considered radially symmetrical. It is possible to achieve if we impose the next condition:

$$Q_1 = \frac{\Delta P \cdot b_1 \cdot h^3}{12 \cdot \mu \cdot l_1} \equiv \frac{\Delta P \cdot b_2 \cdot h^3}{12 \cdot \mu \cdot l_2} = Q_2,$$

from which

$$\frac{dr_1}{r_1} = \frac{dr_2}{r_2}. \tag{3.1.1}$$

Integrating equation Equation 3.1.1 from the capillary orifices to the outlets (Figure **??**), we have

$$\int_{R'}^{R} \frac{dr_1}{r_1} = \int_{\frac{1}{4}}^{R'} \frac{dr_2}{r_2} \Rightarrow R' = \frac{\sqrt{R}}{2}.$$

$$R' = \frac{\sqrt{R}}{2}. \tag{3.1.2}$$

So for $R = 35\text{cm}$, $R' = 17.5\text{cm}$

3.1.1.3 Determination of the Pressure Distribution [48]

The peak pressure P_0 required at the input orifices to support the load on the bearing sphere. The design load is, in this instance, $W = PESODELSISTEMATOTAL$. Let $A = 2\pi r dr$ represent the flat projection of the bearing sphere surface area; then, since $dW = pdA$

$$W = \int_{R'}^{R} p_1 2\pi r\, dr + \int_{\frac{1}{4}}^{R'} p_2 2\pi r\, dr,$$

where p_1 and p_2 are the pressure distributions as given by Equations (**??**) and (**??**).

Thus

$$W = 2\pi \int_{R'}^{R} \frac{6Q\mu}{\pi h^3} r\, dr + 2\pi \int_{\frac{1}{4}}^{R'} \frac{6Q\mu}{\pi h^3} (ln4r) r\, dr, \tag{3.1.3}$$

Next an expression for Q in terms of P_0 must be found. From Equation (**??**), since $p_1 = P_0$ when $r = R'$,

$$Q = \frac{P_0 \pi h^3}{6\mu ln\frac{R}{R'}}. \tag{3.1.4}$$

The use of this expression reduces Equation (Equation 3.1.3) to

$$W = \frac{2\pi P_0}{ln\frac{R}{R'}}\left(\int_{R'}^{R} rln\frac{R}{r}\,dr + \int_{\frac{1}{4}}^{R'} rln4r\,dr\right) =$$
$$= \frac{2\pi P_0}{ln\frac{R}{R'}}\left(\int_{R'}^{R} rlnR\,dr - \int_{R'}^{R} rlnr\,dr + \int_{\frac{1}{4}}^{R'} rln4\,dr + \int_{\frac{1}{4}}^{R'} rlnr\,dr\right).$$

Integrating, solving for P_0, and substituting the numerical values of W, R, and R′ then gives

$$W = 2.7P_0.$$

Obtaining an expression which relates maximum load with inwards pressure.

3.1.2 Reaction Wheels. Physical Approach.

For the complete understanding of the reaction wheel physical approach is necessary to start from the torque term.

- Torque

  The quantitative measure of the tendency of a force to cause or modify the rotational motion of a body is called torque [51]. The torque (or moment) for a magnitude force whose action line is a perpendicular distance l from O is:

  $$\tau = Fl$$

  Using the lever arm Force diagram and the former equation:

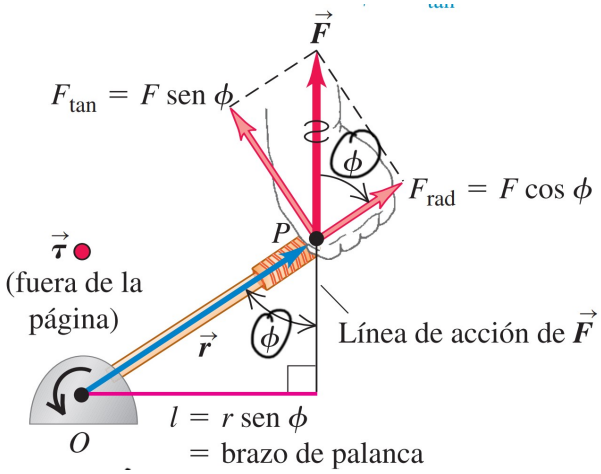


Figure 3.3 – *Torque and Force diagram with lever arm example [51]*

We know there is a radial component F_{rad} along the $\vec{r}$ direction, as well as a

tangential component in right angle position $\vec{F}$, then we have

$$F_{\tan} = F \operatorname{sen} \rightarrow \tau = r(F \operatorname{sen}) \rightarrow \tau = Fl = rF \operatorname{sen} \phi = F_{\tan} r$$

Additionally, when a force $\vec{F}$ acts over a point which has a positional vector $\vec{r}$ respect an origin O, the vectorized torque results in

$$\vec{\tau} = \vec{r} \times \vec{F}$$

- Angular Acceleration

  We want to show that the angualar acceleration of a rigid solid is proportional to the sum of the torque components all along the rotation axis.[51]

  Choosing the z axis for simplicity and using the second Newton's law,

  $$F_{1\,\tan} = m_1 a_{1\,\tan} \rightarrow F_{1\,\tan} r_1 = m_1 r_1^2 \alpha_z$$

  The tangential acceleration of a particle can be expressed in angular acceleration α_z terms. Knowing that $a_{1,\tan} = r_1 \alpha_z$ we observe that the torque respect the rotational axis is $F_{1,\tan} r_1$. Radial axis forces F_{rad} and z-axis forces F_z do not contribute to the tangential force around the z-axis, as those do not modificate the position of the particles.

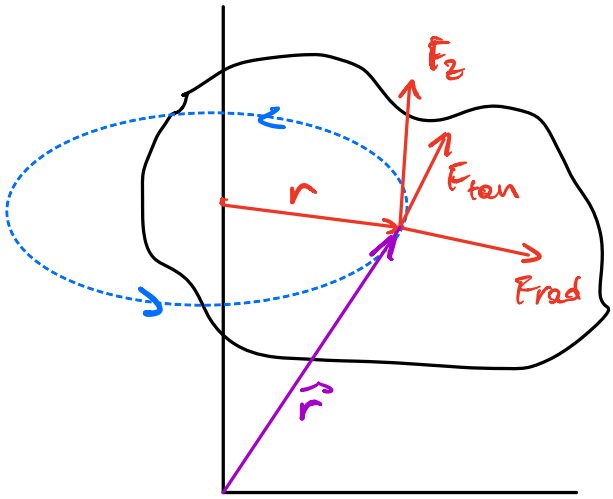


Figure 3.4 – *Radial and Axis Forces Diagram*

Adding to the equation that $m_1 r_1^2 = I_1$, inertial moment of the particle around the rotational axis, we have

$$\tau_{1z} = I_1 \alpha_z = m_1 r_1^2 \alpha_z \Rightarrow \sum \tau_{iz} = \left(\sum m_i r_i^2 \right) \alpha_z$$

- Angular Momentum

  The relation of the equation $\vec{\tau} = \vec{r} \times \vec{F}$ is the same as the linear momentum has [51].

As we already know,

$$L = mvr \operatorname{sen}\phi = \text{mvl} \tag{3.1.5}$$

where the angular momentum is perpendicular to the plane xy. If a force takes action over a particle, changing its velocity and linear momentum, it could also change the angular momentum. How fast the change of the angular moment happens is actually the torque. Deriving respect time

$$\left[\frac{d\overrightarrow{\boldsymbol{L}}}{dt}\right] = \left(\frac{d\overrightarrow{\boldsymbol{r}}}{dt} \times m\overrightarrow{\boldsymbol{v}}\right) + \left(\overrightarrow{\boldsymbol{r}} \times m\frac{d\overrightarrow{\boldsymbol{v}}}{dt}\right) = (\overrightarrow{\boldsymbol{v}} \times \widehat{m\overrightarrow{\boldsymbol{v}}}) + (\overrightarrow{\boldsymbol{r}} \times m\overrightarrow{\boldsymbol{a}})$$

As we know, $\overrightarrow{v} \times \overrightarrow{v}$ is zero, so we only have left the last term.

$$\frac{d\vec{L}}{dt} = \vec{r} \times \vec{F} = \vec{\tau} \tag{3.1.6}$$

Moreover, if the solid is rigid we can determine the absolute angular momentum of every particle. Considering the body is rotating around the z axis each particle has the same distance to this axis. This means that using **??** and the fact that $v = r\omega$,

$$L_i = m_i(r_i\omega)r_i = m_i r_i^2 \omega \tag{3.1.7}$$

where we can substitute by the inertial mass and consider a slice around the z axis, parallel to the plane xy.

$$L = \sum L_i = \left(\sum m_i r_i^2\right)\omega = I\omega \tag{3.1.8}$$

Virtually, considering the rigid solid we claim I is constant, so if the axis has a fixed direction in space we could assume that only $\vec{L}$ and $\vec{w}$ are able to change in magnitud, not in direction. So we can say

$$\frac{dL_z}{dt} = \left[\frac{Id\omega_z}{dt}\right] = I\alpha_z \rightarrow \sum \tau_z = I\alpha_z$$

This last expression is the one we were after. Once everything has been explained, we can proceed with the explanation of the movement by the reaction wheels.

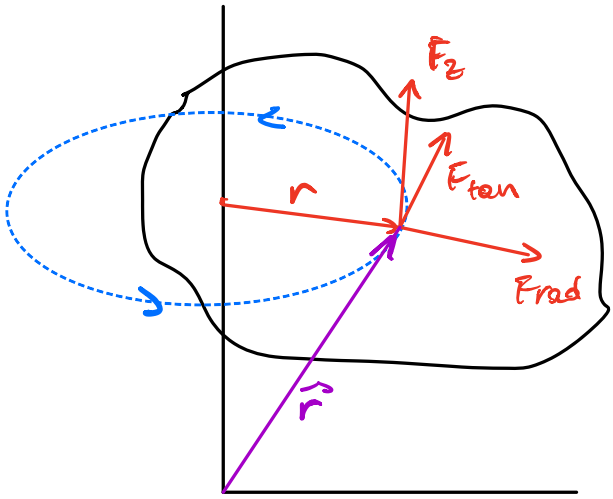


Figure 3.5 – *Radial and Axis Forces Diagram*

Controlling the angular acceleration with the motors we are able to produce torque enough to move the platform. But towards which direction? That is something we calculate with the resultant vector. The modulus of that vector could be calculated as appears in **??**, an easy example of how we plan to calculate the torque.

$$\sqrt{\|\vec{\tau}_z\|^2 + \|\vec{\tau}_y\|^2} = \tau_{res}$$

This links to a video that shows the rotation of the disk, in order to establish the ID torque direction.

3.1.3 Mass Budget

MASS BUDGET					
MODULE	DEVICE	INFO	MASS (g)	QUANTITY	TOTAL (g)
POWER SOURCE	Battery	LiPo Battery 2200 mAh	204	1	204
	Platform	Battery Platform	24	1	24
					228
GA	REACTION	Security Lids	41	3	123
	ARMS	Arm	100	3	300
		Screws & Nuts	10	3	30
					0
	TACHOMETER	PCB	7	3	21
					0
	ESC	ESC 30A Driver	32	3	96
					0
	DC MOTOR	Brushless D2826-6 Motor	50	3	150
					0
	ID	Inertial Disk	90	3	270
					990
CU	PCB	PCB with components	230	1	230
SUPPORT	Support Screws	Endless Screws	40	3	120
		M6 Screws	15	3	45
					165
AIR BEARING	SPHERE	Air Bearing Semisphere	750	1	750
				SYSTEM WEIGHT (g)	
					2363

3.2 Control Unit Analysis

3.2.1 Central Processing Unit

3.2.1.1 I2C

The ESP32 has two I2C bus interfaces which can serve as I2C master or slave, depending on the configuration. These serial ports are asynchronous, so every slave device with its respective address must answer to the master when asked.

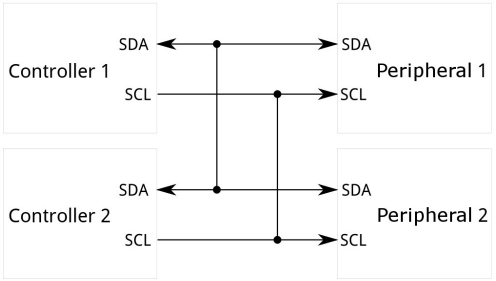


Figure 3.6 – *I2C Block Diagram [8]*

The messages are broken up into frames of data, where we have the start condition, the address frame, a read or write bit, two data frames separated by an ACK-NACK [8] bit and a final stop condition.

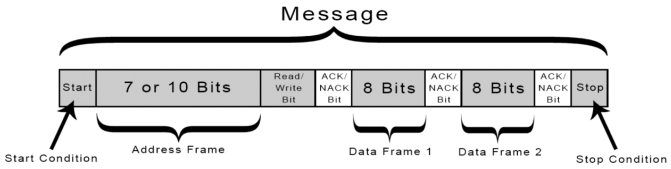
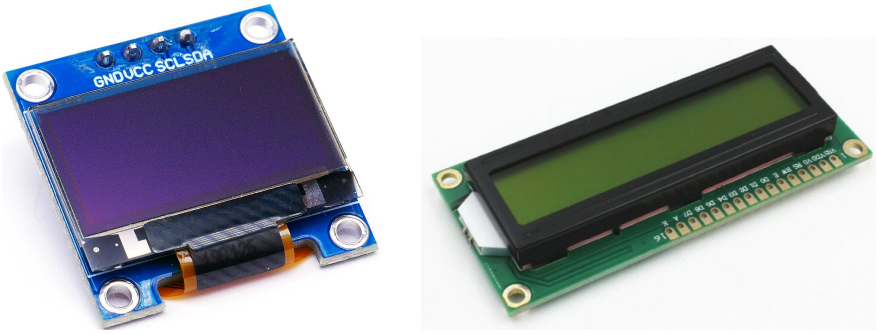


Figure 3.7 – *I2C Message [8]*

3.2.1.2 Sensors

3.2.1.2.1 Display

Choosing a display is a good choice for essential information visualization. We will compare two commercial displays and choose depending on our needs.



(a) *SSD1306 OLED* **(b)** *LCD HD44780*

Figure 3.8 – *Display Comparison [10], [9]*

	128x64 DOT MATRIX PANEL	LCD ADM1602K
Resolution	128x64 DOT MATRIX PANEL	16x2 MATRIX PANEL
Power Supply	1.65V to 3.3V	0 to 7 V
Common Maximum Sink Current	15 mA	
Communication Interface	I^2C	
Operating Temperature Range	-40^oC to 85^oC	-10^oC to 60^oC

Table 3.1 – *Displays Characteristics [49], [46]*

Choosing the SSD1306 implies portability, more on screen information, wide temperature range for harsh environments, etc.

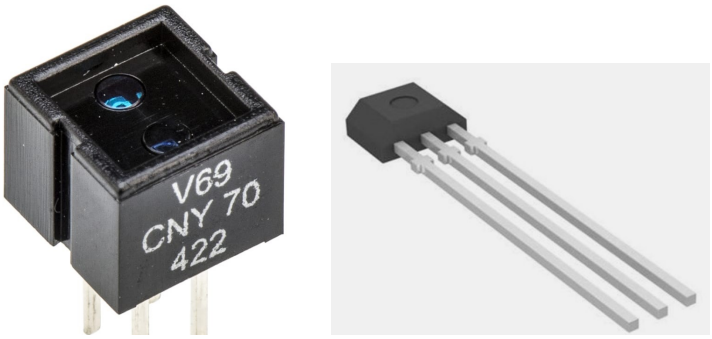
3.2.1.2.2 Tachometer

There are a few options when it comes to designing a tachometer. We will consider using a Hall Sensor and a CNY70 infrared emitter sensor.

- Comparison

  The functioning system of the *CNY70* resides in an emitter diode and a receiver transistor which allows the current flow depending on the quantity of light received from the diode. The maximum advisable distance is 2 cm.

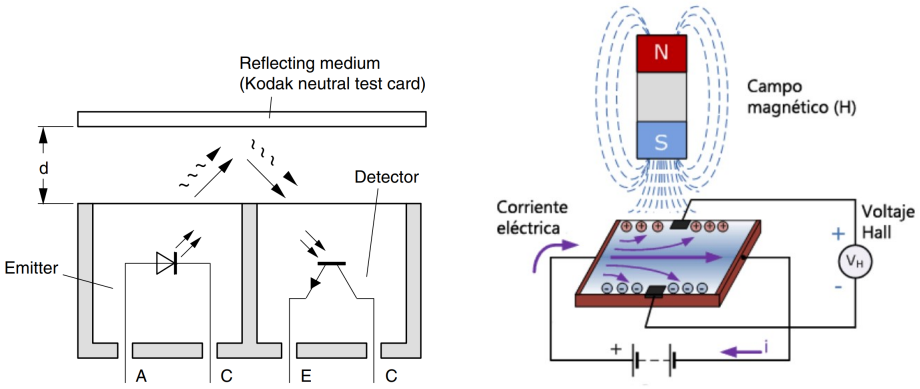
  On the other hand, the hall sensor functioning system depends on the changing magnetic field [24] . This system would not be suitable because it is not recommendable to use magnetic equipment in our case. Not only because of the presence of a magnetometer, but also because of the short distance with a brushless motor and the sensor, which could take corrupted data due to the motor's coil. In this case, the maximum recommendable distance is 1 cm, as the



(a) *CNY70* **(b)** *3144 Hall Sensor*

Figure 3.9 – *Tachometers Comparison [16], [29]*

magnetic strength (Gauss) starts losing integrity, and it would be necessary a very powerful magnet.



(a) *CNY70* **(b)** *3144 Hall Sensor*

Figure 3.10 – *Working Method Comparison [50], [7]*

The next Table 3.2 shows a technical comparison between both of them:

	CNY70 [50]	Hall Sensor 3144 [41]
Collector Current	0.3 to 1 mA	-
Operating Voltage	3 to 8 V	4.5 to 24 V
Magnetic Flux Density	-	Unlimited
Mean Current	max. 45 mA	25 mA
Typical Sound Output	92 dBA	-
Operating Temperature	$-30^{o}C$ to $70^{o}C$	$-40^{o}C$ to $85^{o}C$

Table 3.2 – *Tachometers Characteristics*

After considering both devices, choosing the CNY70 seems to be the best choice.

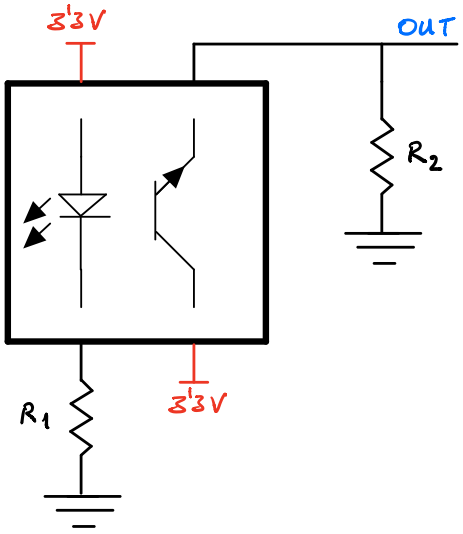


Figure 3.11 – *CNY70 Schematic [50]*

- CNY70 Sensor as a Tachometer

  Our inertial disks have a white line over a black surface which makes the infrared signal from the component to go back to the receiver BJT, putting the signal on HIGH.

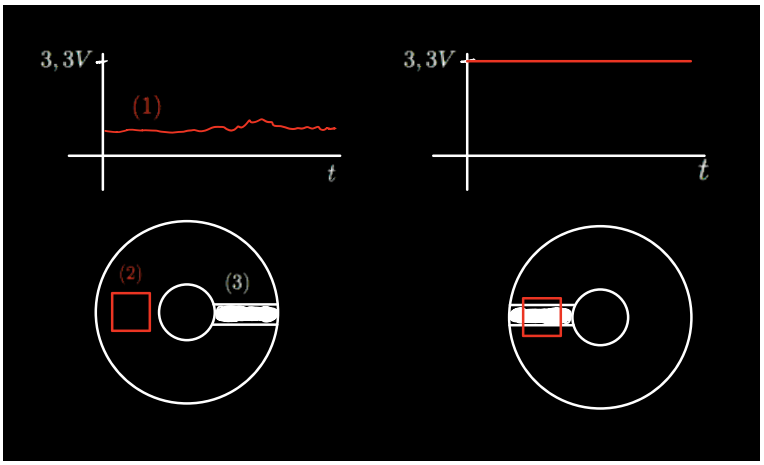


Figure 3.12 – *CNY70 Signal Behavior*

In Figure (Figure 3.12) number (1) represents the signal when the sensor (2) is positioned over a dark surface, while number (3) is the white mark, which puts the signal on HIGH when the sensor is over it. [15] This signal is then transformed into an RPM value, meaning we can be aware of the velocity from each disk.

$$\frac{Nrev}{\mu sec} \times \frac{1e^6 \mu sec}{sec} \times \frac{60sec}{min} = RPM \left[\frac{rev}{min}\right]. \tag{3.2.1}$$

The IMU (**??**) is supposed to work only with the accelerometer and gyroscope

sensor readings, but velocity can help determine if every disk works as expected. Also, it would be interesting to control the angular acceleration (α_z), as for the torque control we need that parameter, (α_z) can be obtained with the RPM velocity formerly calculated

$$RPM\frac{rev}{min}\times\frac{1min}{60sec}\times\frac{2\pi rad}{1rev}=w\left[\frac{rad}{s}\right]$$
$$\alpha_z=\frac{dw}{dt}\quad\xrightarrow[discretize]{}\quad\alpha=\frac{\Delta w}{\Delta t}\equiv\boxed{\frac{w_f-w_0}{t_f-t_0}}. \tag{3.2.2}$$

3.2.1.2.3 Buzzer

Figure 3.13 – *KXG1205 Magnetic Buzzer [32]*

Rated Voltage	5 V
Operating Voltage	3 to 8 V
Mean Current	max. 45 mA
Typical Sound Output	92 dBA
Operating Temperature	-30^oC to 70^oC

Table 3.3 – *Buzzer KXG1205 Characteristics [2]*

The magnetic buzzer used works with the application of a PWM signal. When the squared signal moves through the coil, a fluctuating magnetic field is produced, vibrating the disk at a frequency determined by the same signal. In order to make it sound a little softer a 270 Ohm resistor was put before the buzzer.

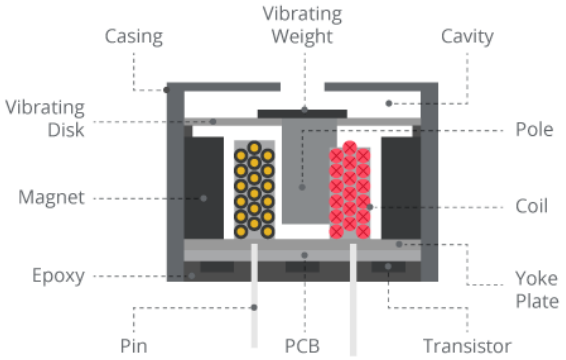


Figure 3.14 – *Buzzer Interior [33]*

Knowing this, we can determine that for a frequency of 2400 Hz (which is recommended by the datasheet) and a symmetric square signal (a constant vibration is produced thanks to the change from LOW to HIGH in the signal through the coil, which produces the sound) we will have the loudest sound possible from the buzzer.

3.2.1.3 Software. FreeRTOS.

It is a real time operating system for embedded devices. This system is written mostly in C to enhance code readability, portability and easy maintenance. It provides methods for threads, tasks, mutexes, semaphores, etc, in order to establish a multitasking platform with input and output control commands.

Figure 3.15 – *FreeRTOS Logo [11]*

One of the main attractions in our case of this operating system are tasks. These tasks allow our system to execute huge payloads of data (IMU data reading, WiFi connection, etc.) almost simultaneously.

These next diagrams by Shawn Hymel [38] show perfectly how tasks work. In this project we won't be using task resumes or task stop commands, as we want those tasks to be running permanently.

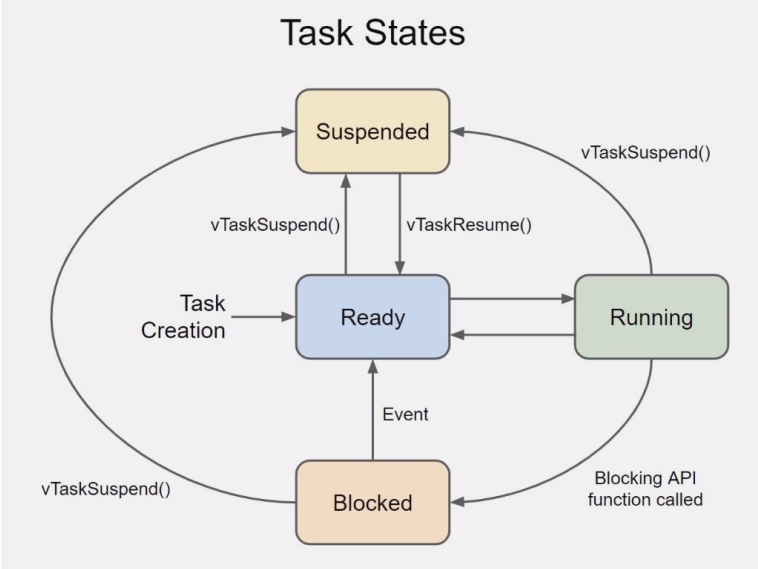


Figure 3.16 – *Task Block Diagram [38]*

3.2.1.4 Dual Core Implementation

Multiprocessing is available in ESP32. This actually allows us to work with a huge load of information, WiFi connection and multitasking.

To fully understand how multicore works, let's take a look at this diagram made by Shawn Hymel [38]

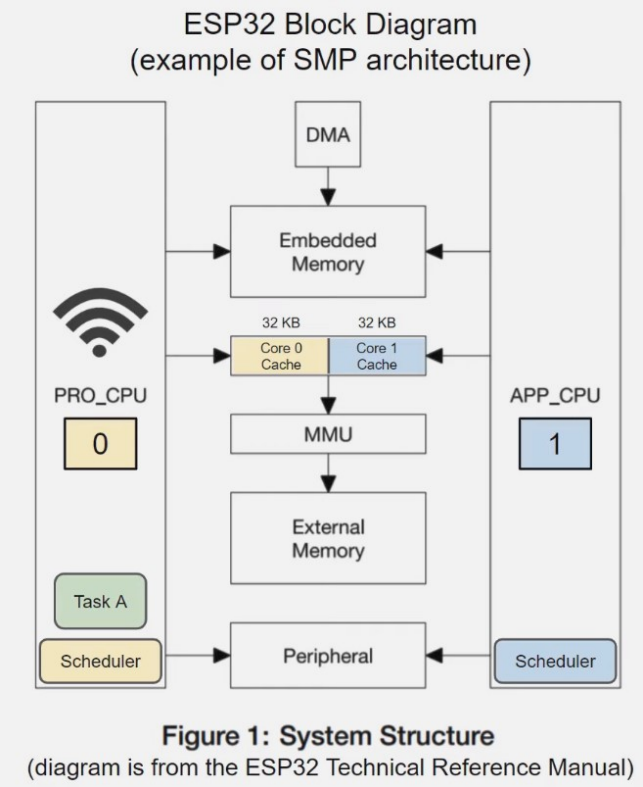


Figure 3.17 – *ESP32 Dual Core Diagram [31]*

Figure 3.17 shows how the two cores work seamlessly, sharing half of the cache and the whole embedded memory. WiFi symbol appears in Core 0, as it is intended to be there. In that case, we will be moving our tasks to CPU 1 in order to decrease the work load. This is done by *pinning* tasks to CPU 1 when defining them.

3.2.1.5 Communications

As commented, we will use preferably WiFi. The ESP32 implements a TCP/IP and full 802.11 b/g/n WiFi MAC protocol. IT supports the Basic Service Set STA and SoftAP operations [40].

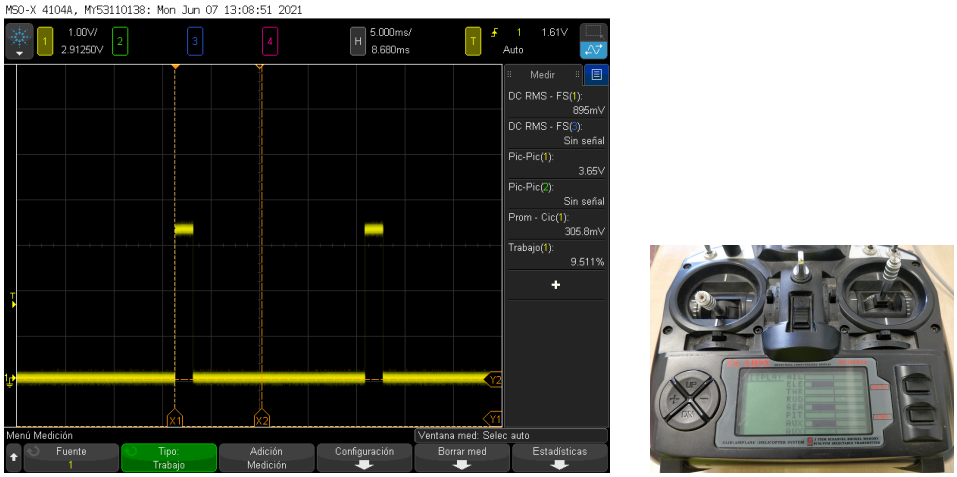
WiFi Radio and Baseband supported:

- 802.11b/g/n
- Adjustable transmitting power
- Up to 20.5 dBm of transmitting power
- Up to 150 Mbps of data rate

3.2.2 Stabilization Performance

3.2.2.1 Motor PWM Signal

For the determination of this signal we first studied the behaviour of our RC controller and the radio receptor which creates the signal. The throttle is the tool we need in order to accelerate or decelerate the motors, as well as their configuration with the ESC.



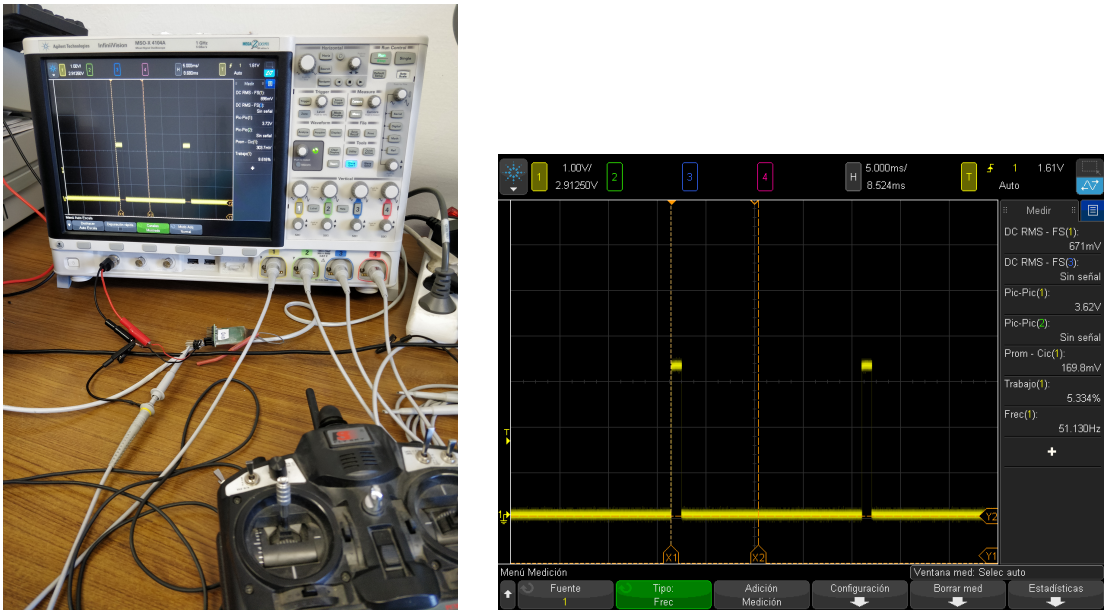
(a) *Max Throttle Position from Oscilloscope* **(b)** *Min Throttle Position*

Figure 3.18 – *Throttle PWM Determination*

With an oscilloscope we were able to see how the throttle works, in order to imitate the PWM signal in our program.

Now we were aware of the frequency for the PWM, 50 Hz, and the duty cycle range, which flows between 5.336 % and 9.511 % .

From the ESP32 manual [40] we can take a block diagram explaining how the MCPWM, the specific command from ESP32 to work with PWM signals, in order to better understand the future design.



(a) *Min Throttle Position View* **(b)** *Oscilloscope with min throttle*

Figure 3.19 – *Throttle PWM in the Oscilloscope*

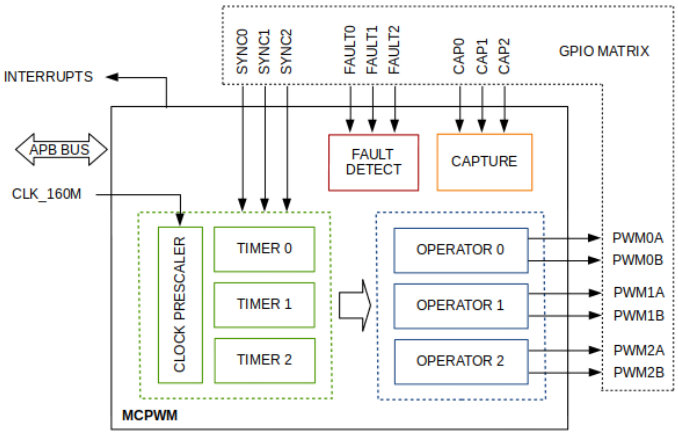


Figure 3.20 – *MCPWM Block Diagram [40]*

3.2.2.2 IMU

This board includes both gyroscope, accelerometer and magnetometer MEMS sensors, as well as a pressure sensor which is not intended for this purpose but that we will keep on due to possible future implementation (as this system is intended to be in a CubeSat, height and temperature could be interesting measures).

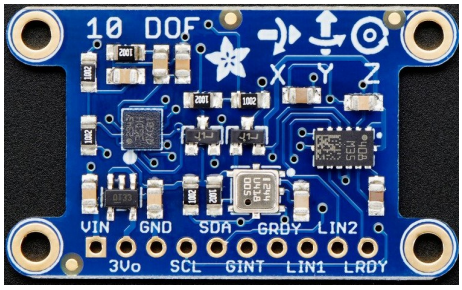


Figure 3.21 – *IMU Adafruit 10 DoF*

Sensors tend to lose precision due to noise, so their calibration is uppermost regarding reliability of the received data.

3.2.2.2.1 Accelerometer

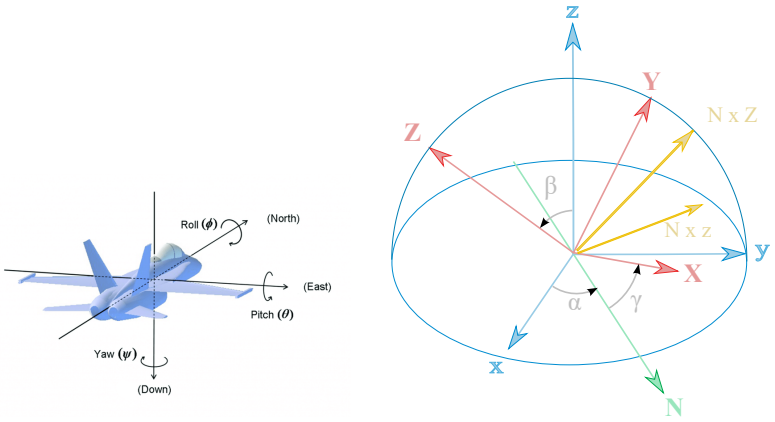
The accelerometer measurements are subjected to a large amount of noise compared to the gyroscope, varying depending on the static or dynamic state of the body, getting larger as the acceleration vector change rate grows. It is possible to substrate this error using the axis orthogonality property so as to determine the maximum gravity output value for each individual axis. Having the positive and negative value of acceleration allows us to obtain the scaling and steady-state error [12].

Supply Voltage	2.16 to 3.16 V
Current Consumption	1 to 110 µA
Operating Temperature	$-40^{o}C$ to $85^{o}C$
Magnetic Resolution	2 mgauss
Communication Interface	I^2C
Linear Accel. zero-g level offset accuracy	$\pm 60mg$

Table 3.4 – *LSM303DLHC Accelerometer Characteristics*

For the accelerometer we have three parameters that will determine the movement of our platform. Pitch, roll and yaw are the three degrees of freedom of our system. These words come from the aeronautical slang, and it is represented as the movements a flying device is able to perform. It is important to point out the fact that accelerometers are *very sensitive to vibration and other nongravity accelerations* [12]. This will be discussed in the gyroscope (paragraph 3.2.2.2.2) section.

Euler angles (deeper mathematical approach in Appendix C), but basically we can determine the orientation of a rigid body with respect to a fixed coordinate system. The following matrix is the result of a long mathematical study, present in the formerly



(a) *Pitch, Roll and Yaw* **(b)** *Euler angles*

Figure 3.22 – *Pitch, Roll and Yaw from Aeronautics perspective and Euler angles [23], [21]*

3

mentioned appendix, where we can see the Euler DCM.

$$\mathbf{R} = \begin{pmatrix} 1 & \theta_z & \theta_y \\ -\theta_z & 1 & \theta_x \\ -\theta_y & -\theta_x & 1 \end{pmatrix}.$$

This result could be seen graphically for our MEMS purpose this way:

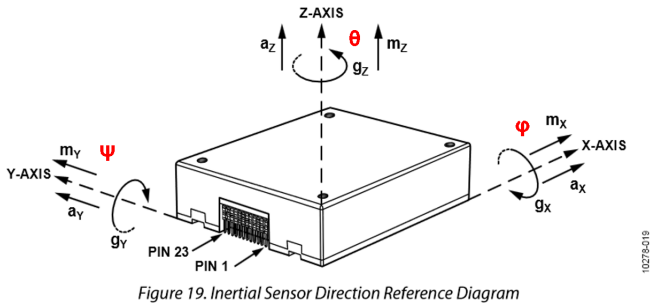


Figure 3.23 – *Euler in IMU [22]*

3.2.2.2.2 Gyroscope

The gyroscope is a device capable of measuring the angular velocity. In our case we are using a MEMS, a vibrating structure gyroscope (defined by the IEEE as a *Coriolis Vibratory Gyroscope*) [25] that uses a vibrating structure to determine the rate of a rotation [27]. The name *Coriolis* of its denomination comes from the underlying physical principle on which this device holds; the Coriolis effect causes the gyroscope to exert a force on its support. Therefore, by measuring this force the rate of rotation can be determined.

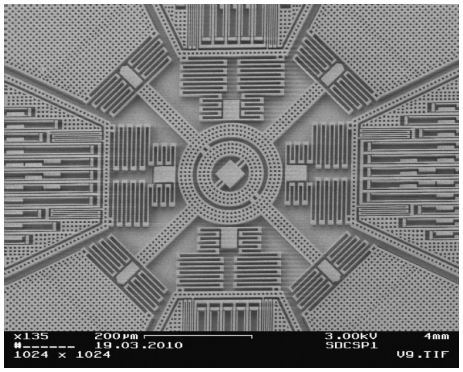


Figure 3.24 – *MEMS Gyroscope [28]*

Supply Voltage	2.4 to 3.6 V
Current Consumption	5 µA to 6.1 mA
Operating Temperature	-40^oC to 85^oC
Measurement range	$\pm 250 to \pm 2000$ dps (degrees per second)
Communication Interface	I^2C
Sensitivity	$250dps \rightarrow 8.75\frac{mdps}{digit}$ $500dps \rightarrow 17.5\frac{mdps}{digit}$ $2000dps \rightarrow 70\frac{mdps}{digit}$

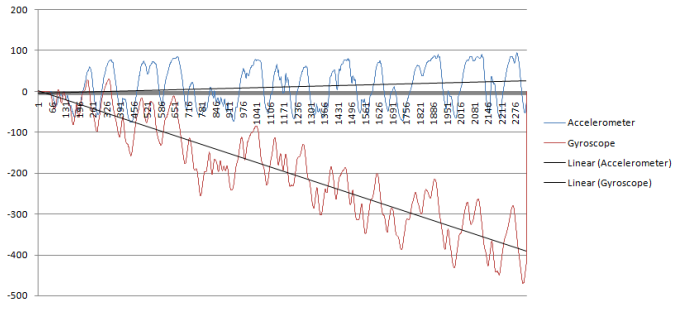
Table 3.5 – *L3GD20 Gyroscope Characteristics [47]*

The gyroscope obtains a signal which can be related to the angular velocity, which means that in order to get the output in radians an integration is needed. The sensor signal is measured over a period of 120 seconds, getting a mean of $-1.867\frac{rad}{s}$ and keeping it as a reference value which has to be subtracted to each measurement received.

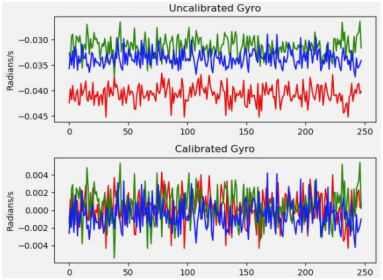
However, *it is subject to bias instabilities in which the initial zero reading of the gyroscope will cause drift over time due to integration of inherent imperfections and noise within the device* . This can be fixed with a complimentary filter (which is explained in Code 4.10)

3.2.3 Electrical Power System

Over the next section the electrical power components and their connections will be discussed.



(a) *Noise (acc) & Cumulative Error (gyro)*



(b) *Pre & Post Gyroscope Calibration*

Figure 3.25 – *Typical errors in gyroscopes and accelerometers [26], [4]*

3

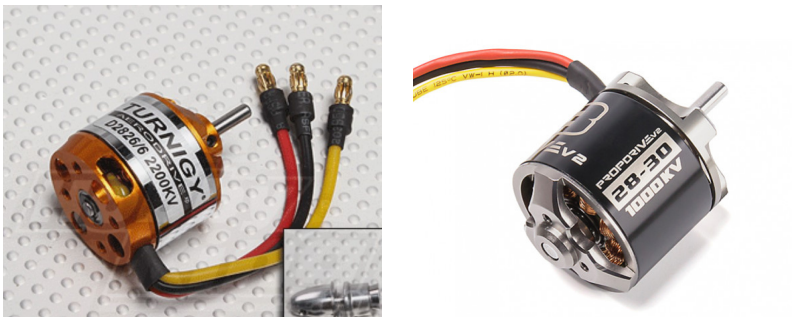
3.2.3.1 Power Budget

POWER BUDGET			MIN CURRENT CONS ()	TYP CURRENT CONS (mA)	MAX CURRENT CONS (mA)	MAX VOLTAGE (V)	POWER (mW)
MODULE	DEVICE	INFO					
uProcessor	ESP-32-WROOM	Max current delivered by power supply		500	500	3,6	1800
IMU	BMP180	Ultra low power mode		0,003		3,6	
		Standard mode		0,005			
		High resolution mode		0,007			
		Ultra high res. mode		0,012			
		Advanced res. mode		0,032			
		Peak current (conversion)			0,65		2,34
	L3GD20	Supply current		6,1	6,1	3,6	21,96
		Current sleep mode		2			
		Current power down		0,005			
	LSM303DLHC	Current normal mode		0,11		3,6	0,396
		Current sleep mode		0,001			
OLED	SSD1306	Max sink current			20	3,6	72
REFLECTIVE OPTICAL SENSOR	CNY70	Diode emitter current			20		
		BJT collector current			1		
		Total			21	3,6	75,6
INFRARED RECEIVER	KY-022	Current consumption	0,4		1,5	3,6	5,4
MAGNETIC BUZZER	KXG-1205	Max current			45	3,6	162
BUCK REGULATOR	KXG-1205	Lowest Efficiency (88%) with 100 mA Output Current			100	1,512	151,2
		Efficiency 1,512 V Buck Drain					
					BOARD(mA)		BOARD(mW)
					594,36		2139,696
							MOTOR(mW)
ESC DC MOTOR	ESC UBEC 30A	Max current for PWM of 7 %			1800	12,6	22680
							SYSTEM(mW)
							24819,696

3.2.3.2 Brushless DC Motors

The motors used in this project are three out runner brushless DC motors. These motors are widely used, from electrically propelled RC model aircraft to CD-ROM computer drives. The outrunners tend to have more poles, so they spin much slower than the inrunners, but the outrunner motor is capable of producing more torque (which is what we are after).

However, issues arose. As a client requirement, I had to use the D2826-6 model because it was the ones we had already bought. As commented before, we need acceleration changes, and this kind of motor starts up already in a very high speed.



(a) *Turnigy D2826-6* **(b)** *NTM Prop Drive Serie 28-30A*

Figure 3.26 – *Brushless Motors [14], [34]*

	Turnigy D2826-6 [14]	NTM Prop Drive [34]
RPM/V	2200	1000
Voltage (Cells)	2S ≈ 3S	3S-4s Lipoly
Max. Current (A)	34	25
Watts (W)	342	370
ESC (A)	40	25-30
Price (€)	10.54	18.05

Table 3.6 – *Brushless Motor Comparison*

The ESC chosen is presented in Figure 3.27, while its datasheet is presented in Figure 3.28

Figure 3.27 – *ESC 30A Driver [17]*

Standard	（A）sustained current	(10s)（A）max instantaneous current	(V/A) Output BEC	BEC mode	Number of battery/lithium battery	Number of battery/nickel-hydrogen battery	mm Size(width*length*height)	g weight
HK20A UBEC	20A	25A	5.5V/3A-1A	SBEC/UBEC	2-4Lipo	5-12NC	54*26*11	30
HK3A UBEC	30A	40A	5.5V/3A-1A	SBEC/UBEC	2-4Lipo	5-12NC	54*26*11	32

Figure 3.28 – *ESC 30A Datasheet [17]*

3.2.3.3 Power Source

For the purpose of making a portable SP we choose a battery as a power source. LiPo batteries, more specifically, offer a considerable energy density, making them ideal for portable purposes.

In our case is mandatory for the battery to be capable of providing the motors the current they need, as our brushless motors consume a high current.

(a) *Turnigy* **(b)** *Rhino*

Figure 3.29 – *Battery Comparison*

	Turnigy 2200mAh 3S 40C	Rhino 2200mAh 4S 50C
Capacity (mAh)	2200	2200
Configuration	3S1P-11.1 V-3 Cell	4S1P-14.8 V- 4 Cell
Constant Discharge	40C	50C
Max. Discharge (10 sec)	50C	-
Price (€)	14.59	16.20
Dimensions & Weight	104x27x35 mm & 204g	107x35x35 mm & 245g

Table 3.7 – *Battery LiPo Comparison*

Even though the Rhino model has a better current discharge, the weight and the price make the Turnigy a better option. As for the current discharge ratio, we have that

$$I_{ConstantDischargeTurnigy} = 40C \times 2.2\frac{A}{h} = 88.8\text{Adischargerate}$$

The three motors would need up to 102 A in case we reach the maximum speed, but as we only need acceleration in order to control our platform, it is not necessary to design the power source for that much current. Limiting the speed by software, defining limits for the PWM is enough to control the current drain.

3.2.3.4 Power Regulation

3.2.3.4.1 Buck Converters

Choosing a buck converter comprehends a lot of variables to have in mind. Maximum output current, ripple current due to switching frequency, etc. Also, price and availability could be decisive.

	LMR33610ADDAR [43]	BD9E104FJ-E2 [42]
Input Voltage Range	3.8 to 36 V	7.0 to 26.0 V
Output Voltage Range	1 to 24 V	1.0 V to $V_{in} \times 0.5$
Switching Frequency	1400 kHz	570 kHz
Min. Efficiency	75%	89 %
Max Junction Temperature	150°C	150°C

Table 3.8 – *Comparison Buck Characteristics*

The BD9E104FJ-E2 was chosen because of its low price and availability, and also

because of the SLLM mode present. While the LMR33610ADDAR has better frequency switching capabilities, when the load current trespasses the 200 mA threshold, PWM configuration changes to SLLM [3] in the BD9. However, the LMR336 changes to PFM, decreasing efficiency about 10%.

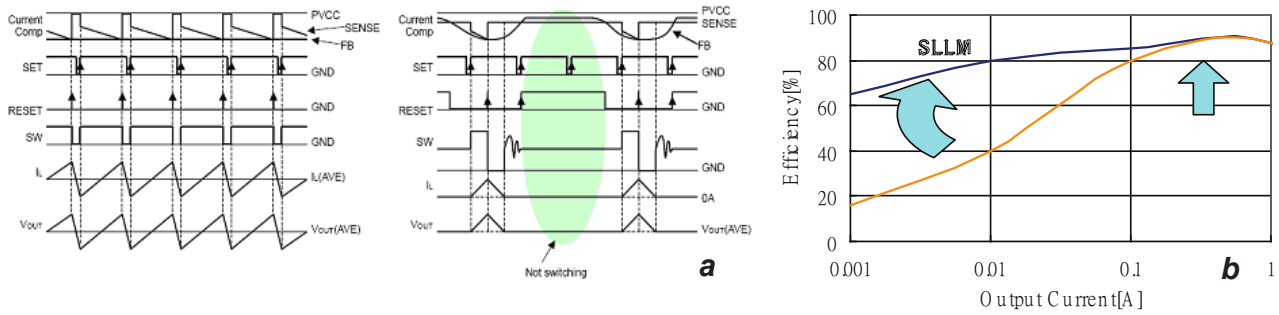
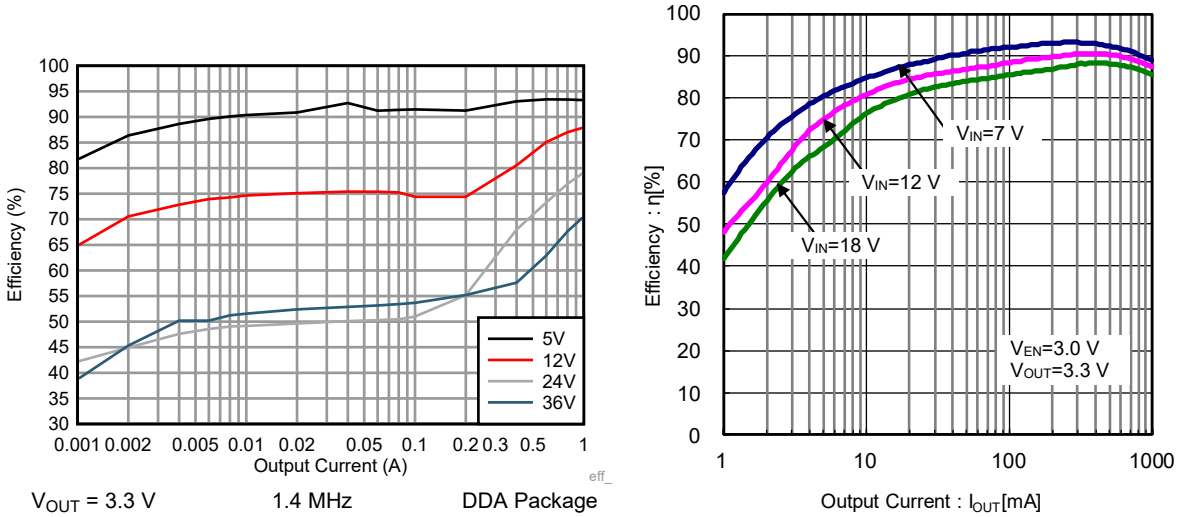


Figure 3.30 – *SLLM Mode [3]*

Under higher load levels, normal current mode PWM control is utilized. When a light load is detected, the switching pulse (SW) turns OFF the normal PWM control loop allowing linear operation without an excessive voltage drop or deterioration in transient response during the switching from light load to heavy load or vice versa.

The former sentence comes from an ROHM article, showing that their technology is even 200% more efficient than PFM technology. We see that it is true when comparing the efficiency tables from each buck:



(a) *LMR33610 Efficiency* **(b)** *BD9E Efficiency*

Figure 3.31 – *Buck Efficiency Comparison*

Then, choosing the BD9E104FJ-E2 buck is the best option. Recommended values present in the datasheet [42] were used to obtain the desired output voltage.

3.2.3.4.2 Current Measurement

We will use the INA219 microchip in order to determine the current measure, important for the well being of our system and the battery health. Strong currents will appear due to the brushless motors, we have to make sure they do not damage our SP.

V_{SHUNT}	$\pm$320mV
Power Supply	3 to 5.5 V
IN+ Pin Input Bias Current	20 µA
ADC Basic Resolution	12 bits
ADC Conversion Time	-30^oC to 70^oC

Table 3.9 – *INA219 Characteristics [44]*

If we wanted to measure the current [30], the simplest way would be

$$RealShuntVoltage = ShuntVoltageRegister * 10e-6$$

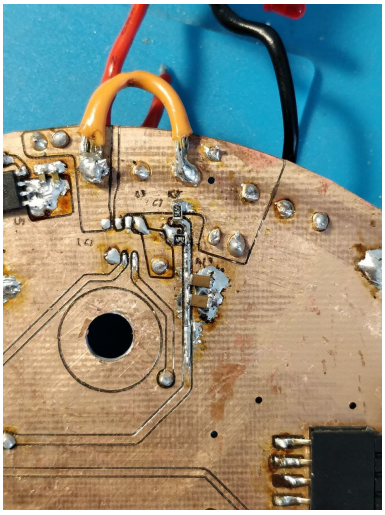
$$Current = \frac{V}{R} = \frac{ShuntVoltageRegister * 10e-6}{Rshunt} \rightarrow Current = (ShuntVoltageRegister * 1e-4)$$

Where the 1e-4 value is defining the decimal point four places to the left.

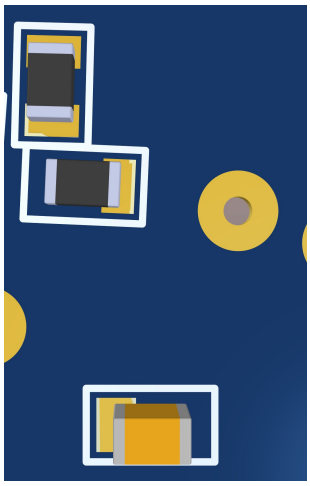
3.2.3.5 Energy Storage

3.2.3.5.1 Battery Voltage Measurement

Making a voltage divider between the output battery voltage and the power supply of our system (board and motors) allows us to determine the battery left.



(a) *Filter PCB Welding*



(b) *Detailed View of the Filter*

Figure 3.32 – *Resistors and Capacitor of the RC Filter*

We had to face an issue when designing this; choosing resistors between $10k\Omega$ to $60k\Omega$ meant that the thermal noise (Equation Figure 3.2.3.5.1) would be low, but the draining current could turn considerably high (as the system must be connected at all times when working). On the other side, choosing high values (between $1M\Omega$ to $10M\Omega$) would dramatically increase thermal noise, while draining a minimum amount of current.

$$v_n = \sqrt{4K_BTR\Delta f} \Rightarrow \begin{cases} v_n & \text{Thermal Noise, } [Voltage] \\ K_B & \text{Boltzmann's Constant, } 1.380649 \times 10^{-23} \left[\frac{Joules}{Kelvin}\right] \\ T & \text{Temperature, } 298K[Kelvin] \\ R & \text{Resistance, } [\Omega] \\ \Delta f & \text{Bandwidth Frequency of the noise, } [Hertz] \end{cases}$$

The variables which only depend of our design are Δf and of course the resistor, as the temperature varies depending on the place (we will consider room temperature by default). As we want to increase the resistor, the only possible option is to **reduce the bandwidth** of the noise. We can achieve that with a **low pass filter**.

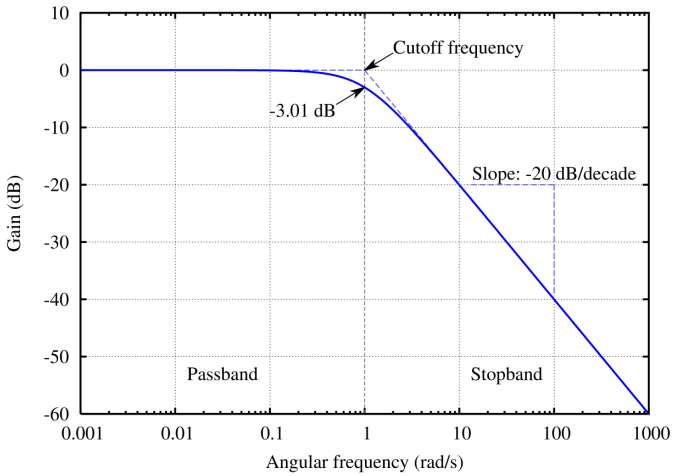


Figure 3.33 – *Low Pass Filter [1]*

3

A capacitor in parallel with R_2 (the resistor between the ADC pin and GND) would create a very simple low pass filter with a fall of $-20dB/dec$, adequate for our system (Figure 3.34).

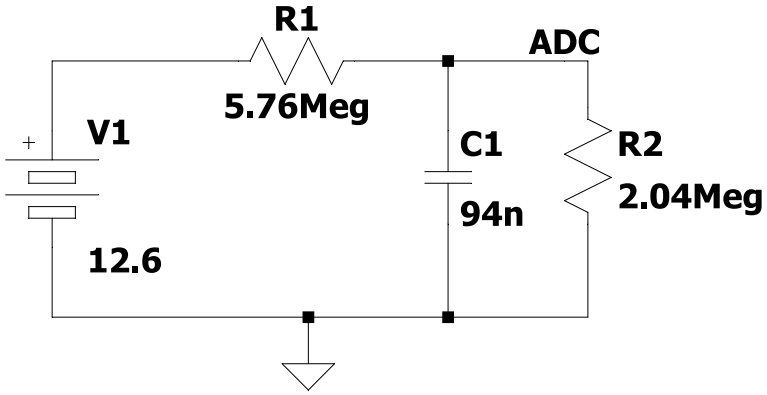


Figure 3.34 – *RC Filter Schematic*

The following equation determines the filter behaviour through a voltage gain form

$$\frac{V_{BAT}-V_{ADC}}{R_1}=\frac{V_{ADC}}{R_2||C}\Rightarrow\frac{V_{ADC}(s)}{V_{BAT}(s)}\equiv\frac{V_0(s)}{V_i(s)}=\frac{R_2}{R_2+R_1}\left(1+sC\frac{R_2R_1}{R_2+R_1}\right)^{-1}.$$

For the resistors design we could stipulate $s=0$

$$\frac{R_2}{R_2+R_1}=\frac{3.3V}{12.6V}\xrightarrow[resistors]{in-stock}\begin{cases}R_1=5.76M\Omega\\R_2=2.04M\Omega\end{cases}\tag{3.2.3}$$

Now, choosing a **very low cut-off frequency $\mathbf{w_c}$** with capacitors and the former resistors (Equation 3.2.3) present in the laboratory, we claim

$$\frac{V_{ADC}(s)}{V_{BAT}(s)} \xrightarrow[s=jw]{} \frac{R_2}{R_2+R_1}\left(1+jwC\frac{R_2R_1}{R_2+R_1}\right)^{-1} \Rightarrow$$
$$\boldsymbol{w_c} = \frac{R_2+R_1}{CR_2R_1} = \frac{2.04+5.76}{C\cdot 2.04\cdot 5.76} \approx 1-10\ \ Hz\cdot$$

As we only have ceramic capacitors (high frequency limiters) under **47nF**, we could put two of them in parallel to achieve one capacitor of **94nF**

$$w_c = \frac{2.04+5.76}{94\cdot 2.04\cdot 5.76} \approx 7.167\ \ Hz \quad \rightarrow \quad \boldsymbol{\Delta f \downarrow} \quad \rightarrow \quad \boldsymbol{v_n \downarrow} \cdot \tag{3.2.4}$$

3

Chapter 4

System Design

4.1 Mechanical Platform

4.1.1 Air Bearing

4.1.1.1 Design & Mechanical Characterization

The Air Bearing Design was conducted by Ángel Paredes Parrilla, a former GranaSAT intern whose project was designing and manufacturing an Air Bearing system (Figure 4.1) made of aluminum. This project was based under a Master's Thesis from Jorge Prado [45].

(a) *Semisphere Milling* **(b)** *Defining Semisphere Circular Profile* **(c)** *Cup Holder Milling*

Figure 4.1 – *Manufacturing of Air Bearing system by Ángel Paredes Parrilla*

A rendered image from the design can be seen next (Figure 5.1):

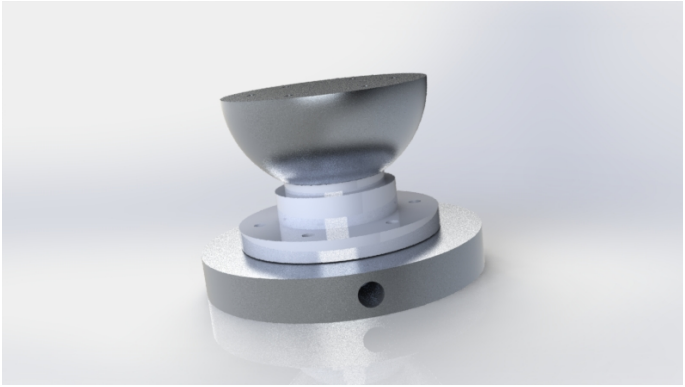
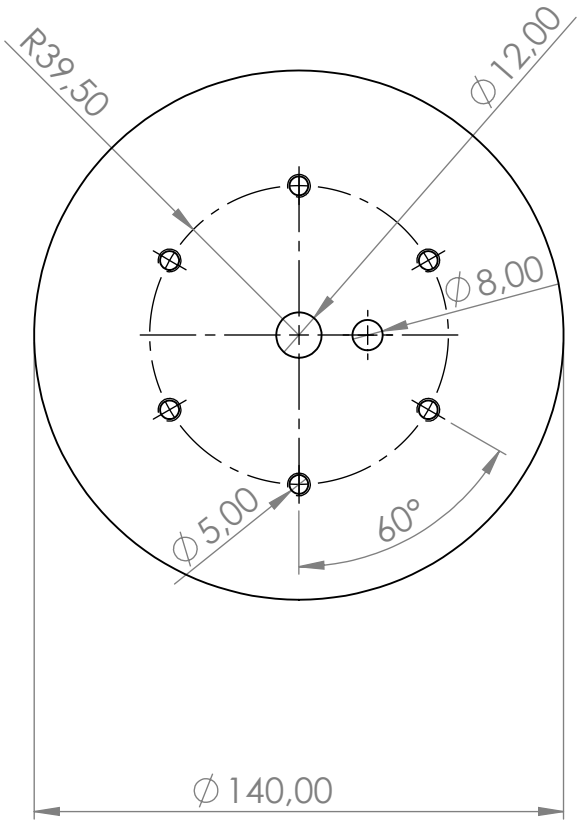
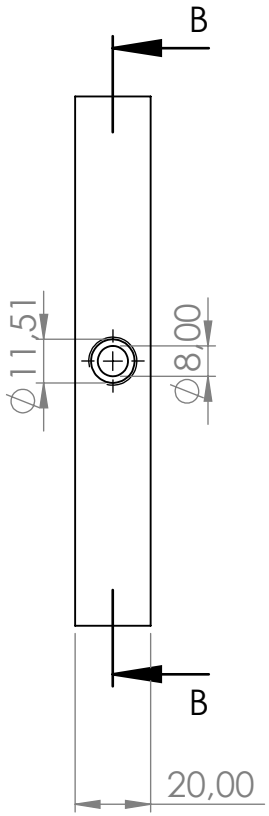
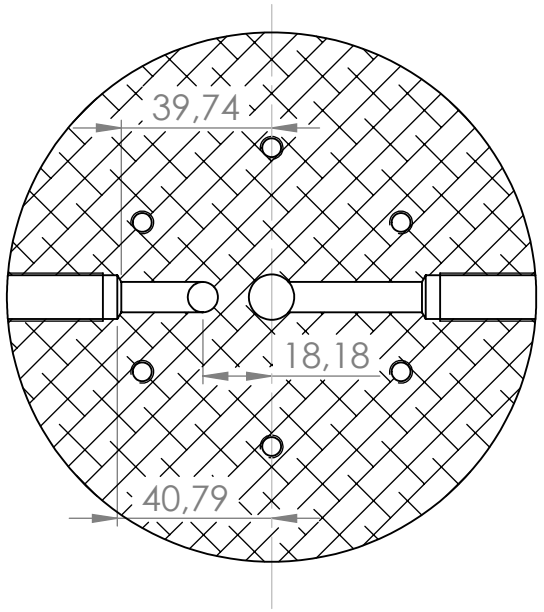


Figure 4.2 – *Render from the sphere and the pneumatic bearing system in SolidWorks*

Over the next pages the drawing with dimensions of the Air Bearing system will be displayed.

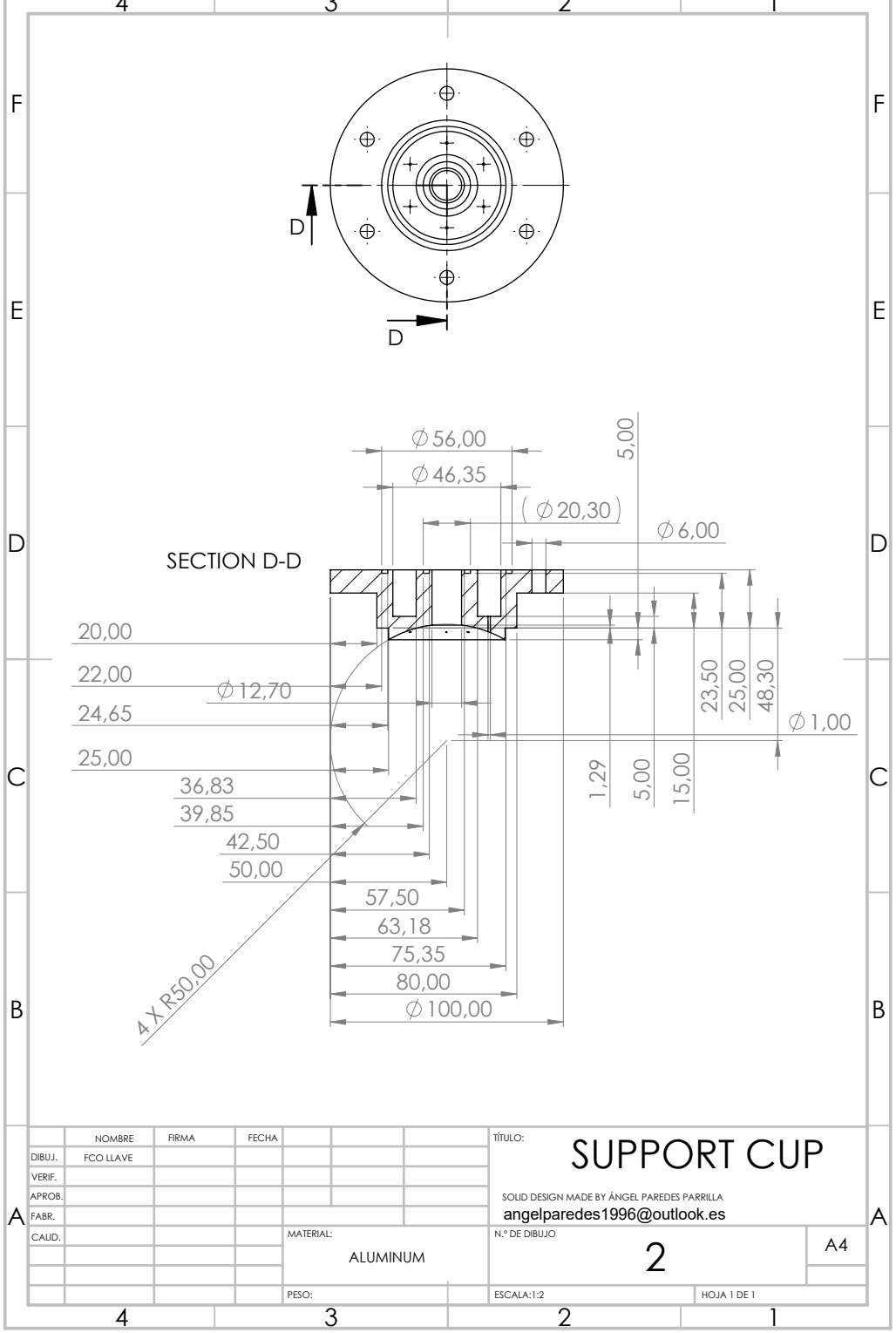




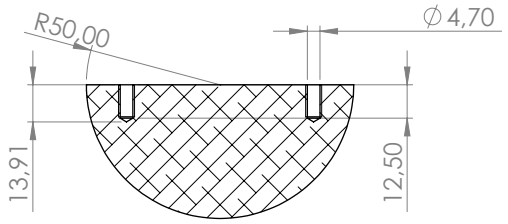


SECTION B-B

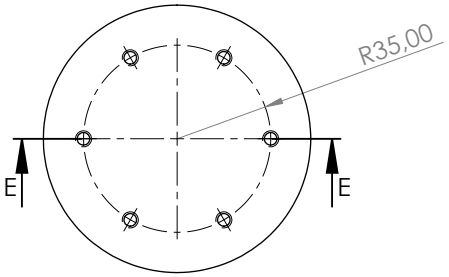
	NOMBRE	FIRMA	FECHA				TÍTULO:
DIBUJ.	FCO LLAVE		12/06/2021				AIR BASE
VERIF.							SOLID DESIGN MADE BY ÁNGEL PAREDES PARRILLA
APROB.							angelparedes1996@outlook.es
FABR.							
CALID.				MATERIAL:			N.º DE DIBUJO
				ALUMINUM			1
							A4
				PESO:		ESCALA:1:2	HOJA 1 DE 1



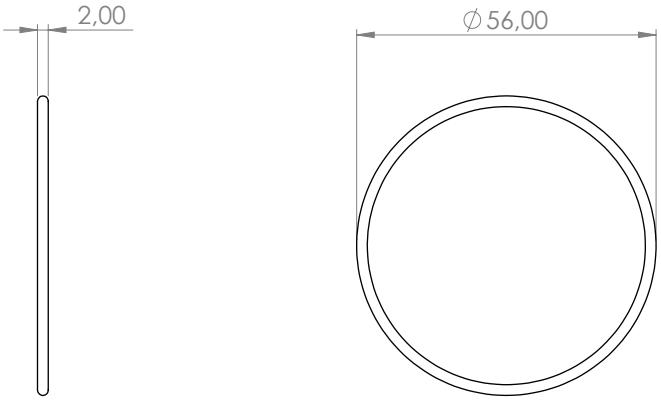
	NOMBRE	FIRMA	FECHA				TÍTULO:		
DIBUJ.	FCO LLAVE						SUPPORT CUP		
VERIF.									
APROB.							SOLID DESIGN MADE BY ÁNGEL PAREDES PARRILLA		
FABR.							angelparedes1996@outlook.es		
CALID.				MATERIAL:			N.º DE DIBUJO		A4
				ALUMINUM			2		
				PESO:			ESCALA:1:2	HOJA 1 DE 1	



SECTION E-E



SEMISPHERE



O-RING

	NOMBRE	FIRMA	FECHA				TÍTULO: SPHERE & O-RING	
DIBUJ.	FCO LLAVE						SOLID DESIGN MADE BY ÁNGEL PAREDES PARRILLA angelparedes1996@outlook.es	
VERIF.								
APROB.								
FABR.								
CALID.				MATERIAL: ALUMINUM			N.º DE DIBUJO 3	A4
				PESO:			ESCALA:1:2	HOJA 1 DE 1

4.1.1.2 Manufacturing

I will explain the creation and assembly process carried on by me to build the necessary SP:

- For an instance, we need to guarantee the tower stabilization (Figure 4.3 a) but also due to the air compressor (Figure 4.3 b) weight (around 8 Kg), we need to provide the mounting with a transport method. The availability of recycled office chair wheels in the lab allowed us to implement a wheeled support table. Brakes were not needed as the weight of the compressor was enough.

(a) *Tower Support for the Simulation Platform*

(b) *Tower with Air Compressor*

Figure 4.3 – *Moving Mounting for SP*

- Once the mounting was operative, we could proceed with the positioning of a wooden platform (Figure 4.4) made in the laser cut machine (Figure 4.5 c) .

Figure 4.4 – *Wooden Platform for SP*

- The machines used to craft all of the former elements appear in (Figure 4.5)

(a) *Drill Press*

(b) *Laser Cut Machine*

(c) *Sanding Machine*

Figure 4.5 – *Machines Used during the fabrication process*

4.1.2 Reaction Wheel

Reaction wheels design should be focused on stability and an anti-vibrational approach. For this, the following prototypes were designed.

4.1.2.1 Design & Mechanical Characterization

This design is based on the Dimitri Gruebel's [5] design for a platform like ours [6].

4.1.2.1.1 Gripping Arms GA

Over the next images (Figure 4.6, Figure 4.7) are the render of the gripping arms design in solidworks. They are the ones that will be in charge of linking the IDs with the SP, in order to transmit the torque for balancing and rotating it.

The gripping arm for the Z axis is exposed in Figure 4.7, where the torque will be headed **upwards**

In this link there is a video showing the twisting direction determination of the ID, so we can be sure of the torque direction.

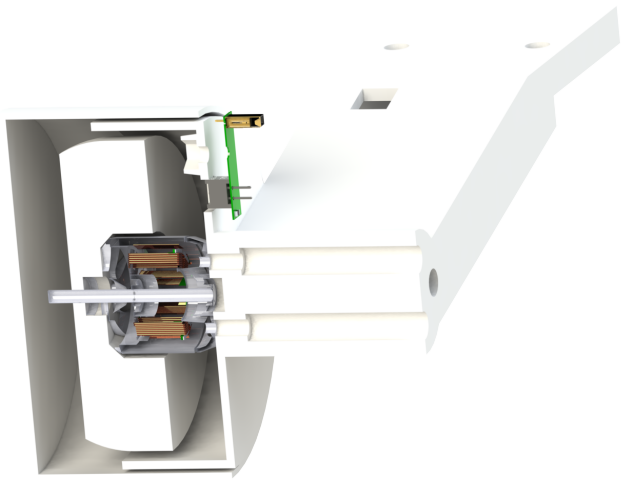


Figure 4.6 – *Gripping Arm for X & Y axis*

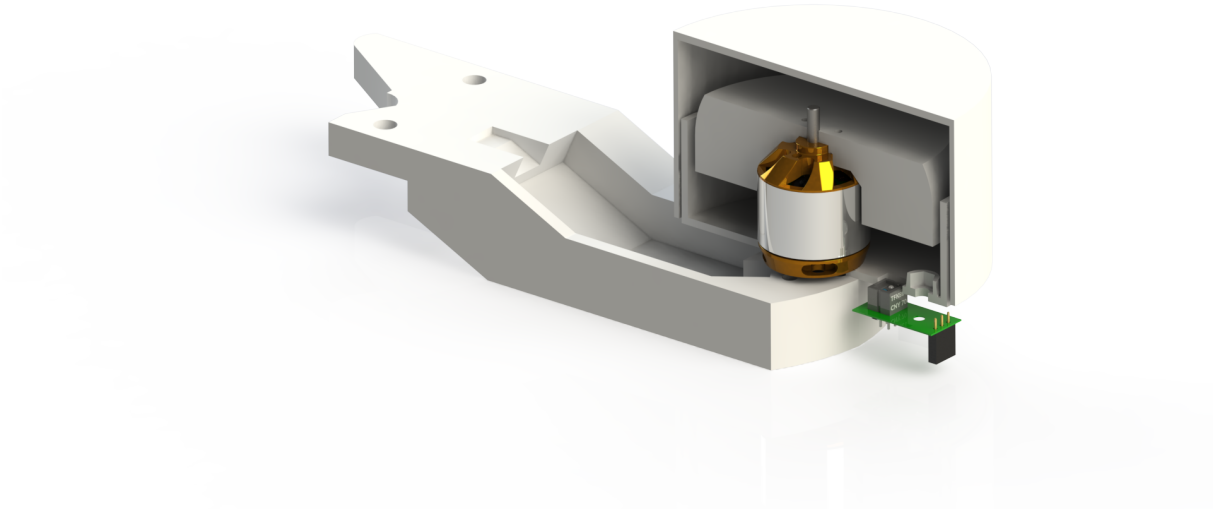
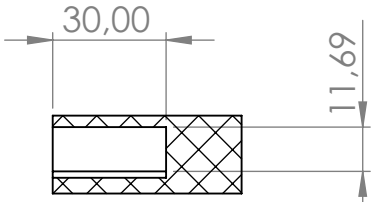


Figure 4.7 – *Gripping Arm for Z axis*

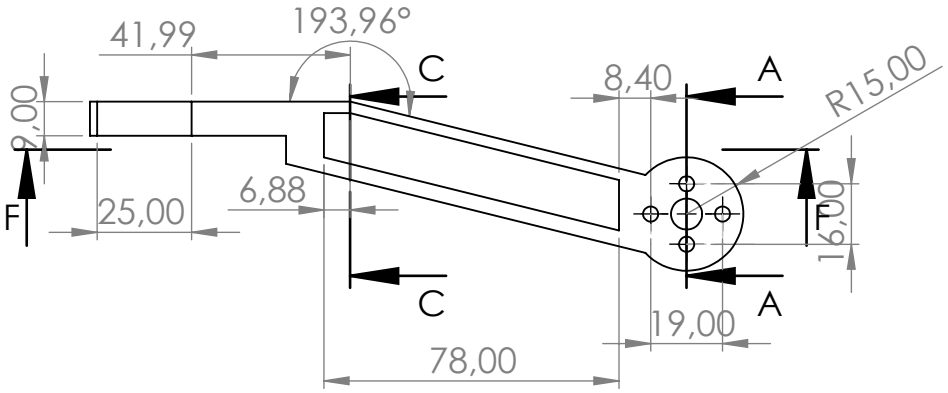
4.1.2.1.2 ID

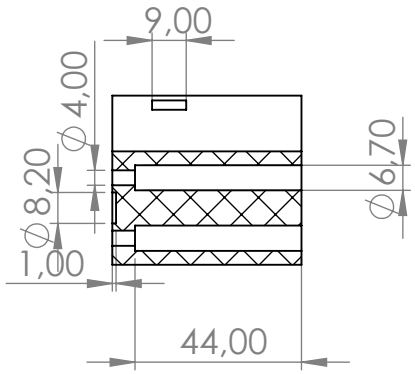
ID design was decided to be like that after a few trials with former designs made by other people.

At first, the design proposed by my mentor Andrés Roldán was the one in Figure 4.8 a. However, this design had a major problem. These IDs can experience speeds of 18000 RPM, where the grip system promoted by the motor company was not suitable for us. Thinking of a solution I discovered three holes for a Metric 2 screw, which gave me the idea to screw the ID to the motor (Figure 4.8 b). This idea was essential, as I have had a lot of trouble with the ID coming out of the motor.

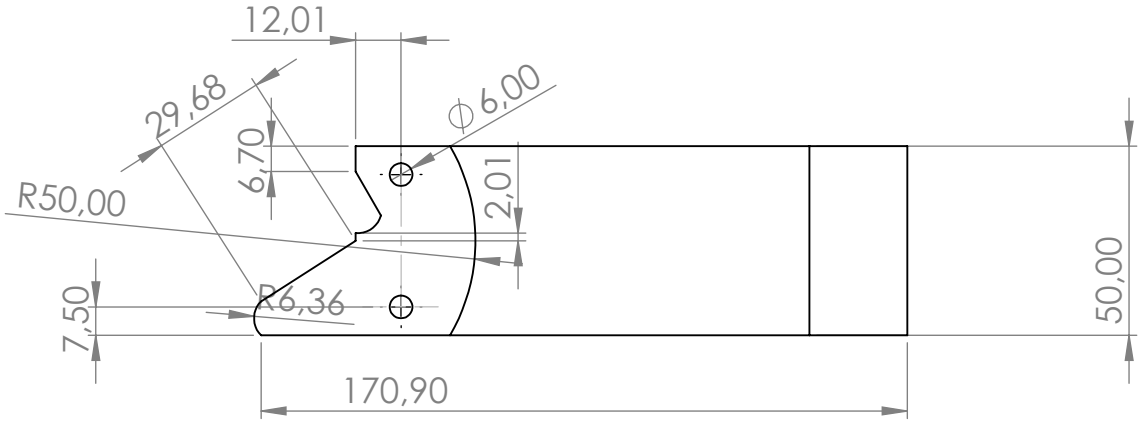


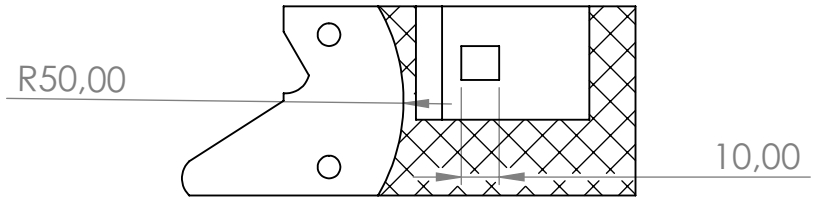
SECTION C-C





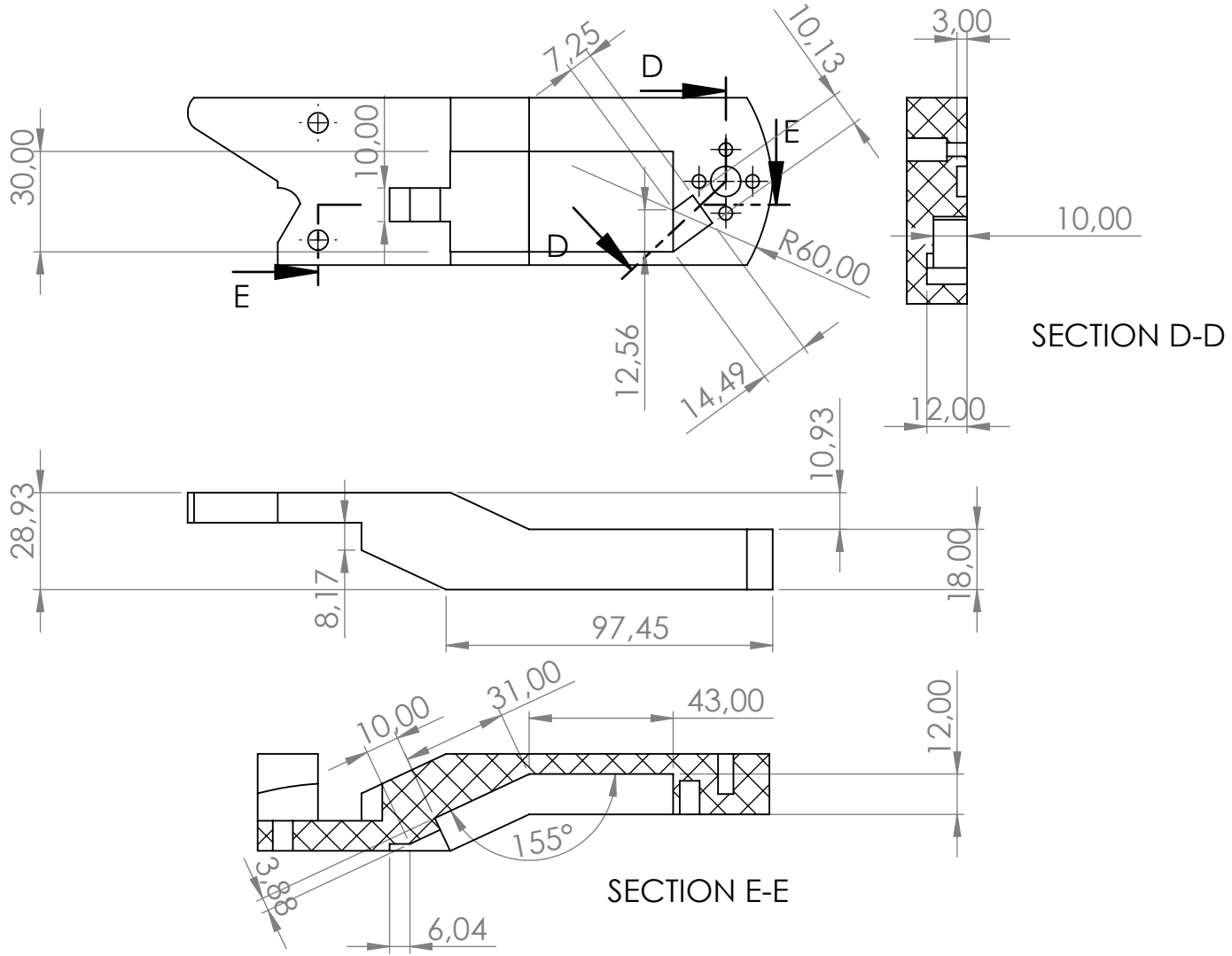
SECTION A-A





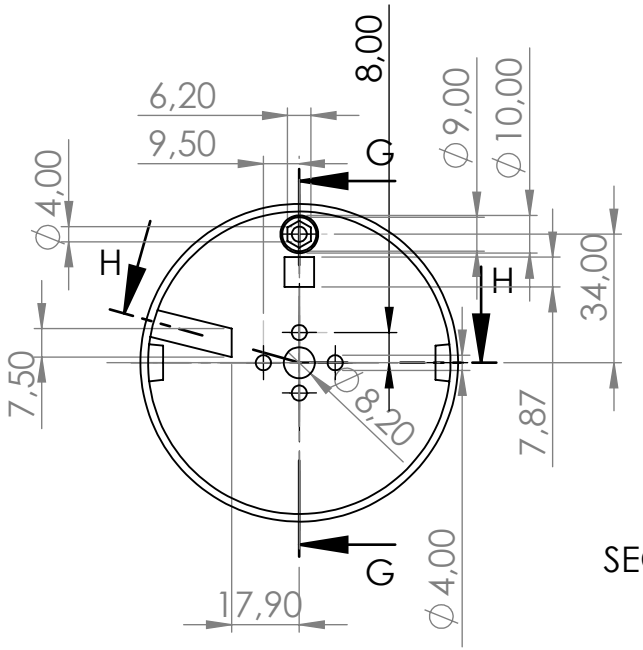
SECTION F-F

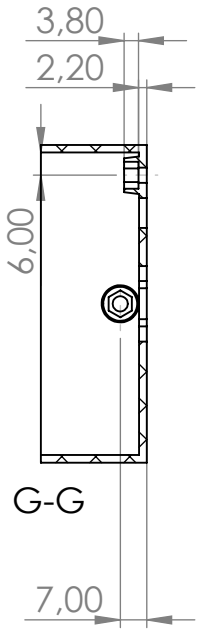
	NOMBRE	FIRMA	FECHA				TÍTULO: ARM FOR X & Y AXIS
DIBUJ.	FCO LLAVE						SOLID DESIGN MADE BY FRANCISCO LLAVE IGLESIAS francisco.llaveiglesias@gmail.com
VERIF.							
APROB.							
FABR.							
CALID.				MATERIAL: PLA PLASTIC			N.º DE DIBUJO 1 — A4
				PESO:			ESCALA:1:2 — HOJA 1 DE 1



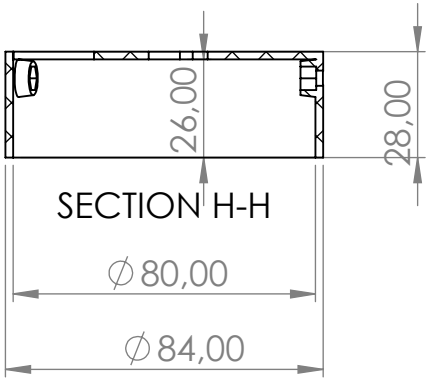
	NOMBRE	FIRMA	FECHA				TÍTULO: ARM FOR Z AXIS
DIBUJ.	FCO LLAVE						SOLID DESIGN MADE BY FRANCISCO LLAVE IGLESIAS francisco.llaveiglesias@gmail.com
VERIF.							
APROB.							
FABR.							
CALID.				MATERIAL: PLA PLASTIC			N.º DE DIBUJO 2 — A4
				PESO:			ESCALA:1:2 — HOJA 1 DE 1

PROTECTIVE LID FOR X AND Y AXIS





SECTION G-G

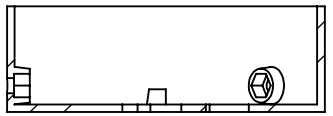


SECTION H-H

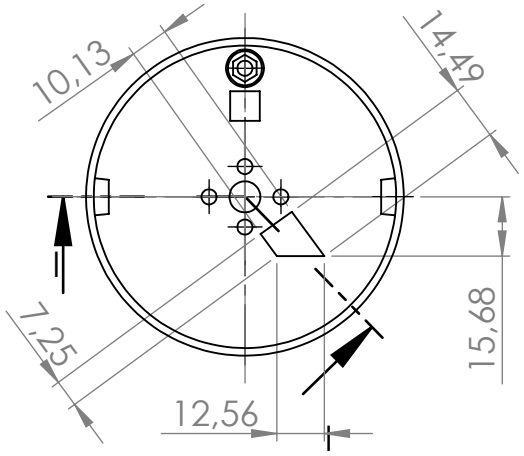
⌀80,00

⌀84,00

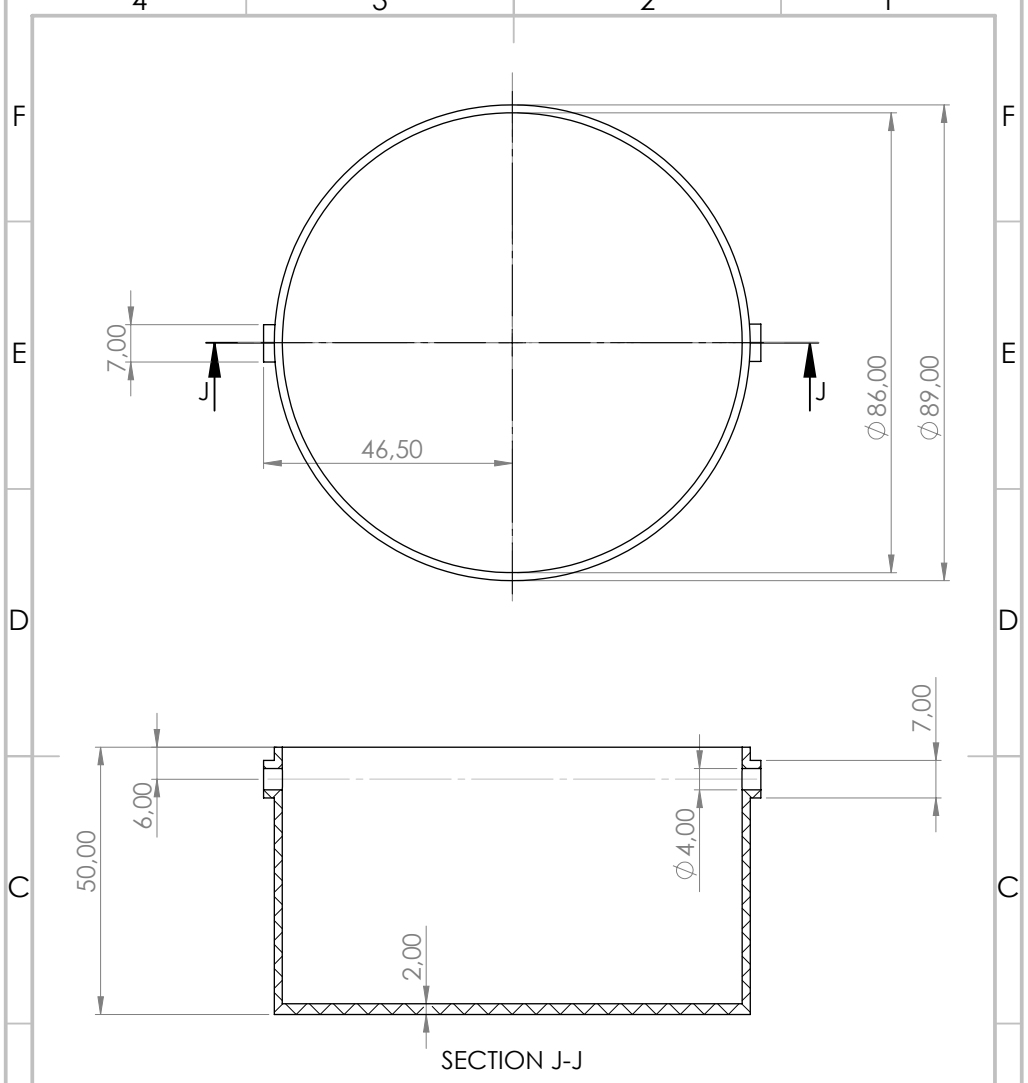
PROTECTIVE LID FOR Z AXIS



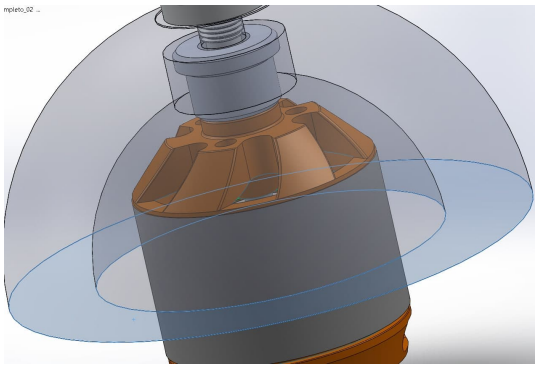
SECTION I-I



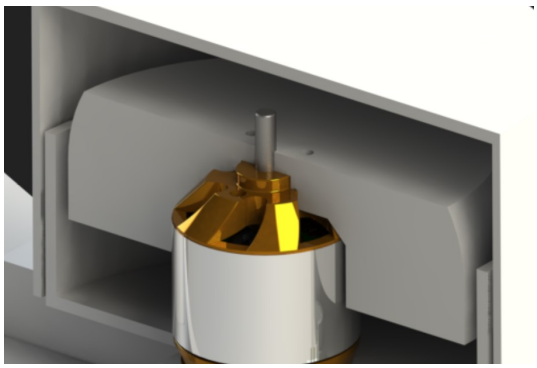
	NOMBRE	FIRMA	FECHA				TÍTULO: PROTECTIVE LIDS
DIBUJ.	FCO LLAVE						SOLID DESIGN MADE BY FRANCISCO LLAVE IGLESIAS francisco.llaveiglesias@gmail.com
VERIF.							
APROB.							
FABR.							
CALID.				MATERIAL: PLA PLASTIC			N.º DE DIBUJO 3 — A4
				PESO:			ESCALA:1:2 — HOJA 1 DE 1



	NOMBRE	FIRMA	FECHA				TÍTULO:	
DIBUJ.	FCO LLAVE						PROTECTIVE LIDS	
VERIF.							SOLID DESIGN MADE BY FRANCISCO LLAVE IGLESIAS	
APROB.							francisco.llaveiglesias@gmail.com	
FABR.								
CALID.				MATERIAL:			N.º DE DIBUJO	A4
				PLA PLASTIC			3	
				PESO:			ESCALA:1:2	HOJA 1 DE 1



(a) *First Own Prototype for ID*



(b) *Final ID Prototype*

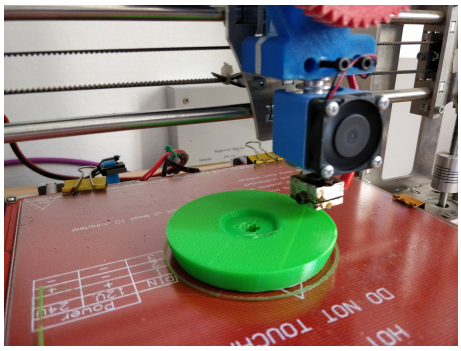
Figure 4.8 – *ID Prototypes*

4.1.2.2 Manufacturing

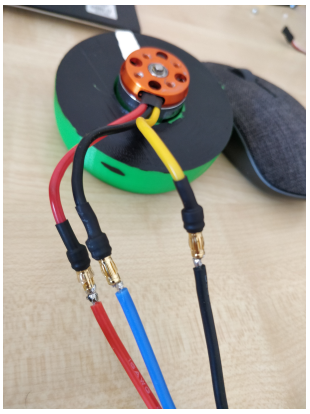
The manufacturing of the gripping arms was made with the GranaSAT 3D printer. As I spent a lot of time familiarizing with this tool, I decided to include in Appendix B how to properly calibrate the printer. This is something that is advisable to do after every printed component in order to avoid wrong future printings, wasting time and plastic.

The plastic used for this printer is PLA with 1.75 mm.

- The ID design (explained in Subsection 4.1.2) was made to fit perfectly with the brushless motor. The black paint with a white mark on it is for the tachometer. A green surface is very reflective, so we had to fix it if we wanted the emitting diode to work properly.



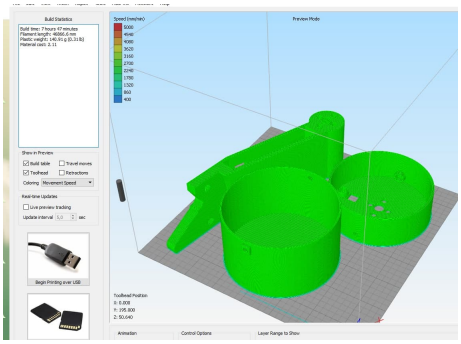
(a) *ID Printing*



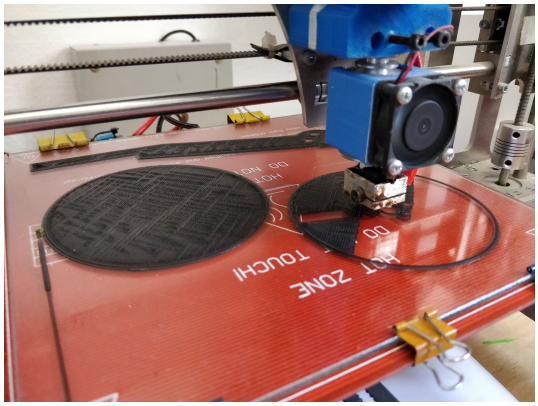
(b) *ID just painted*

Figure 4.9 – *ID Manufacturing*

- The next images show a screenshot from Simplify 3D ©



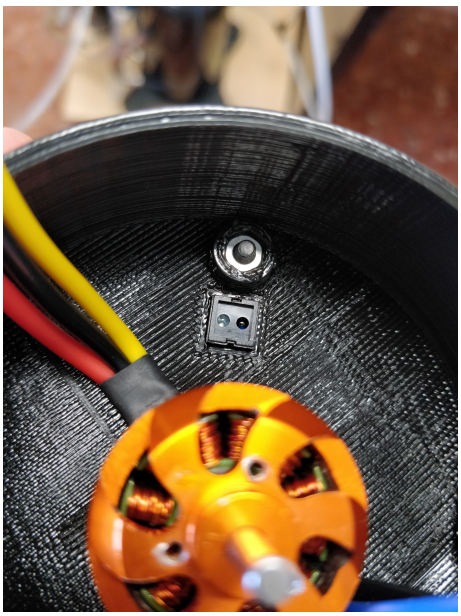
(a) *Simplify Screenshot*



(b) *GA Printing*

Figure 4.10 – *GA Manufacturing*

- This design comprehends the tachometer implementation and a security system, after the high speeds reached by the ID in the test were considered dangerous for the user. The nuts for linking the lid to the security system and the tachometer PCB were embedded into the plastic using a soldering iron.

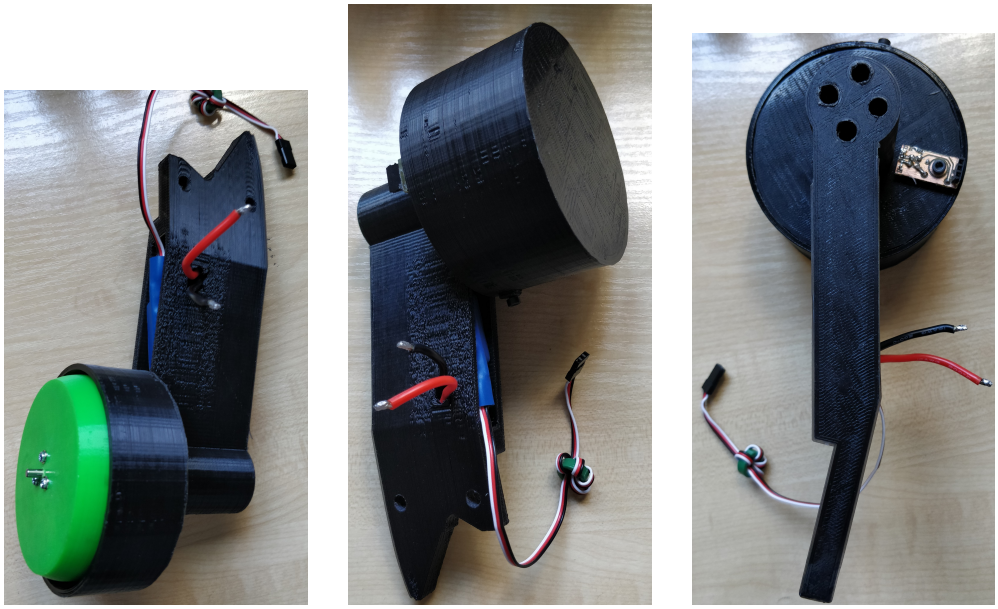


(a) *Tachometer and its nut Detail*

(b) *Wires welded*

Figure 4.11 – *Details of the ID Security System*

- As a result, the GA with the ID assembled looks like this:



(a) *GA without the Security Lid*

(b) *Front GA View*

(c) *Back GA View with Tachometer PCB*

Figure 4.12 – *Final GA Prototype*

4

In the next figure we see the printer used:

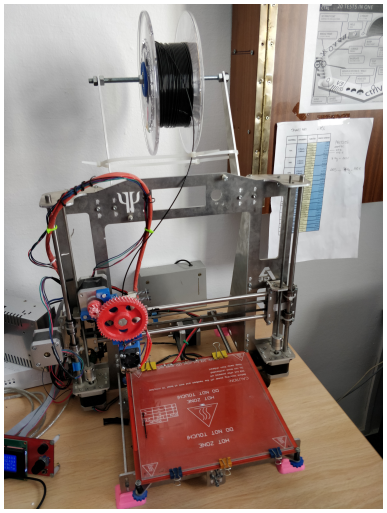


Figure 4.13 – *GranaSAT Prusa 3D Printer*

4.1.3 Simulation System

4.1.3.1 CAD Assembly

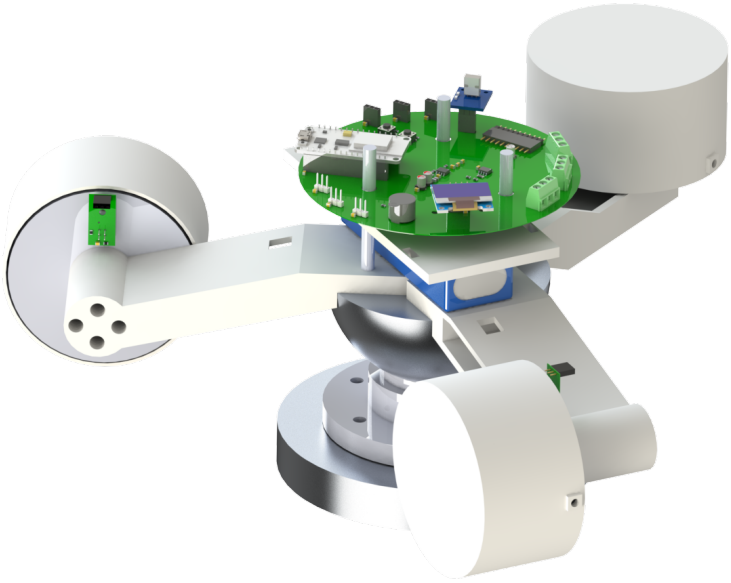


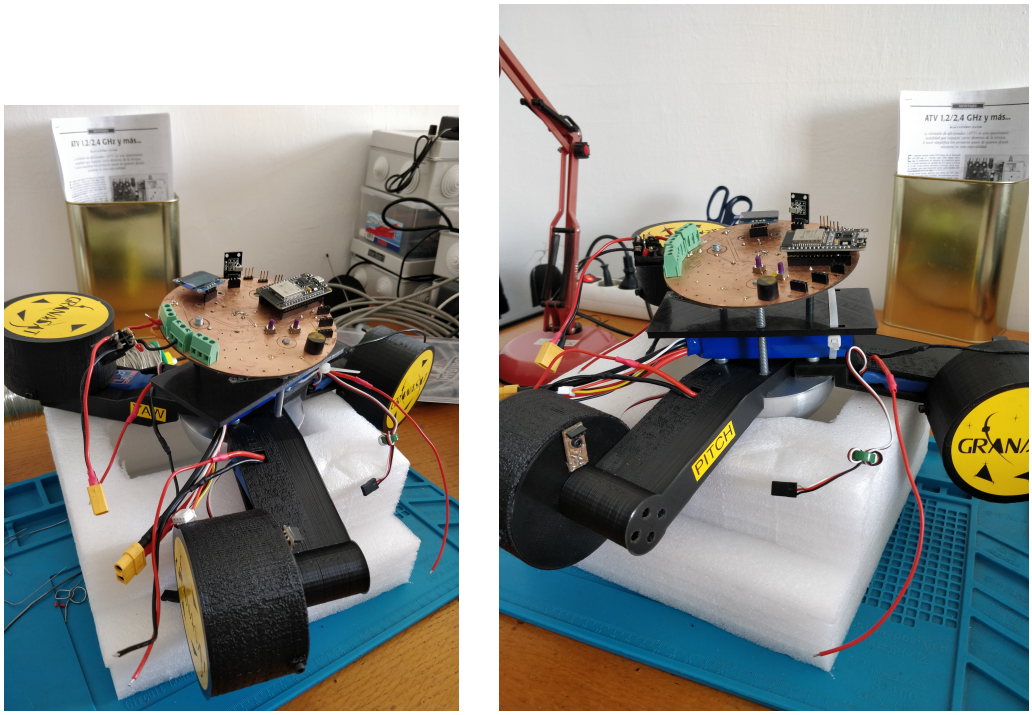
Figure 4.14 – *Render from the sphere and the pneumatic bearing system in SolidWorks®*

4.1.3.2 Prototype Assembly

4.2 Control Unit

4.2.1 Central Processing Unit

The central processing unit will be the ESP32, our microchip which gathers all of the necessary capabilities for the proper functioning of our system.



(a) *First View* **(b)** *Second View*

Figure 4.15 – *Physical Appearance of the whole Integrated System*

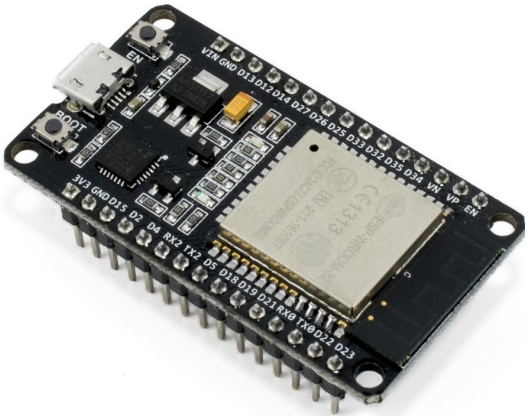


Figure 4.16 – *Node MCU ESP32 WROVER [19]*

The characteristics will be widely explained in the following sections, but a technical approach of its datasheet is displayed here:

Module model	ESP-WROOM-32s
Size	25.4*48.26*3mm(±0.2mm)
Certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC/ BQB/ RoHS/REACH
SPI Flash	32Mbit(default)
Support interface	UART/GPIO/ADC/DAC/SDIO/SD card /PWM/I2C/I2S
Integrated crystal oscillator	40MHz Crystal oscillator
IO Port	38
Antenna	Onboard antenna
Power Supply	Voltage 3.0V ~ 3.6V, Typical 3.3V, Current >500mA
Operating Temperature	-40 ℃ ~ 85 ℃
Storage Environment	-40 ℃ ~ 120 ℃

Figure 4.17 – *Node MCU Datasheet*

4.2.2 Communications

4.2.2.1 Wire

Wired communication can be made with the Serial command from C. It is advisable for local configurations. For the ESP32 this must be made with a rate of 115200.

```
1       Serial.begin(115200);
2
```

Code 4.1 – *Serial Communications*

4.2.2.2 Wireless

For this communication we need to define WiFi capabilities in our ESP32. It is important to keep in mind that due to overcurrent draining protection, all of the ADC2 pins from the Node MCU can not be used while using WiFi [39]. This was not a problem for us because we had plenty of pins, but for other projects could be an irrevocable drawback.

```
1       const char* ssid = "network";
2 const char* password = "password";
3 ...
4       WiFi.begin(ssid, password);
5   while (WiFi.status() != WL_CONNECTED) {
6     delay(1000);
7     Serial.println("Connecting to WiFi..");
8   }
9
```

Code 4.2 – *WiFi Setup*

As you can see, the configuration is pretty simple. After this, we will contemplate the interface creation. This process is more difficult.

4.2.2.3 HTML Interface

Creating an HTML interface requires establishing an asynchronous communication between the Control Unit and the monitoring interface. Knowledge in CSS, HTML and JavaScript had to be learnt in order to modify this program from Random Nerds Tutorials [18] . A screenshot from the iPad can be seen next (Figure 4.18)

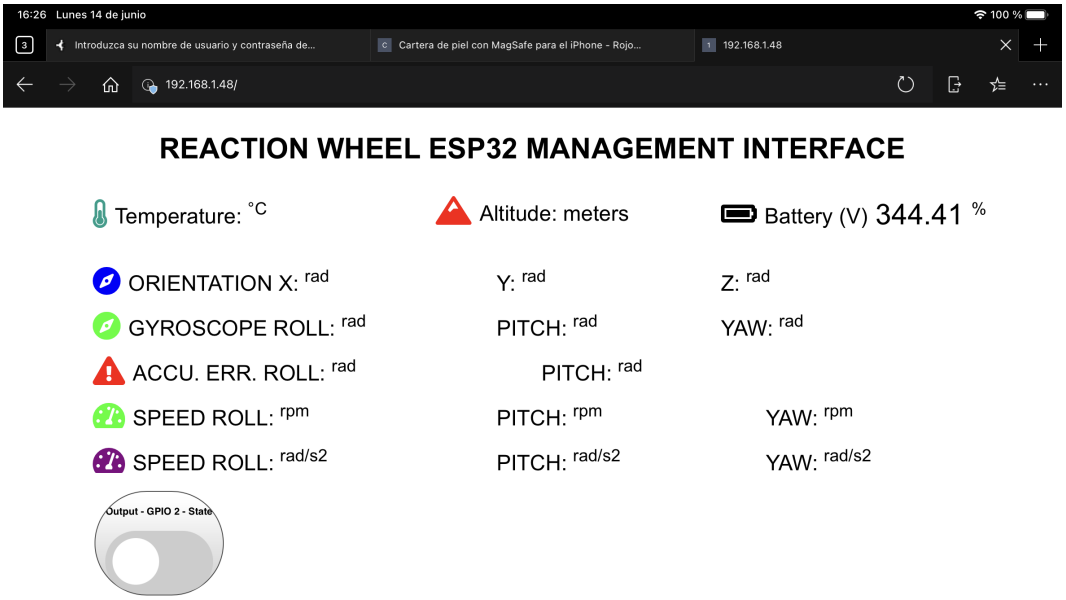


Figure 4.18 – *HTML Interface*

The code is quite long, so we will differentiate three steps:

- Graphic Interface Design

  The design of the Interface is made with CSS. For the icons we used Fontawesome Icons [**?**].

```
1                imu_fusion_x { position:absolute; left:100px;
top:175px;}
```

```
                ...
                <p><imu_fusion_x>
                <i class="fas fa-compass" style="color:#0000ff;
"></i>
                <span class="dht-labels">ORIENTATION X: </span>
                <span id="fusion_x">%FUSION_X%</span>
                <sup class="units">rad</sup>
              </imu_fusion_x></p>

```

Code 4.3 – *Data Reading Variable definition*

In Code 4.3 we can see an example of how to create a simple data variable to show the data from the IMU, more specifically the orientation fusion value of x.

- Interface Variable Behaviour

  In order to show and upgrade this value each time it changes in the CU, we need to establish through JavaScript how often we want to pull the value from the CU. This is called request, and we make it every 100 milliseconds.

```
                setInterval(function ( ) {
                  var xhttp = new XMLHttpRequest();
                  xhttp.onreadystatechange = function() {
                    if (this.readyState == 4 && this.status ==
200) {
                      document.getElementById("fusion_x").
innerHTML = this.responseText;
                    }
                  };
                  xhttp.open("GET", "/fusion_x", true);
                  xhttp.send();
                }, 100 ) ;

```

Code 4.4 – *Data Reading Variable Pulling*

The last line informs about the pulling period time.

- Interface Communication

  Last two code extracts (Code 4.3 and Code 4.4) are inside the HTML file. However, in order to communicate with our main program, we need to create a sort of *gateway* functions, as well as the necessary libraries (first three lines in Code 4.5). This functions are in charge of passing the variable from the program to the interface and viceversa. Two steps are clearly identified in the next code extract:

```
                /*------------------WIFI-----------------
*/
```

```
                    #include <ESPAsyncWebServer.h>
                    #include <AsyncTCP.h>
                    // String function which checks for changes
in the program variable
                  String processor(const String& var){
                    else if(var == "TEMPERATURE"){ return
temp_str;}
                    ...
                    return String();
                    }
                    // Server request function which sends the
variable to the interface
                    // Request to get temperature and altitude
from the barometer
                    server.on("/temperature", HTTP_GET, [](
AsyncWebServerRequest *request){
                      request->send_P(200, "text/plain", temp_str
.c_str());
                    });

```

Code 4.5 – *WiFi Data Handling*

4

4.2.3 Sensors

4.2.3.1 IMU

The control of our platform twisting is made by the readings from an inertial measurement unit. Over the next section we will explain how these readings are scrutinized, conditioned and interpreted.

Our IMU has 4 MEMS sensors in 3 microchips; an accelerometer with a magnetometer in the same chip, a gyroscope and a barometer. All of these are implemented in the same PCB, which is designed by Adafruit, so the libraries used to program these sensors must come from them.

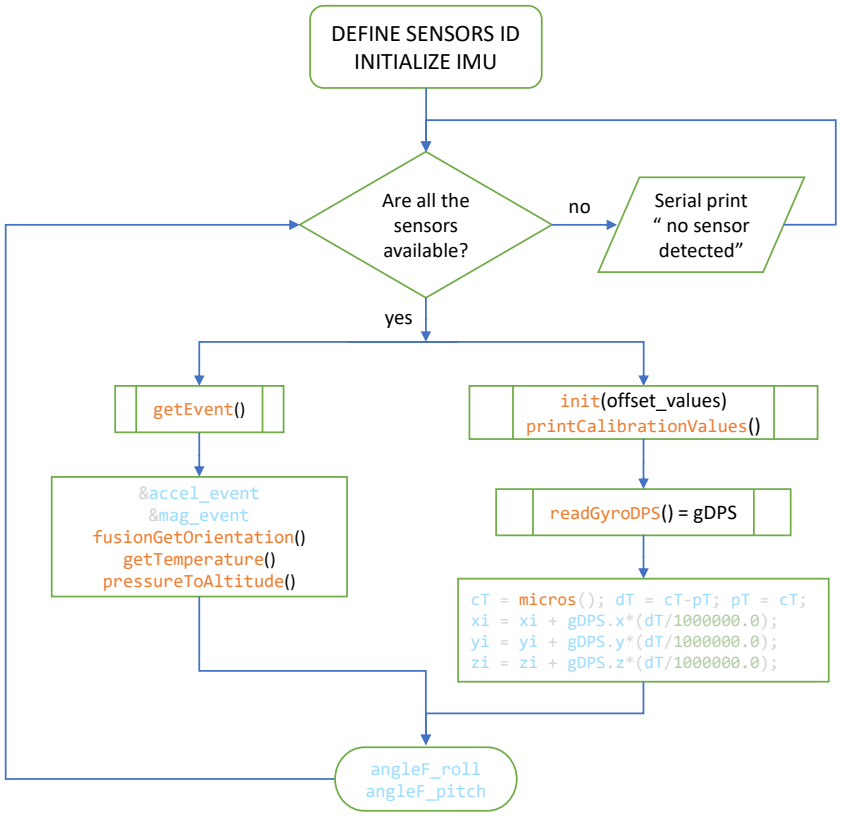


Figure 4.19 – *Diagram Block of the IMU functioning system*

4.2.3.1.1 Accelerometer

The **Euler angles** determine the orientation of a rigid body with respect to a fixed coordinate system (as commented in paragraph 3.2.2.2.1). As of the Euler DCM (Figure 3.2.2.2.1), we can calculate the cartesian coordinates orientation of our system easily. The next implementation code comes from the Adafruit 10 DoF library:

- Roll

  The **roll** is a rotation around the longitudinal axis (the plane body or X axis). In our case, the roll calculation is designed to be positive and increasing when moving downwards. The following equation Equation 4.2.1 is the one we must implement in C (Code 4.6).

$$Roll = \arctan\left(\frac{y}{\sqrt{x^2 + z^2}}\right)^2 \tag{4.2.1}$$

```
1      orientation->roll = (float)atan2(accel_event->
acceleration.y, accel_event->acceleration.z);
2      /* Convert angular data to degree */
3      orientation->roll = orientation->roll * 180 / PI_F;
4
```

Code 4.6 – *Roll Calculation in Adafruit Library*

- Pitch

  The **pitch** is a rotation around the lateral axis (the wing span or Y axis). In our case, the pitch calculation is designed to be positive and increasing when moving upwards. The following equation Equation 4.2.2 is the one we must implement in C (Code 4.8).

$$Pitch = \arctan\left(\frac{x}{\sqrt{y^2 + z^2}}\right)^2 \tag{4.2.2}$$

```
1      orientation->pitch = (float)atan(-accel_event->
acceleration.x / (accel_event->acceleration.y * sin(orientation
->roll) + \
2
                accel_event->acceleration.z * cos(orientation->roll)
));
3      orientation->pitch = orientation->pitch * 180 / PI_F;
4
```

Code 4.7 – *Pitch Calculation in Adafruit Library*

4

4.2.3.1.2 Magnetometer

In order to get the most precise measure we need to use the magnetometer. This sensor is very sensitive to the magnetic fields, so when near of them the measures could result in error. However, we use this sensor with a very low gain (and therefore low impact on the data reading), only to determine the **yaw** imprecise data given by the accelerometer.

The **yaw or heading** is a rotation around the Z axis. In our case, the yaw calculation angle is designed to be positive when the rotation is clockwise around the Z axis. This calculation requires the pitch and roll angles, as the element moves along the XY planar coordinates. For reference, we will consider the angles from Figure 3.22 (a), where $\theta \rightarrow Pitch$ and $\phi \rightarrow Roll$.

$$Yaw = \arctan\left(\frac{z\sin(\phi) - y\cos(\phi)}{x\cos(\theta) + y\sin(\theta)\sin(\phi) + z\sin(\theta)\cos(\phi)}\right)^2 \tag{4.2.3}$$

where x,y,z are the magnetic data angles.

```
1   orientation->heading = (float)atan2(mag_event->
2          magnetic.z * sin(orientation->roll) - mag_event->magnetic.
y * cos(orientation->roll), \
3          mag_event->magnetic.x * cos(orientation->pitch) + \
4          mag_event->magnetic.y * sin(orientation->pitch) * sin(
orientation->roll) + \
```

```
5           mag_event->magnetic.z * sin(orientation->pitch) * cos(
    orientation->roll));
6   orientation->heading = orientation->heading * 180 / PI_F;
7
```

Code 4.8 – *Pitch Calculation in Adafruit Library*

4.2.3.1.3 Gyroscope

As commented in paragraph 3.2.2.2.2, the gyroscope suffers drift over time. This can be fixed with a simple filter where the accelerometer and the gyroscope take part. In order to fix these issues we must do two things; **calibrate** all of the gyro's coordinates each time the program starts (lines 1-5 Code 4.10), and create a **complimentary filter** which takes into account the accelerometer and the gyroscope roll and pitch values (line 9 Code 4.10), each of them with a percentage related to their error quantity.

As we know, the accelerometer gives as the orientation in radians, but the gyroscope expels angular velocity. In order to find the accumulative error, we need to integrate:

$$\int_{t_1}^{t_2} \omega \, dt = \int_{t_1}^{t_2} \beta \Rightarrow \beta_2 = \beta_1 + \int_{t_1}^{t_2} \omega \, dt \tag{4.2.4}$$

Once integrated, we will add the accumulated error in a weighted sum between the accelerometer and the gyroscope:

$$\beta_2 = 0.95 * (\beta_1 + \int_{t_1}^{t_2} \omega \, dt) + 0.05 * \xi \tag{4.2.5}$$

where ξ is the accelerometer's roll angle. In Code 4.9 and Code 4.10 we have the code implementation.

```
1       cT = micros(); dT = cT - pT; pT = cT;
2       xi = xi + gDPS.x*(dT/1000000.0);
3
```

Code 4.9 – *Angular Velocity Integral*

where differential time (dT) is the difference between the current time (cT) and the previous time (pT), obtaining in a discrete way the integer sum (xi) of the measure.

```
1     /*----L3G4200D GYRO LIBRARY----*/
2       for(int i=0; i<samples; i++)
3       { GyroDPS dps = readGyroDPS();
4         xt += dps.x;
5         ...}
6       Serial.println("Gyro Offset (dps): ");
```

```
7         Serial.print("X: "); Serial.print(xt/float(samples),5);
8         ...
9     /*------COMPL FILTER--------*/
10    angleF_roll = 0.95*(angleF_roll + gDPS.x*(dT/1000000.0)) + 0.05*
orientation.roll;
11    ...
12
```

Code 4.10 – *Accumulated Error by the gyroscope and accelerometer*

4.2.3.2 Buzzer

For the magnetic buzzer used the only extra design was a 270 Ω resistor used to make it sound softer. We can see the code block diagram in Figure 4.20.

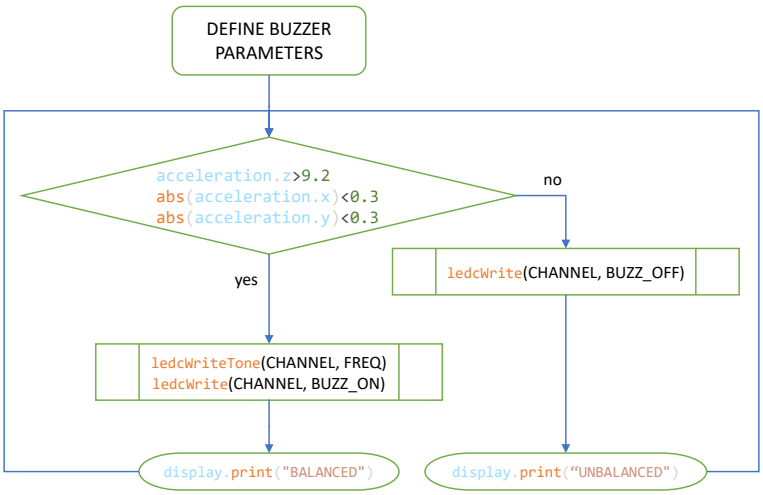


Figure 4.20 – *Buzzer Program Block Diagram*

4.2.4 Software Management. On-Board Data Handling.

In this section we will make a short code approach of every component used. However, the main code program can be seen in Appendix D for a more detailed understanding of the whole system programming.

4.2.4.1 FreeRTOS Programming

FreeRTOS programming is based on simultaneous task management [31] . In order to make it work properly we only need to configure some commands, which are specific from this operating system [38].

```
1     /*------------------CORE SELECTION FROM ESP32
```

```
 --------------------*/
#if CONFIG_FREERTOS_UNICORE
  static const BaseType_t app_cpu = 0;
#else
  static const BaseType_t app_cpu = 1;
#endif
...
void BatteryMeasure(void* parameters)
{
  while(1){
  ...}
  }
...
setup(
xTaskCreatePinnedToCore( BatteryMeasure,"Voltage battery measure"
 ,10240, NULL,1, NULL,app_cpu);)

```

Code 4.11 – *FreeRTOS Configuration and Task Handling*

At first we need to create a definition of which cores we will be using. I use for all of my tasks CPU 1 as CPU 0 is where WiFi works, and it is advisable not to put a lot of work load in it. After that, the functions we want to use must be written as appears, so the parameters can be shared between different functions. At last, the task initialization must be defined in the setup(). There we will define which task we want to work with, how much memory we will let it have and in which CPU should the task work.

4.2.4.2 Sensors Programming

4.2.4.2.1 IMU

Over the next lines I will explain shortly the programming necessary to make the IMU work properly.

Note: the microchip I use was with no reference number or original datasheet when it was given to me in GranaSAT.Therefore I was not sure of all the sensors being the ones Adafruit claims. I came up with that issue while programming the gyroscope, which needed a different library (line 4 Code 4.12). Once that problem was fixed when I found the proper library, the system worked seamlessly with all the sensors taking measurements at the same time.

```
    // Libraries
    #include <Adafruit_Sensor.h>
    #include <Adafruit_LSM303_U.h>
    #include <Adafruit_BMP085_U.h>
    #include <L3G4200D.h>
```

```
#include <Adafruit_10DOF.h>
// Sensors ID
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);
Adafruit_10DOF                dof   = Adafruit_10DOF();
L3G4200D gyro;

```

Code 4.12 – *Libraries & Sensors Identification*

The Code 4.12 determines the needed libraries and the identification of each sensor in the Adafruit PCB. As the data is sent through I2C protocol, every component must be addressed in its respective manner or number.

In order to get the values, an **event protocol** is done (Code 4.13) and addressed to an specific place, in this case *accel event*, *mag event*, etc.

```
// Read values FROM 10DOF CALCULATIONS
sensors_event_t accel_event;      // event ask for accelerometer data
sensors_event_t mag_event;        // event ask for magnetometer data
sensors_event_t bmp_event;        // event ask for barometer data
sensors_event_t event;            // event ask for gyroscope data
sensors_vec_t   orientation;      // orientation vector pointer
GyroDPS gDPS;                     // gyro request for degrees per second

```

Code 4.13 – *Sensors Events*

After having done the former configuration, we can now proceed to read from them. First we make sure each and every sensor has been detected (Code 4.14 for every sensor),

```
 if(!bmp.begin())
{Serial.print("Ooops, no BMP085 detected ... Check your wiring or I2C ADDR!");
while(1);}

```

Code 4.14 – *Barometer Error Message*

afterwards we only need to proceed with the **getEvent()** command to get the data into our *accel event* variable (line 1 Code 4.16), being able to display it as well. As the gyroscope has a different library, it uses a different method to read data (line 7 Code 4.16).

```
1        accel.getEvent(&accel_event);  // Get event from
accelerometer
2        ...
3          display.print(String(accel_event.acceleration.x));  //
Display through OLED screen
4          Serial.print(accel_event.acceleration.x);           //
Display through Serial Monitor
5        ...
6        // Measurements from gyroscope in DPS (degrees per second)
7        gDPS = gyro.readGyroDPS();
8
```

Code 4.15 – *Data Event Reading & Display*

To get the actual orientation of the platform we use the library from Adafruit. This command merges the readings from two sensors; the accelerometer and the magnetometer (Note: the usage of the magnetometer sensor could turn into a problem if **strong magnetic fields** are applied near the sensor).

```
1      if (dof.fusionGetOrientation(&accel_event, &mag_event, &
orientation)){
2        fusion_pitch = String(orientation.pitch);
3        fusion_roll  = String(orientation.roll);
4        fusion_heading = String(orientation.heading);
5      }
6
```

Code 4.16 – *Data Event Reading & Display*

4.2.4.2.2 OLED Display

The programming of the OLED screen follows the next diagram (Figure 4.21):

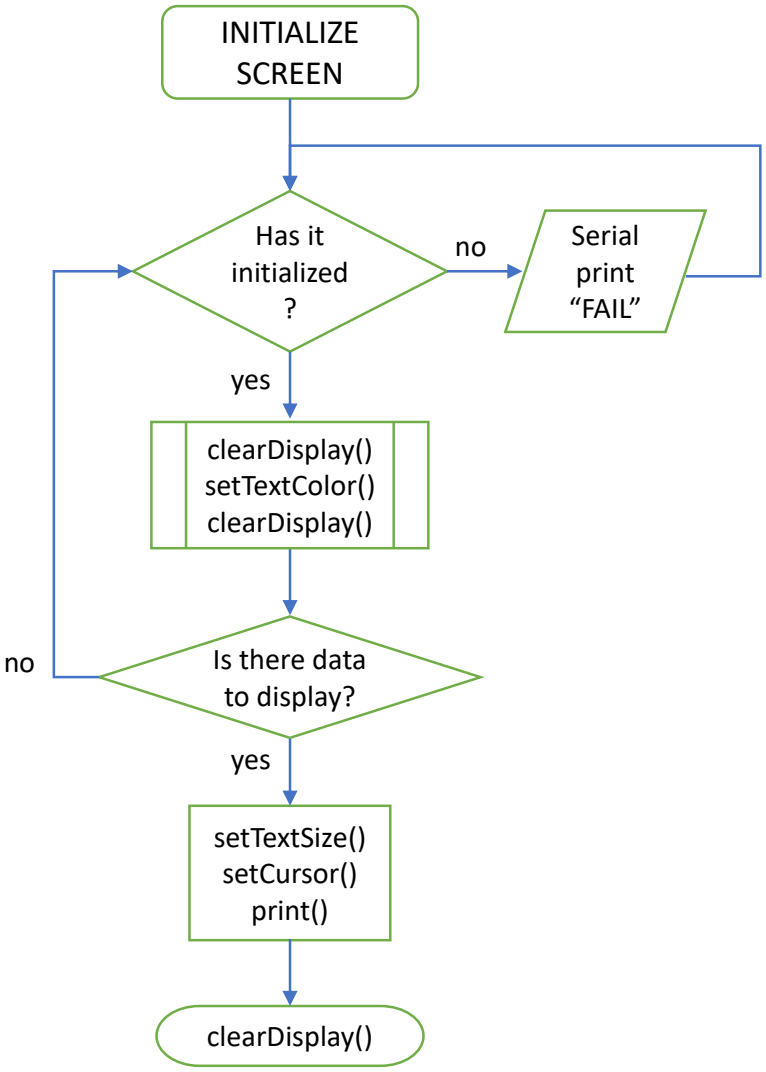


Figure 4.21 – *SSD1306 Program Block Diagram*

Over the next figures (Figure 4.22 and Figure 4.23) we can observe the internal interconnection of the SSD1306:

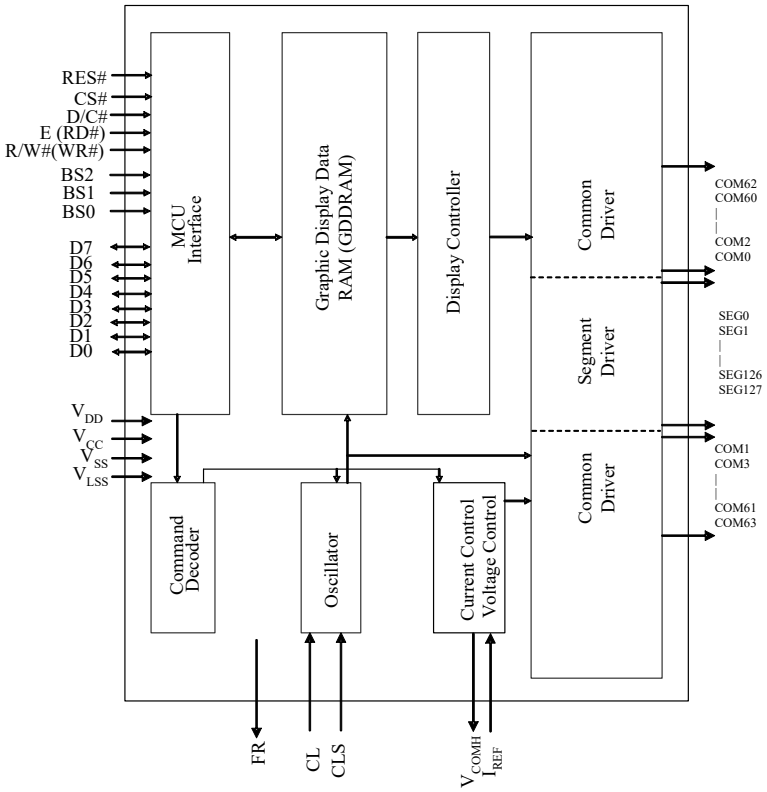


Figure 4.22 – *Block Diagram of the internal SSD1306 screen circuitry*

The connection between these pins (Figure 4.22) is made through I2C protocol (Figure 4.23):

Pin Name / Bus Interface	Data/ Command Interface								Control Signal				
	D7	**D6**	**D5**	**D4**	**D3**	**D2**	**D1**	**D0**	**E**	**R/W#**	**CS#**	**D/C#**	**RES#**
8-bit 8080	D[7:0]								RD#	WR#	CS#	D/C#	RES#
8-bit 6800	D[7:0]								E	R/W#	CS#	D/C#	RES#
3-wire SPI	Tie LOW					NC	SDIN	SCLK	Tie LOW		CS#	Tie LOW	RES#
4-wire SPI	Tie LOW					NC	SDIN	SCLK	Tie LOW		CS#	D/C#	RES#
I^2C	Tie LOW					SDA_{OUT}	SDA_{IN}	SCL	Tie LOW			SA0	RES#

Figure 4.23 – *MCU interface assignment*

The OLED screen communicates through I2C protocol, so we will need to assign its address in the program (line 3 Code 4.19), as well as the pixel numbers to make the program aware of the screen limits (lines 1-2 Code 4.19).

```
1       #define SCREEN_WIDTH 128 // OLED display width, in pixels
2       #define SCREEN_HEIGHT 64 // OLED display height, in pixels
3       #define SCREEN_ADDRESS 0x3C ///< See datasheet for Address;
0x3C for 128x64, 0x3D for 128x32
4         Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &
Wire, -1);
5
```

Code 4.17 – *OLED display definition*

Its usage is very simple, just change the serial command for the display command (Code 4.18) .

```
        Serial.print(accel_event.acceleration.x); Serial.print(F(";
")); // Through Serial
        display.print(String(accel_event.acceleration.x));  //
Through OLED screen

```

Code 4.18 – *Display command for OLED screen*

For the display of our data we only need to write down the size (*setTextSize()*) and the original position of the sentence or figure (*setCursor()*) we would like to print:

```
          display.clearDisplay();         // Clear former data shown
in OLED
            display.setTextSize(1); display.setCursor(0,26); display
.print("Y: ");
            display.print(String(accel_event.acceleration.y));

```

Code 4.19 – *OLED display definition*

4.2.4.2.3 Button

In our case we will use the button for the activation of the Motors altogether with a virtual button present in our Interface [20]. The implementation is more or less the same steps to follow as in subsubsection 4.2.2.3, but here we will implement the virtual and physical button code implementation.

```
          //
////////////////////////////////////////////////////////////////////

          // String function differences compared to the variables
for data reading
          String processor(const String& var){
 //Serial.println(var);
 if(var == "BUTTONPLACEHOLDER"){
   String buttons ="";
   String outputStateValue = outputState();
   buttons+= "<h4>Output - GPIO 2 - State <span id=\"outputState
\"></span></h4><label class=\"switch\"><input type=\"checkbox\"
onchange=\"toggleCheckbox(this)\" id=\"output\" " +
outputStateValue + "><span class=\"slider\"></span></label>";
```

```
9       return buttons;
10    }}
11    ////////////////////////////////////////////////////////
12    button_read = digitalRead(buttonPin);
13    if (button_read != lastButtonState) {
14          lastDebounceTime = millis();   // update the value of
      debounce from button
15      }
16    if ((millis() - lastDebounceTime) > debounceDelay) {
17        if (button_read != button_state) {     // if the value read is
      different from the
18          button_state = button_read;          // former state value of
       the button, update it.
19          if (button_state == HIGH) {          // If it's HIGH, change
      the state of the variable
20              activateMotors == true;
21
22
```

Code 4.20 – *Button Implementation [20]*

Basically, after reading the signal with the corresponding delay (pushing a button is not instantaneous), it determines if the signal is HIGH or LOW, taking care of the corresponding task and activating the *flag activateMotors*, which then activates the function for the motors initialization.

4.2.4.2.4 Buzzer

First is the definition of the buzzer, with the ESP's channel we will use for the PWM [35], the frequency of the signal and the resolution (line 1 Code 4.21), as well as the pin (line 2). Once the former configuration is done, we can use the proper commands to make it sound (lines 4-5 Code 4.21).

```
1       ledcSetup(channel_buzzer, freq_buzzer, resolution_buzzer);
2       ledcAttachPin(13, channel_buzzer);
3       ...
4       ledcWriteTone(channel_buzzer, freq_buzzer);
5       ledcWrite(channel_buzzer, duty_c_buzzer_on);
6
```

Code 4.21 – *Buzzer Definition & Activation [35]*

4.2.5 PCB Design

The next boards have been designed to meet the requirements of our system. The tachometer and the first prototype have been produced in the GranaSAT facilities. The

CNC machine used was MODEL OF THE CNC MACHINE USED.

Altium Designer® 19 was chosen as the program to design the PCB because of its popularity in the electronic business and the integrated solutions offered; manufacturer part search with stock availability, 3D CAD advanced solutions, budget and schematic documentation with a high-end level, etc. For the next step, pre-production and production, two programs were used. The first one is CircuitCAM 5.0 ©, a CAM system for PCBs. This program was chosen because of its versatility, easy-to-use interface and low economic cost. To manage and control the CNC PCB Milling machine we used the program BUNGARD RoutePro 2000 ©.

4.2.5.1 Design

The design of the PCB for the whole system should adapt to the simulation platform form, so that's why we decided to make it round. This design allows us to maintain a good proportion of space for every component, and at the same time keep the overall form of the system. All of the components used in this PCB can be seen in Section 3.2.

Over the next pages the CAD drawing from the main PCB is shown.

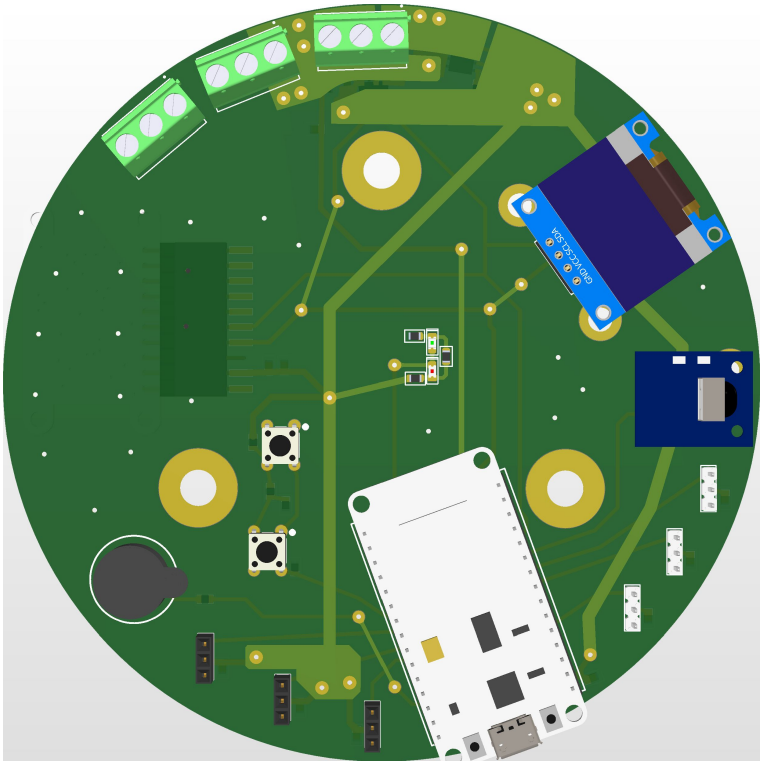


Figure 4.24 – *Top View of the main PCB*

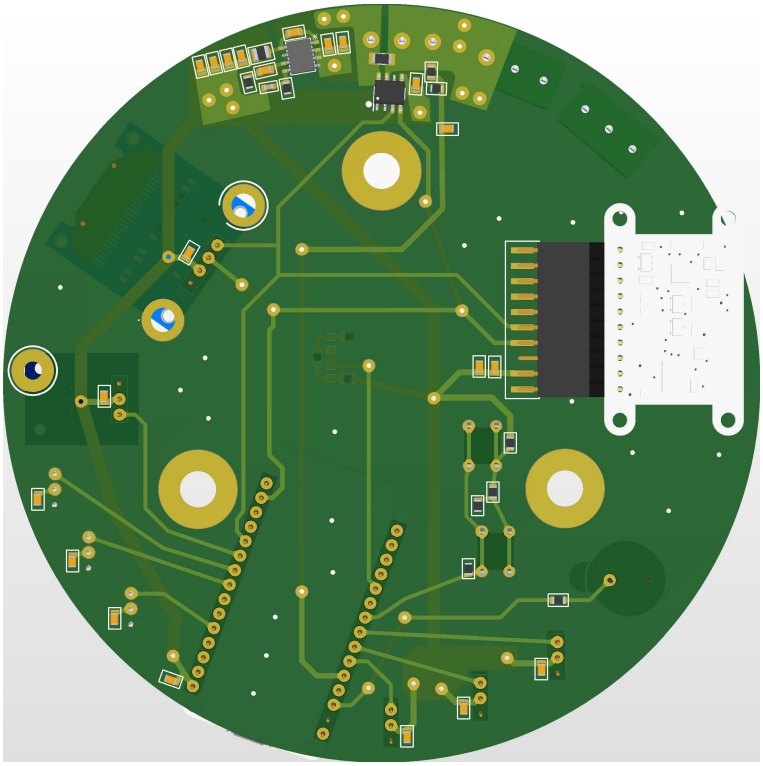


Figure 4.25 – *Bottom View of the main PCB*

For the next 3D pdf, if available, please use Adobe Acrobat Reader for 3D objects visualization.

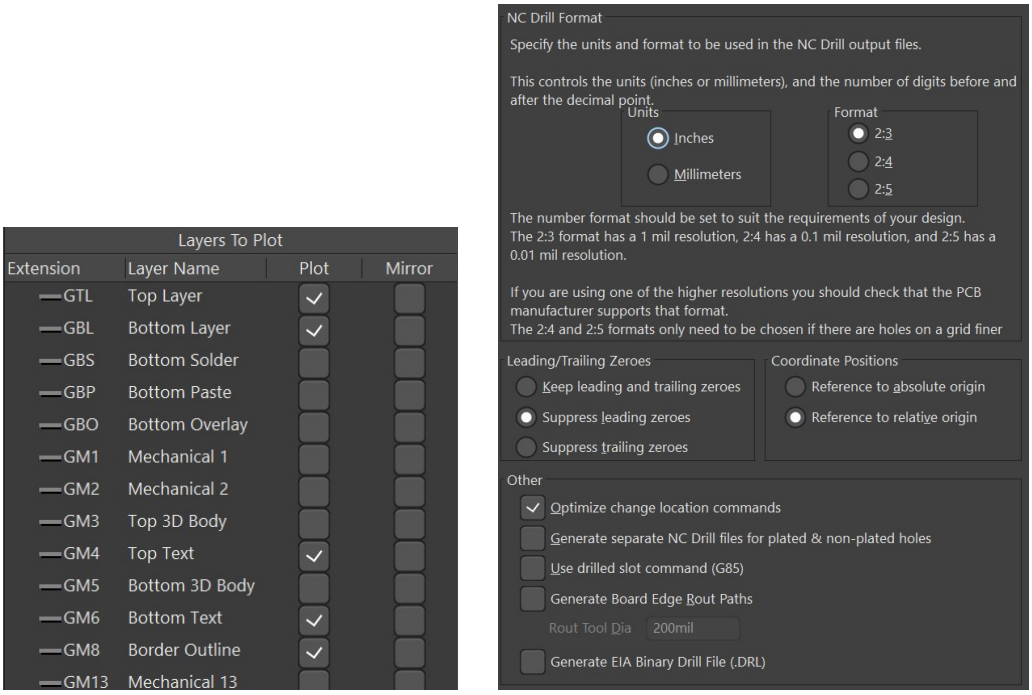
4.2.5.2 Manufacturing & Soldering

First we will address the preparation for **Manufacturing**.

The steps to be followed by are explained and illustrated below:

- Export Gerber and Drilling files from Altium Designer ©

  We must export the layers related to the tracks, component designation, vias and holes (Figure 4.26). These are the top and bottom layers, the top text and bottom text layers, and the drilling layer (for components through hole and fiduciary or component placing dedicated holes) respectively.



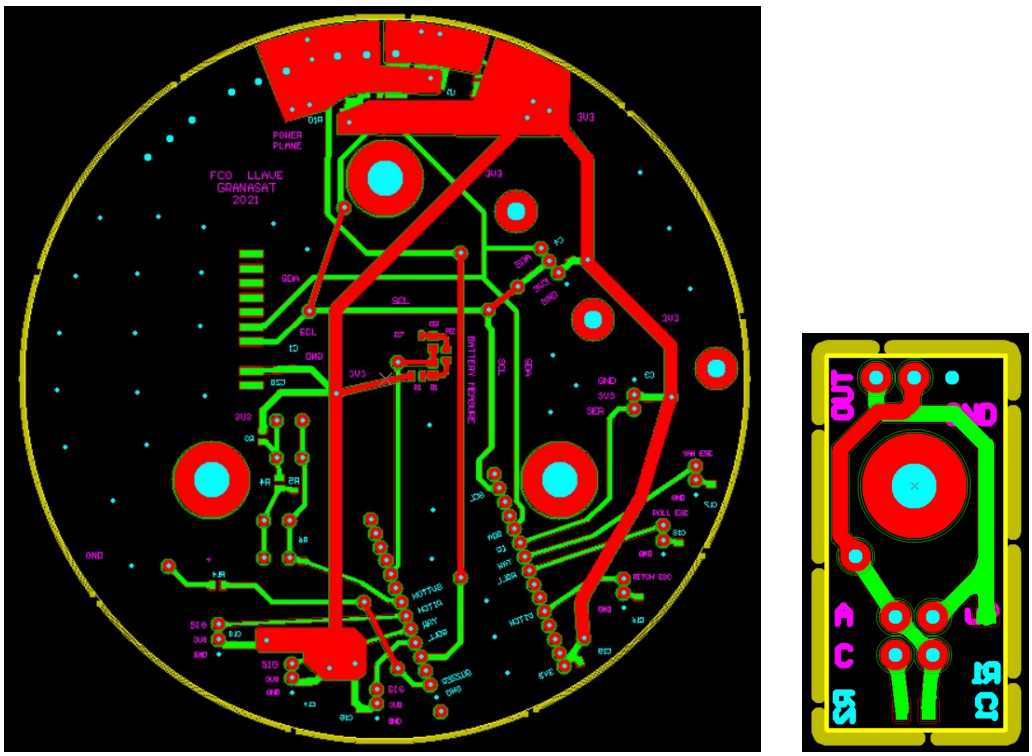
(a) *Gerber Files* **(b)** *NC Drilling Files*

Figure 4.26 – *Altium Designer® 19 windows of the Exportation configuration*

  It is paramount to add the **fiducials**. These holes are needed when twisting the PCB for the bottom tracks definition, in order to keep the origin on the CNC working map. In figure (4.29) two holes next to the PCB, one at each side at a distance of 20*mm* (because of the tool diameter) from the PCB minimum, define fiducials' position.

  Options like *Suppress Leading Zeros* or the *Units* and *Format* of exportation must be kept in mind when importing the files in CircuitCAM 5.0 © in order to keep the scale and position of the elements properly.
- Import Gerber and Drilling files into CircuitCAM 5.0 ©

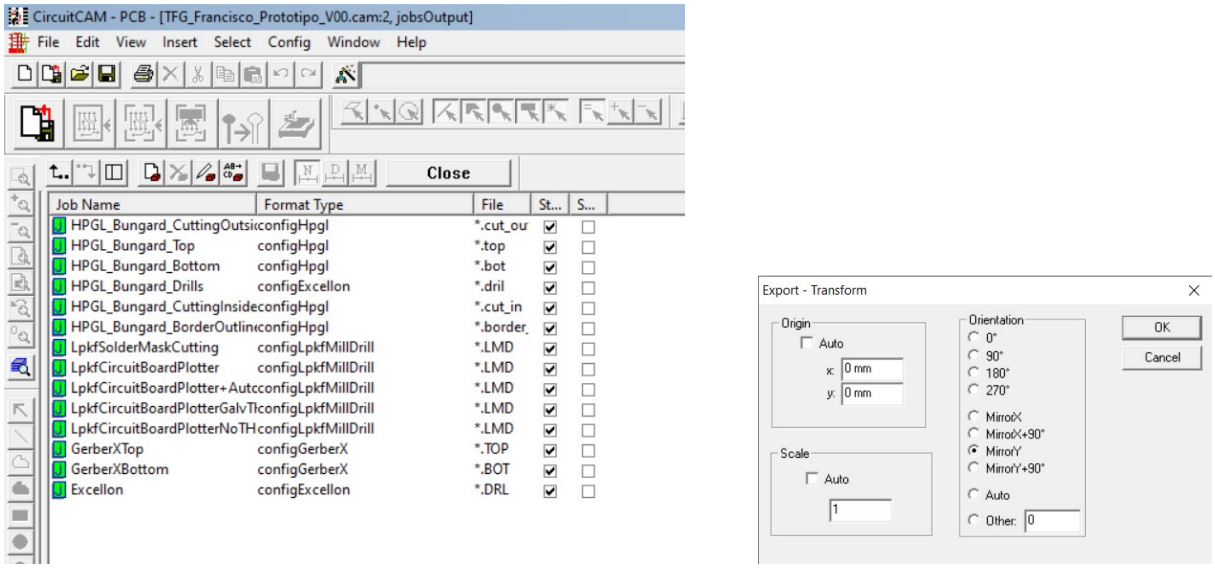
In CircuitCAM 5.0 © (Figure 4.27) we will select which tools to use, how layers will be made (bottom layer needs to be told how will we twist the PCB, over the Y or X axis) and determine the outline border (where it will be cut) for the PCB.



(a) *System PCB* **(b)** *Tachometer PCB*

Figure 4.27 – *View from CircuitCAM 5.0 © before exportation to RoutePro*

For the proper exportation of these files into the fabrication program, some adjustments must be made. As an example, we will show one step regarding the bottom layer indication



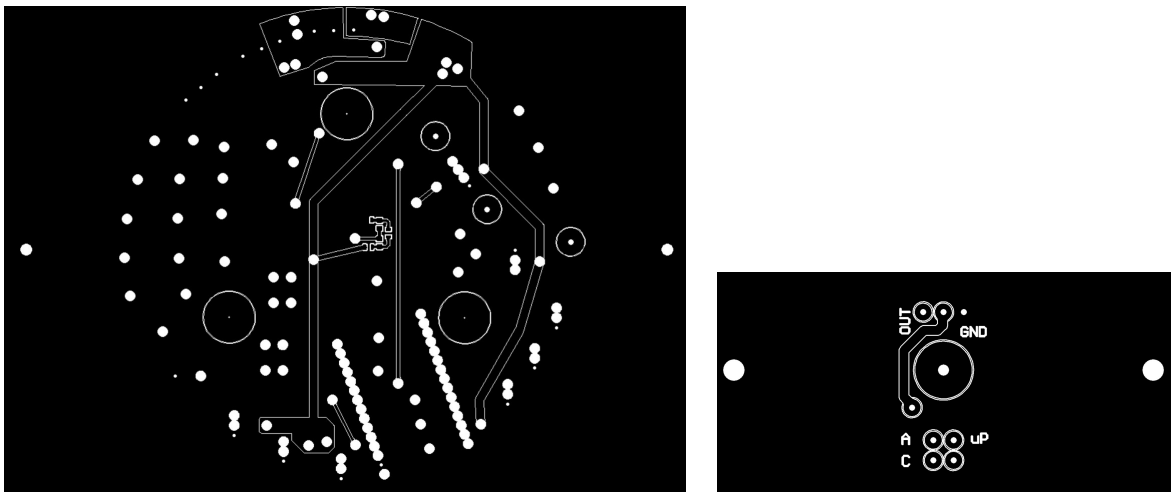
(a) *Export Files Configuration window* **(b)** *Configuration Tab in the HPGL_Bungard_Bottom*

Figure 4.28 – *Export Configuration Windows in CircuitCAM 5.0 ©*

These configuration parameters attend to the way the tools must behave, which nominal values will be passed to RoutePro and how the tracks or vias will be made.

- Import CircuitCAM 5.0 © files into *RoutePro*2000

  After the fabrication files preparation has been made, we need to import those files into RoutePro



(a) *System PCB* **(b)** *Tachometer PCB*

Figure 4.29 – *View from the RoutePro before fabrication starts*

The white rounded elements in (4.29) represent the drilling holes (vias, pins and fiduciary holes). On the other hand we can see the thin white lines representing the tracks. This is made with a 0.2*mm* mill tool, which needs to be calibrated manually

over the PCB copper board. For that we use a magnifying lens which tells us if the hole it das meets the 0.2*mm* requirement. This is very important because the mill is triangular, so we need to be careful not to make it bigger as we could link undesired elements.

Once the former steps have been followed, we can start with the manufacturing.

- First we proceed to drill the holes. Here it is paramount not to forget to put caps in the fiduciary holes, as they will prevent the board from moving when routing. It is advisable to use duct tape to stick the sides to the CNC machine, so it prevents from slipping as well.

  We will use three drilling tools, 0.9*mm* for the vias, 2.5*mm* for the fiduciary holes and 3*mm* for mounting holes.

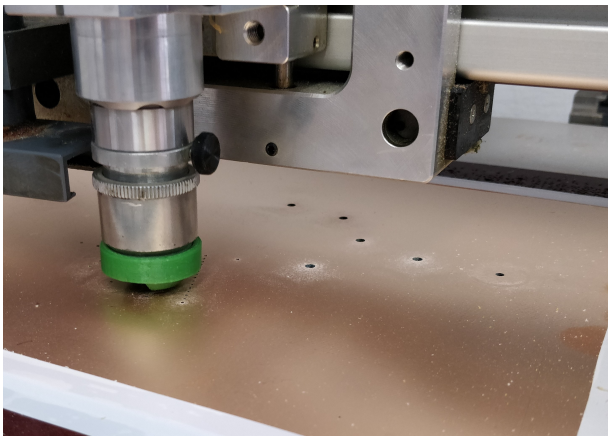
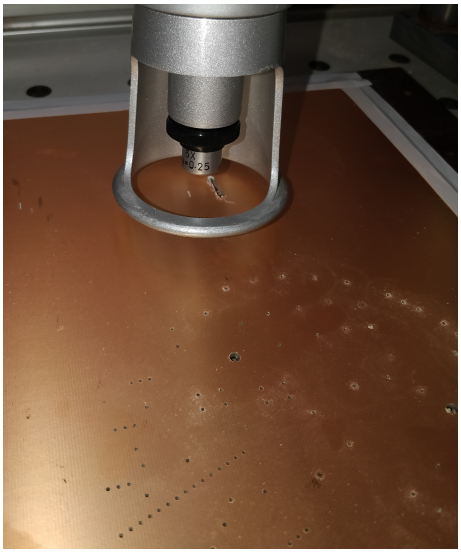


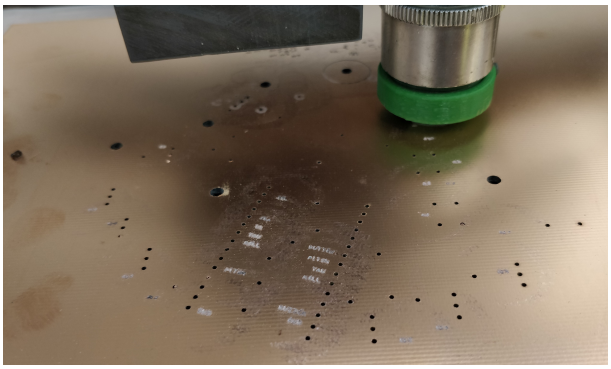
Figure 4.30 – *Drilling Holes*

- Once the drilling is done we can proceed with the routing. For this we will use the milling tool, which will engrave the text and the tracks, separating the mass plane from the conducting tracks.

  We will use one milling tool, a 0.2*mm* triangular tool which has to be calibrated. In order to obtain the 0.2*mm* we will use a magnifying lens (Figure 4.31 a) which has a 0.2mm scale, to know if we are working with the desired measurement.



(a) *Magnifying Lens*

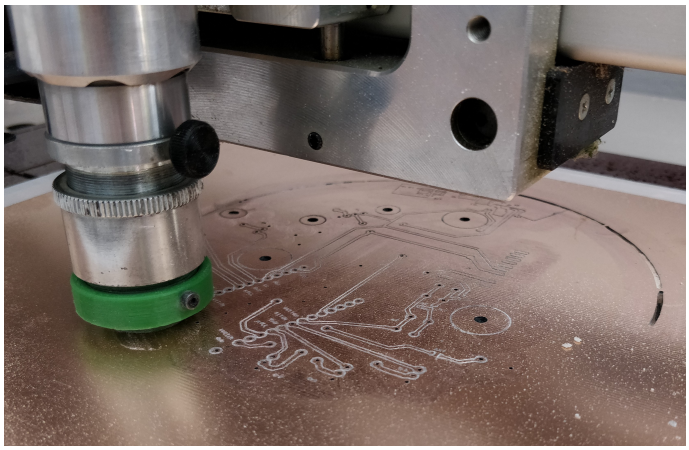


(b) *Milling Process*

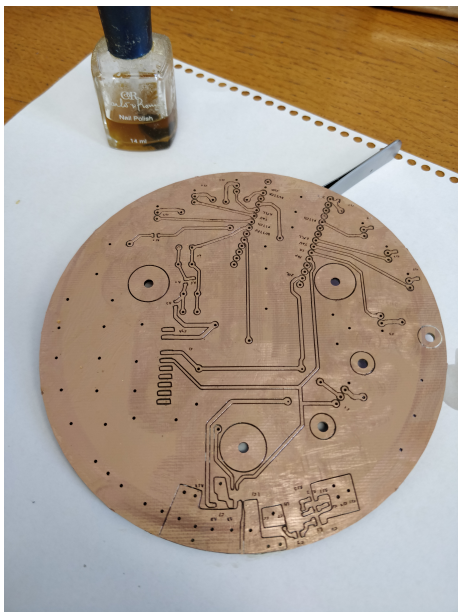
Figure 4.31 – *Milling Procedure*

Once calibrated, we can proceed with the milling job (Figure 4.31 b). Once the top layer has been milled, we have to twist the board to mill the bottom one. As commented earlier, this is what fiduciary holes were designed. It is paramount to know for sure **if the twist must be made throughout the X or Y axis**, as the vias must connect the right tracks.

- After the milling job is finished in both sides, we only have the border line cutting left (Figure 4.32 a). This will allow us to separate the PCB from the board.



(a) *Cutting Outline PCB Border*



(b) *Finished PCB Pre-polished*

Figure 4.32 – *Final Procedure & Polishing*

In order to prevent the board from rusting, it is advisable to apply nail polish over the surface. This will create a sort of mask, which also helps to avoid unexpected contacts between electrical elements (Figure 4.32 b).

- Ultimately, we reach the completion of the PCB manufacturing. After the polishing, we need to let it dry in the Sun (Figure 4.33) for half an hour approximately.

Figure 4.33 – *Polished PCB Drying*

Having finished with manufacturing we can undertake the **Soldering**. We will have to address standard and SMD soldering. For the vias communication between layers we will use recycled axial resistors legs. In autoref we can see a close image of the PCB, with the soldered components and connections.

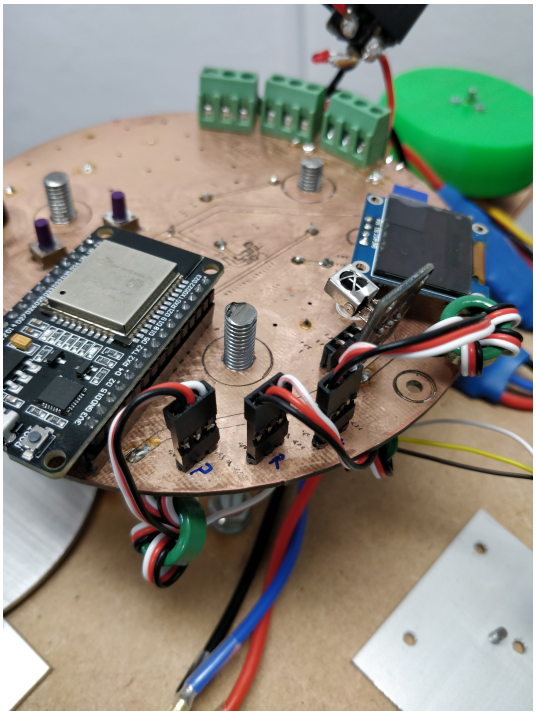


Figure 4.34 – *Detail of the PCB with all the components*

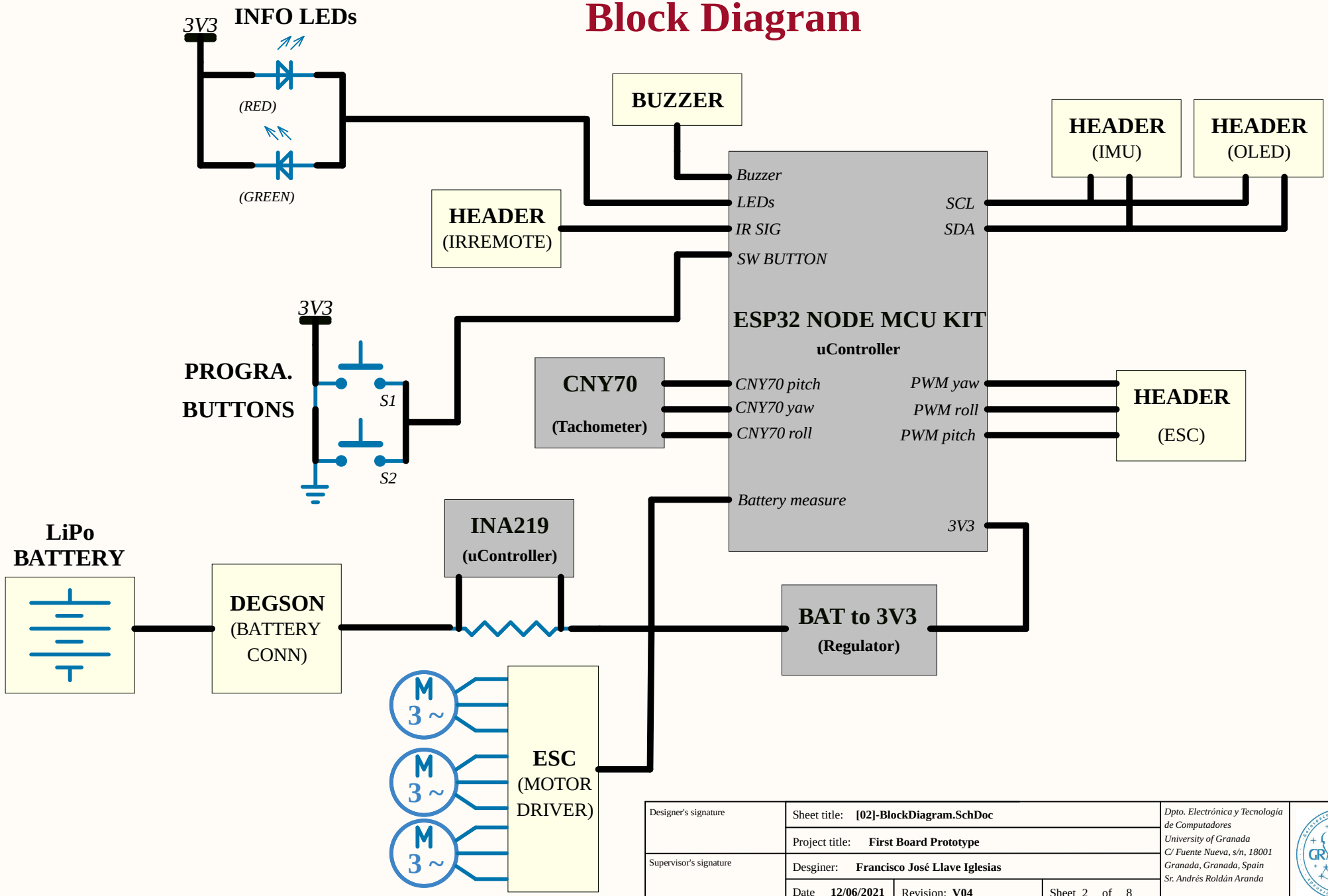
POWER BUDGET

MODULE	DEVICE	INFO	MIN CURRENT CONS ()	TYP CURRENT CONS (mA)	MAX CURRENT CONS (mA)	MAX VOLTAGE (V)	POWER (mW)
uProcessor	ESP-32-WROOM	Max current delivered by power supply		500	500	3,6	1800
IMU	BMP180	Ultra low power mode		0,003		3,6	
		Standard mode		0,005			
		High resolution mode		0,007			
		Ultra high res. mode		0,012			
		Advanced res. mode		0,032			
		Peak current (conversion)			0,65		2,34
	L3GD20	Supply current		6,1		3,6	21,96
		Current sleep mode		2			
		Current power down		0,005			
	LSM303DLHC	Current normal mode		0,11		3,6	0,396
		Current sleep mode		0,001			
OLED	SSD1306	Max sink current			20	3,6	72
REFLECTIVE OPTICAL SENSOR	CNY70	Diode emitter current			20		
		BJT collector current			1		
		Total			21	3,6	75,6
INFRARED RECEIVER	KY-022	Current consumption	0,4		1,5	3,6	5,4
MAGNETIC BUZZER	KXG-1205	Max current			45	3,6	162
					BOARD(mA)		BOARD(mW)
					594,36		2139,696

Designer's signature	Sheet title: **[01]-CoverPage.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores*
	Project title: **First Board Prototype**		*University of Granada*
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		*C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
	Date **12/06/2021**	Revision: **V04**	Sheet 1 of 8

Sr. Andrés Roldán Aranda

Block Diagram



Designer's signature	Sheet title: **[02]-BlockDiagram.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores*
	Project title: **First Board Prototype**		*University of Granada*
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		*C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain*
	Date **12/06/2021**	Revision: **V04**	Sheet 2 of 8 *Sr. Andrés Roldán Aranda*

ADAFRUIT IMU 10 DOF

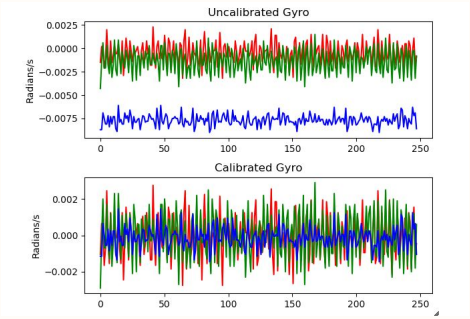
DATASHEET

	L3GD20
SUPPLY VOLTAGE	2.4 to 3.6 V
CURRENT CONSUMPTION	5 µA to 6.1 mA
OPERATING TEMPERATURE	-40 to +85 ºC
MEASUREMENT RANGE	±250 to ±2000 dps (degrees per second)
SENSITIVITY	250 dps -> 8.75 mdps/digit 500 dps -> 17.5 mdps/digit 2000 dps -> 70 mdps/digit

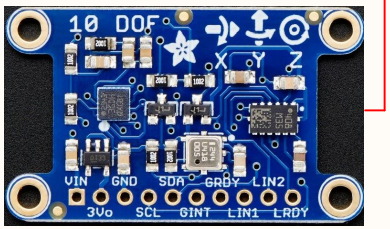
DATASHEET

	LSM303DLHC
SUPPLY VOLTAGE	2.16 to 3.16 V
CURRENT CONSUMPTION	1 to 110 µA
OPERATING TEMPERATURE	-40 to +85 ºC
MAGNETIC RESOLUTION	2 mgauss
LINEAR ACCEL. ZERO-G LEVEL OFFSET ACCURACY	± 60 mg

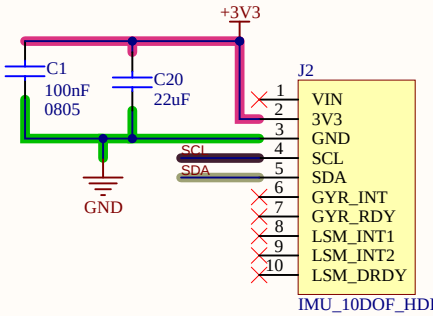
GYROSCOPE CALIBRATION COMPARISON CHART:

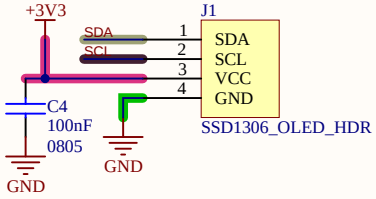


REAL MODEL



The labels GYR_RDY, GYR_INT, LSM_INT_1, LSM_INT_2, LSM_DRDY are for extra settings (WE DO NOT USE THEM)

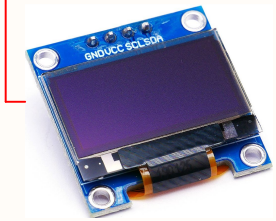




SSD1306 OLED SCREEN 128x64

The connection to the I2C of the ESP32 does not need pull up resistors as the OLED is connected between the IMU (which already has pull up resistors) and the SDA and SCL of the ESP32.

REAL MODEL



DATASHEET

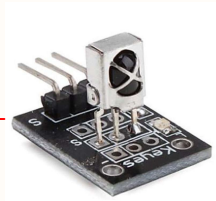
	OLED SCREEN
POWER SUPPLY	1.65 to 3.3 V
SEGMENT MAXIMUM SOURCE CURRENT	1.25 to 1.6 V
COMMON MAXIMUM SINK CURRENT	940 nm
AMBIENT TEMPERATURE RANGE	-40 to + 85 ºC

PCB VIEW

HOLE ⌀M6 MH1 HOLE ⌀M6 MH2 HOLE ⌀M6 MH3

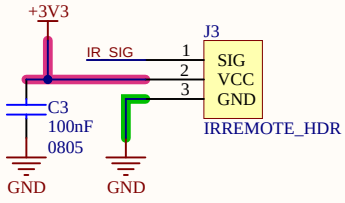
INFRARED REMOTE RECEIVER KY-022

REAL MODEL



DATASHEET

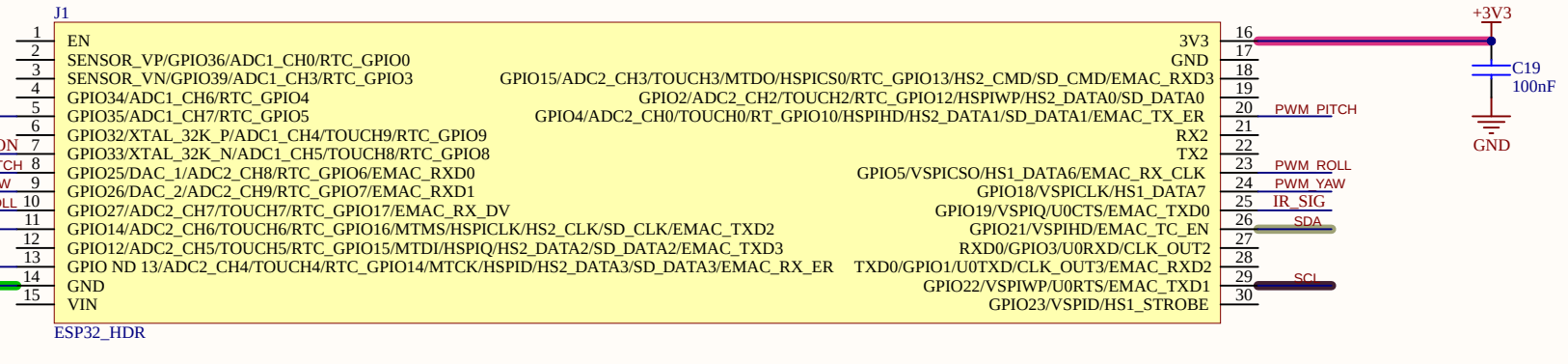
Operating Voltage	2.7 to 5.5V
Operating Current	0.4 to 1.5mA
Reception Distance	18 meters
Reception Angle	±45º
Ambient Light Filter	up to 500 Lux
Carrier Frequency	38 kHz



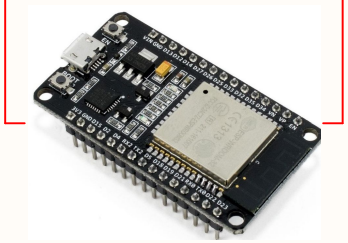
HOLE ⌀M3 MH4 HOLE ⌀M3 MH5 HOLE ⌀M3 MH6

Designer's signature	Sheet title: **[03]-IMU-OLED-IR-MH.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
Supervisor's signature	Project title: **First Board Prototype**		
	Desginer: **Francisco José Llave Iglesias**		
	Date **12/06/2021**	Revision: **V04**	Sheet 1 of 2

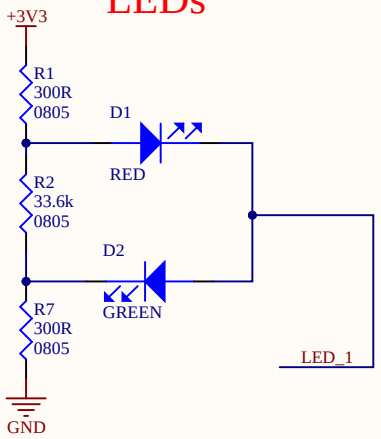
ESP32 NODE MCU



REAL MODEL



LEDs

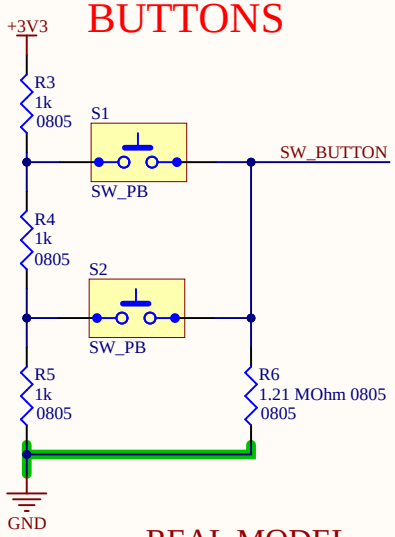


DATASHEET

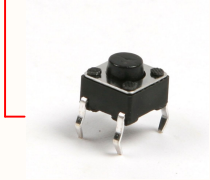
COLOR	MATERIAL	LUM. INTEN.	DOM. WAVELEN	FOR. VOLT.
GREEN	Gap	40 lux	570 nm	min 1.8 V
RED	AllInGap	80 lux	635 nm	min 1.7 V

THESE VALUES ARE VALID FOR A FORWARD CURRENT OF I = 20 mA

BUTTONS



REAL MODEL

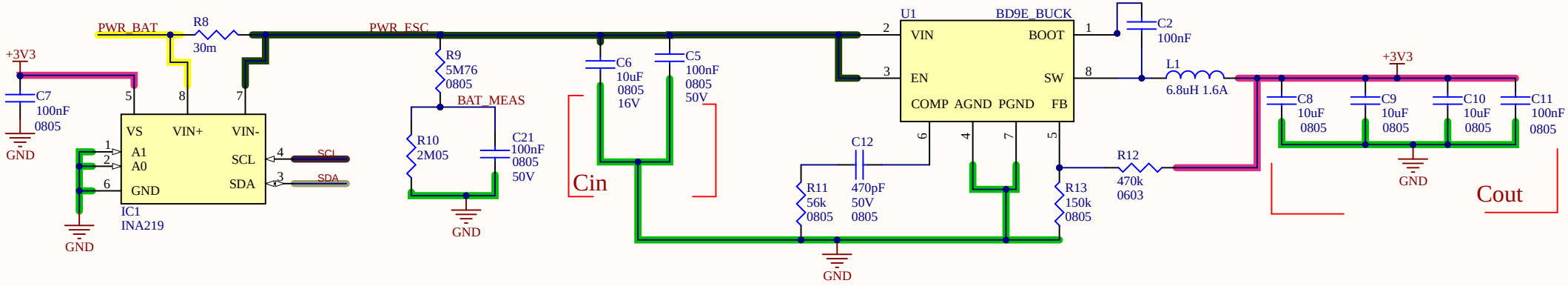


DATASHEET

Module model	ESP-WROOM-32s
Size	25.4*48.26*3mm(±0.2mm)
Certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC/ BQB/ RoHS/REACH
SPI Flash	32Mbit(default)
Support interface	UART/GPIO/ADC/DAC/SDIO/SD card /PWM/I2C/I2S
Integrated crystal oscillator	40MHz Crystal oscillator
IO Port	38
Antenna	Onboard antenna
Power Supply	Voltage 3.0V ~ 3.6V, Typical 3.3V, Current >500mA
Operating Temperature	-40 ℃ ~ 85 ℃
Storage Environment	-40 ℃ ~ 120 ℃

Designer's signature	Sheet title: **[04]-ESP32-LED-SW.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	Project title: **First Board Prototype**		
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		
	Date **12/06/2021**	Revision: **V04**	Sheet 2 of 2

INA219 BATT MEASURE BUCK CONVERTER



R8 has been designed for a maximum current of approx. 10A, which gives us 2.44mA of resolution.

DESIGN NOTE:
- Differential Voltage: By default it is programmed to 320 mV.

- Power Supply Bypassing: Use a 0.1-µF ceramic capacitor for power-supply bypassing, placed as closely as possible to the supply and ground pins.

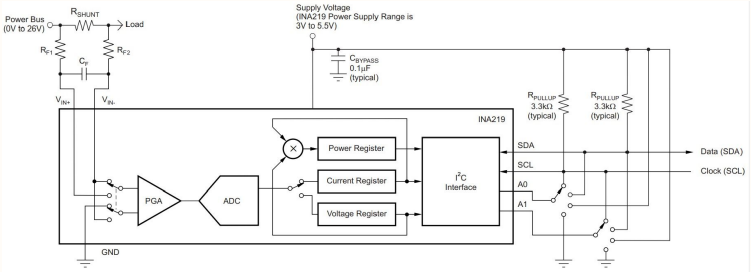
- Input Filter Circuit: It is not necessary to use Rfi and Cf, only if there are transients at exact harmonics of the 500-kHz (±30%) sampling rate (>1 MHz).

- Pull-Up Resistors: they are not needed if there are pullup resistors on these same lines elsewhere in the system.

$$L_{MIN} \geq 0.36 \cdot \frac{V_{OUT}}{f_{sw}} = \frac{3.3V}{400kHz} = 8.25\mu H$$

DESIGN NOTE:
- Cout: This can be achieved with a bank of 2 × 22-µF. In some cases, an aluminum electrolytic capacitor can be placed in parallel with the ceramics to help build up the required value of capacitance. Use a capacitor of at least 10 V for output voltages of 3.3 V.
Ceramic capacitors in the range of 1nF to 100 nF can be very helpful in reducing voltage spikes on the output caused by inductor and board parasitics.

- Cin: A minimum of 4.7 µF (50 V) of ceramic capacitance is required on the input. This capacitance can be increased to reduce input voltage ripple. A 220-nF (50 V) ceramic capacitor must be used at the input as close a possible to the regulator, providing a high frequency bypass.



f_{SW} (kHz)	V_{OUT} (V)	L (µH)	C_{OUT} (RATED CAPACITANCE)	R_{FBT} (Ω)	R_{FBB} (Ω)	C_{IN} + C_{HF}	C_{BOOT}	C_{VCC}	C_{FF}
400	3.3	10	2 × 22 µF	100 k	43.2 k	4.7 µF + 220 nF	100 nF	1 µF	Open
1400	3.3	2.2	1 × 22 µF	100 k	43.2 k	4.7 µF + 220 nF	100 nF	1 µF	Open
400	5	10	2 × 22 µF	100 k	24.9 k	4.7 µF + 220 nF	100 nF	1 µF	Open
1400	5	2.2	1 × 22 µF	100 k	24.9 k	4.7 µF + 220 nF	100 nF	1 µF	Open
400	12	15	2 × 22 µF	100 k	9.09 k	4.7 µF + 220 nF	100 nF	1 µF	Open
1400	12	4.7	2 × 10 µF	100 k	9.09 k	4.7 µF + 220 nF	100 nF	1 µF	Open

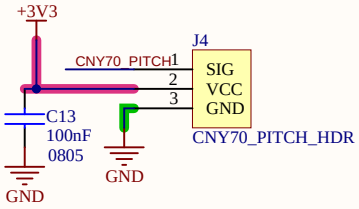
DATASHEET

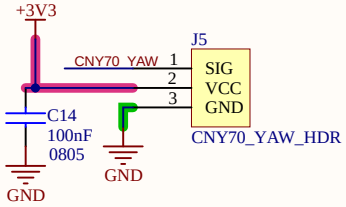
Input Voltage Range	3.8 to 36 V
Output Voltage Range	1 to 24 V
Switching Frequency	400 kHz
Min. Efficiency	95%
Junction Temperature	−40°C to 150°C

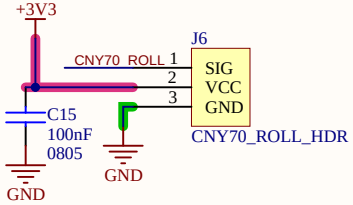
Designer's signature	Sheet title: **[05]-INA-BUCK.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores*
	Project title: **First Board Prototype**		*University of Granada*
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		*C/ Fuente Nueva, s/n, 18001*
	Date **12/06/2021**	Revision: **V04**	Sheet 5 of 8	*Granada, Granada, Spain*
			Sr. Andrés Roldán Aranda

CNY70

Female headers to feed the CNY70 and read from the the pulses generated:







REAL MODEL

In order to measure the spinning velocity of the inertial disks we have chosen the CNY70 to work as a tachometer.

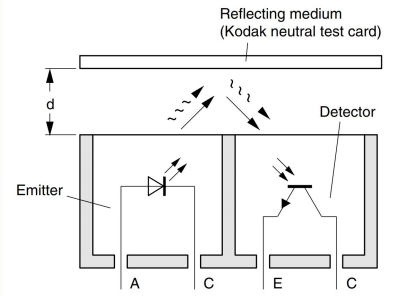
IT contains an IR-emitting diode as a transmitter and a phototransistor as a receiver. The radiation emitted has a wavelength of 950nm, which then reflects over a bright surface (the best and most effective is white colour) and goes back to the phototransistor. This answer can be seen as a voltage change, passing from low to high.

Later on, knowing the period of time and how much of it has it been high, we can calculate the revolutions per minute (rpm) of our inertial disk, eventually calculating the angular velocity (rad/s).

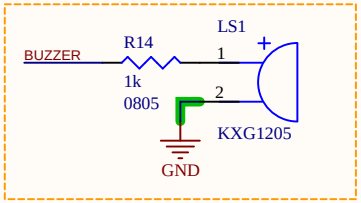
DATASHEET

	CNY70
COLLECTOR CURRENT	0.3 to 1 mA
DIODE FORWARD VOLTAGE	1.25 to 1.6 V
PEAK WAVELENGTH	940 nm
AMBIENT TEMPERATURE RANGE	-40 to + 85 ºC

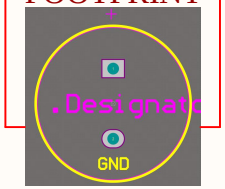
Emitting and receiving process:



MAGNETIC BUZZER KXG1205



FOOTPRINT



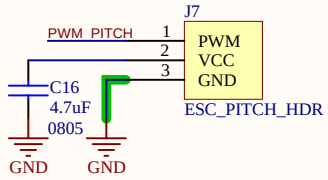
REAL MODEL

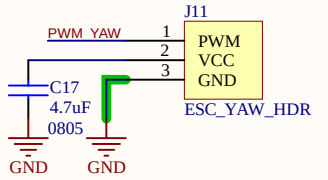
The magnetic buzzer works through PWM signal reading.
To get the maximum volume we must select the PWM to the 50%, but it will also consume the maximum current, so it is better to select 20-30%.

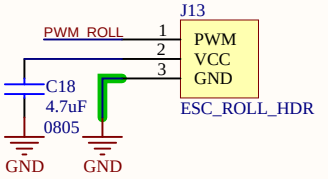
DATASHEET

	KXG1205
RATED VOLTAGE	5 V
OPERATING VOLTAGE	3 to 8 V
MEAN CURRENT (square)	Max. 45 mA
SOUND OUTPUT (typ)	92 dBA
OPERATING TEMPERATURE	-30 TO 70 ºC

Designer's signature	Sheet title: **[06]-CNY70-BUZZER.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	Project title: **First Board Prototype**		
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		
	Date **12/06/2021**	Revision: **V04**	Sheet 6 of 8







ESC 30A UBEC

REAL MODEL

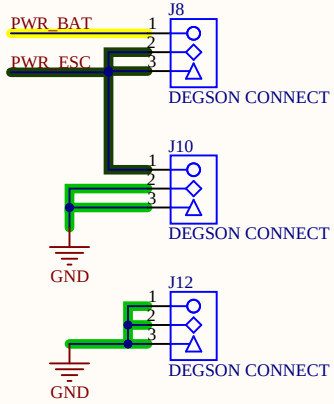
DATASHEET

Standard	（A）sust ained current	(10s) （A）max instantane ous current	(V/A) Output BEC	BEC mode	Number of battery/lit hium battery	Number of battery/ni ckel-hydrogen battery	mm Size(width*len gth*height)	g weight
HK20A UBEC	20A	25A	5.5V/3A-1A	SBEC/ UBEC	2-4Lipo	5-12NC	54*26*11	30
HK3A UBEC	30A	40A	5.5V/3A-1A	SBEC/ UBEC	2-4Lipo	5-12NC	54*26*11	32

LiPo Battery 2200 mAh

REAL MODEL

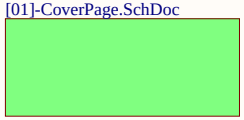
DEGSON



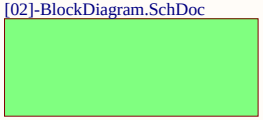
REAL MODEL

Designer's signature	Sheet title: **[07]-ESC-LIPO.SchDoc**		*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	Project title: **First Board Prototype**		
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		
	Date **12/06/2021**	Revision: **V04**	Sheet 7 of 8

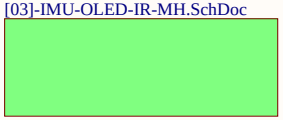
Designator
[01]-CoverPage.SchDoc



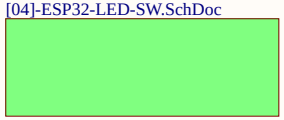
Designator
[02]-BlockDiagram.SchDoc



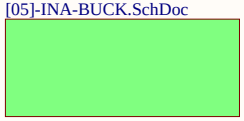
Designator
[03]-IMU-OLED-IR-MH.SchDoc



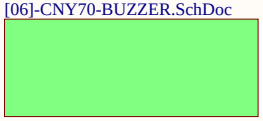
Designator
[04]-ESP32-LED-SW.SchDoc



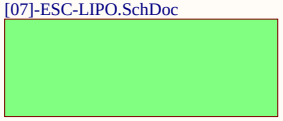
Designator
[05]-INA-BUCK.SchDoc



Designator
[06]-CNY70-BUZZER.SchDoc



Designator
[07]-ESC-LIPO.SchDoc



Designer's signature	Sheet title: **First_Prototype_Board.SchDoc**			*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	Project title: **First Board Prototype**			
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**			
	Date **12/06/2021**	Revision: **V04**	Sheet 8 of 8	

4.3 Stabilization Performance

4.3.1 Motor PWM Signal. Actuator.

How we need to implement the PWM signal was explained in subsubsection 3.2.2.1. Now we will explain how to implement this on code. For this job, ESP32 has a function called MCPWM. This allows us to design a PWM with a very specific frequency, being able to work as well with specific duty cycles.

```
1           mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, GPIO_PWM0A_OUT);
2           mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM1A, GPIO_PWM1A_OUT);
3           ...
4           mcpwm_config_t pwm_config;
5  pwm_config.frequency = 50;                              //frequency =
   50Hz,
6  pwm_config.cmpr_a = 0.0;                                   // (duty
   cycle) from PWMxA = 0
7  pwm_config.cmpr_b = 0.0;                                   // (duty
   cycle) from PWMxb = 0
8  pwm_config.counter_mode = MCPWM_UP_COUNTER;              //For MCPWM
   asymmetric
9  pwm_config.duty_mode = MCPWM_DUTY_MODE_0;                //duty cycle
   high
10          ...
11          mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config);
12          ...
13          mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_A,
   duty_cycle);
14          ...// FUNCTIONAL EXAMPLE
15          for(float i=5.0 ; i<=7.0 ; i+= 0.1){
16        brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, i);...}
17
```

Code 4.22 – *MCPWM Configuration*

In the former code (Code 4.22) we can see how the configuration of the frequency (line 4) must be made. This parameter is not very important, as we will work with the width, but we should keep it as close to 50 hz as possible. After that the configuration parameters present only ask for symmetric or asymmetric signal, how will the duty cycle behave in terms of HIGH or LOW signal, and the initializing of the units. As we saw in Figure 3.20, we will be using only one timer but three different operators, as the pairs must be programmed only for the motor to go backwards or forwards.

After the functional example comment we see how can we work with this function. Just by writing the duty cycle (with float precision), we will obtain a perfect PWM signal.

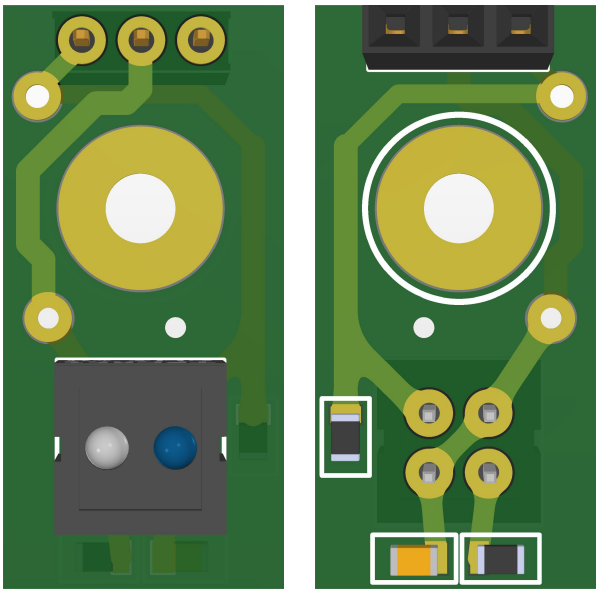
4.3.2 Sensor: Tachometer

4.3.2.1 Design

The CNY70 needs to be addressed with two external resistors in order to work properly. Resistor R_1 determines the maximum current through the emitter diode, while R_2 is an output resistance to make sure the output signal does not lose integrity. If we want the **maximum possible distance** between the CNY70 and the reflective surface we need a current of $20mA$

$$R_1 = \frac{V_{in} - V_{th}}{I_D} = \frac{3.3V - 1.25V}{20mA} = 102.5\Omega \cdot \tag{4.3.1}$$

Where V_{th} is the threshold voltage of the emitting diode.



(a) *Front part* **(b)** *Back part*

Figure 4.35 – *PCB 3D Model of the Tachometer*

In Figure 4.36.b at the bottom there are two resistors (SMD black) and a ceramic capacitor (SMD orange). The resistor in the left is the output resistor, R_2, which was decided to be $47k\Omega$.

4.3.2.2 Code Implementation

- First of all, we need to study the behavior between the sensor and the reflective surface (in our case, the inertial disk). As explained in Figure (Figure 3.12), the surface is black (low reflectiveness) with a white mark (high reflectiveness).
  Comparing the signal when positioning the sensor over the black surface and the

white mark, we are able to determine the thresholds (Code Code 4.23).

```
1           Serial.print("Lectura: ");
2           Serial.println(analogRead(ROLL_PIN));
3
```

Code 4.23 – *Analog reading from CNY70*

These program shows that when the sensor is over the dark color values vary between $1200 - 1400$, depending on the ambient brightness. On the other hand, the white (or even the original green color from the PLA material) mark gives out 4095 which is the maximum analog reading, hence the ADC from the ESP32 has a resolution of 12 bits.

Low Level Threshold	2000
High level threshold	3700

Table 4.1 – *CNY70 Reading Thresholds*

The thresholds leave enough clearance with the values read as a safety measure.

- Then we have to count how many times per minute the sensor overcomes the high threshold value, which means it has reached the white mark [15]. For this part we will translate the Equations (Equation 3.2.1,Equation 3.2.2)

```
1           thisReading_p = analogRead(PITCH_PIN);
2           if (thisReading_p > highLevel){
3             if (fallen_p == true){
4               thisTime_p = micros();
5               RPM_p = 60000000 / (thisTime_p - lastTime_p);
6               rads_p = (RPM_p*2*PI) / 60;
7                 ang_accel_p = (rads_p - rads_p_f) / (thisTime_p
  - lastTime_p);
8                 rads_p_f = rads_p;
9               lastTime_p = thisTime_p }}
10
```

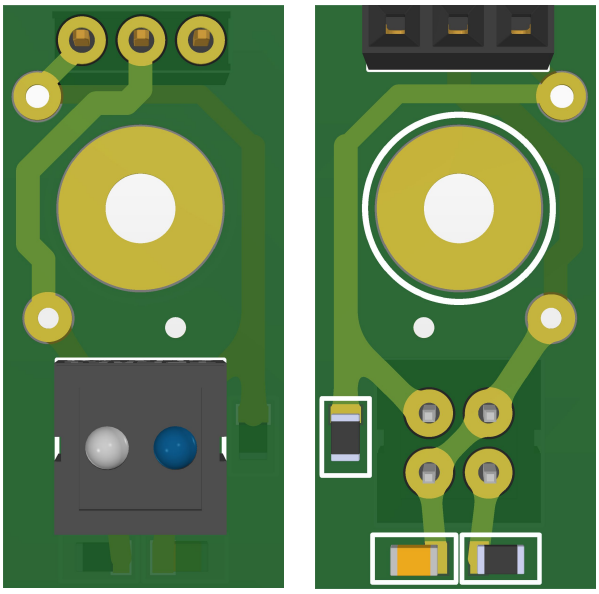
Code 4.24 – *Tachometer Code for RPM and Angular acceleration*

We can see clearly how in Code (Code 4.24) a flag ($fallen_p$) is activated when the sensor perceives the rising edge from the white mark.

- The last part is the acceleration and RPM calculation. This appears in lines 5, 6&7 in in Code (Code 4.24), computing the RPM, the angular velocity and the angular acceleration respectively.

4.3.3 PCB Design

The PCB designed for the tachometer meets the requirements; the necessary emitter diode current, a capacitor for filtering high frequencies to obtain a clean power supply and a female socket, where the connection between the motherboard and the tachometer PCB is made.



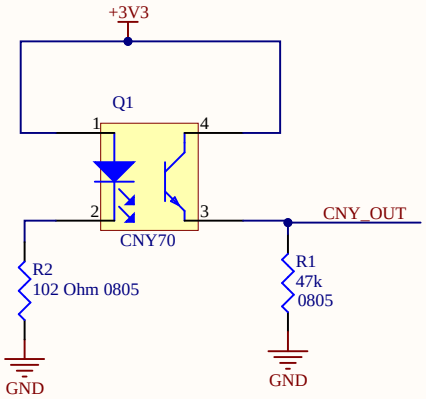
(a) *Front part* **(b)** *Back part*

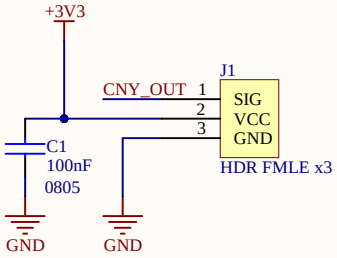
Figure 4.36 – *PCB 3D Model of the Tachometer*

In Figure 4.36.b at the bottom there are two resistors (SMD black) and a ceramic capacitor (SMD orange). The resistor in the left is the output resistor, R_2, which was decided to be $47k\Omega$.

For the next 3D pdf, if available, please use Adobe Acrobat Reader for 3D objects visualization.

CNY70

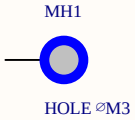




REAL MODEL

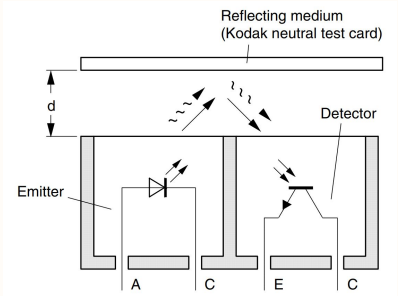
DATASHEET

	CNY70
COLLECTOR CURRENT	0.3 to 1 mA
DIODE FORWARD VOLTAGE	1.25 to 1.6 V
PEAK WAVELENGTH	940 nm
AMBIENT TEMPERATURE RANGE	-40 to + 85 ºC



DUDA
Debería poner el condensador contra altas frecuencias a la llegada del 3V3 en el female o cerquita del CNY70 tal y como está? SI, PORQUE ASÍ ELIMINAS POSIBLES INTERFERENCIAS A LA ENTRADA DE LA PLACA, DEJANDO LA SEÑAL LIMPIA. si la distancia fuese mayor, incluso deberían ponerse dos, uno a la entrada y otro lo más cercano posible al CNY70.

Emitting and receiving process:



In order to measure the spinning velocity of the inertial disks we have chosen the CNY70 to work as a tachometer.

IT contains an IR-emitting diode as a transmitter and a phototransistor as a receiver. The radiation emitted has a wavelength of 950nm, which then reflects over a bright surface (the best and most effective is white colour) and goes back to the phototransistor. This answer can be seen as a voltage change, passing from low to high.

Later on, knowing the period of time and how much of it has it been high, we can calculate the revolutions per minute (rpm) of our inertial disk, eventually calculating the angular velocity (rad/s).

Designer's signature	Sheet title: **CNY70 BOARD LAYOUT**		*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	Project title: **CNY70_board.PrjPcb**		
Supervisor's signature	Desginer: **Francisco José Llave Iglesias**		
	Date **12/06/2021**	Revision: **V03**	Sheet 1 of 1

4.3.3.1 Manufacturing & Soldering

The process undertaken is the same as in subsubsection 4.2.5.2. The resulting board for the CNY70 is as follows:

Figure 4.37 – *CNY70 PCB Manufactured*

Chapter 5

Validation

In this chapter we will determine the validation of our project. This verification is made through photos and tests. Software implementations and final assemblies working will show if the project has reached the expected goals and has met all the requirements.

5.0.1 Final Product

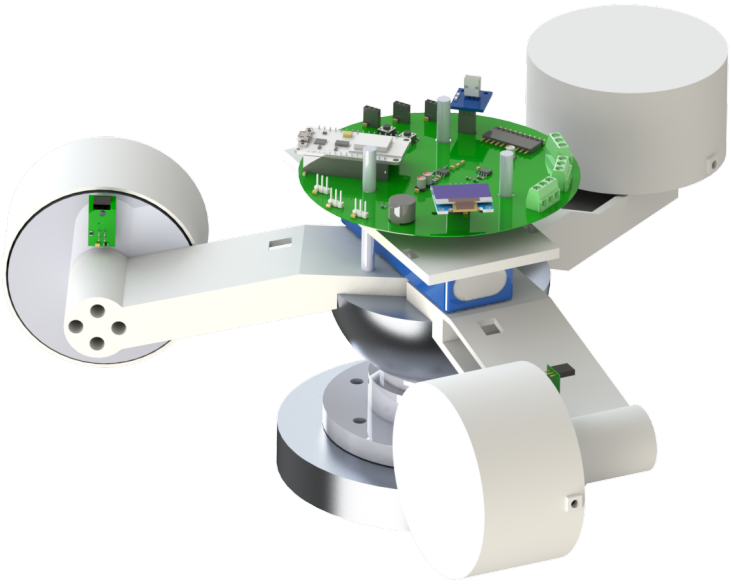


Figure 5.1 – *Render from the sphere and the pneumatic bearing system in SolidWorks*

5.0.2 Control Unit

5.0.2.1 Sensors and Control Unit

The next image shows the first prototype, with all of the necessary components and a switch for the battery connection. Disconnecting and connecting the battery every time could eventually damage the contacts, so this was the decision.

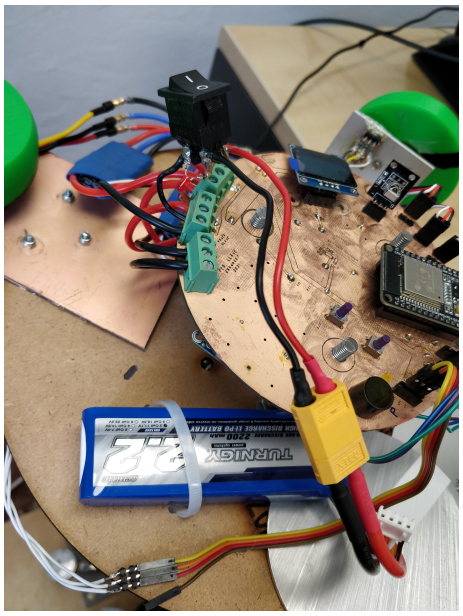


Figure 5.2 – *PCB Detail*

The final look of the system and the PCB, with a motor working, can be watched in this link video.

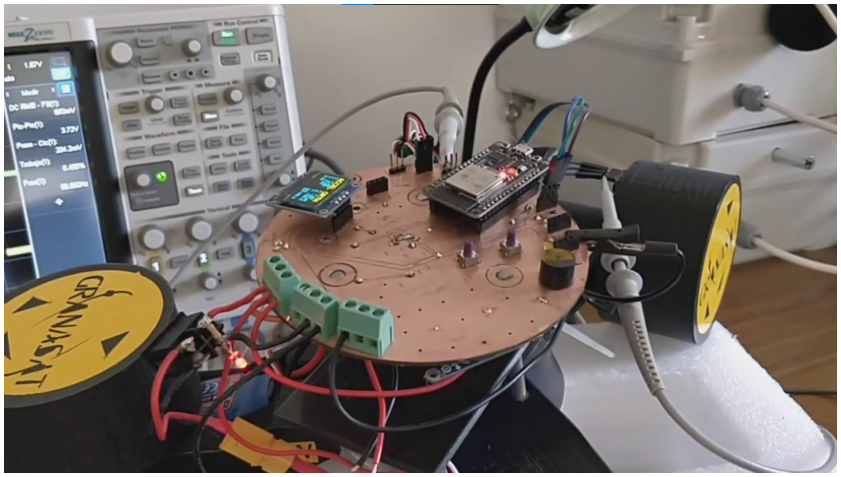


Figure 5.3 – *Extract of the Video from the Final Assembly*

As it can be seen, everything works perfectly. There is a problem with CNY70

measurements to be fixed, but it will be presented during the defense of this Thesis.

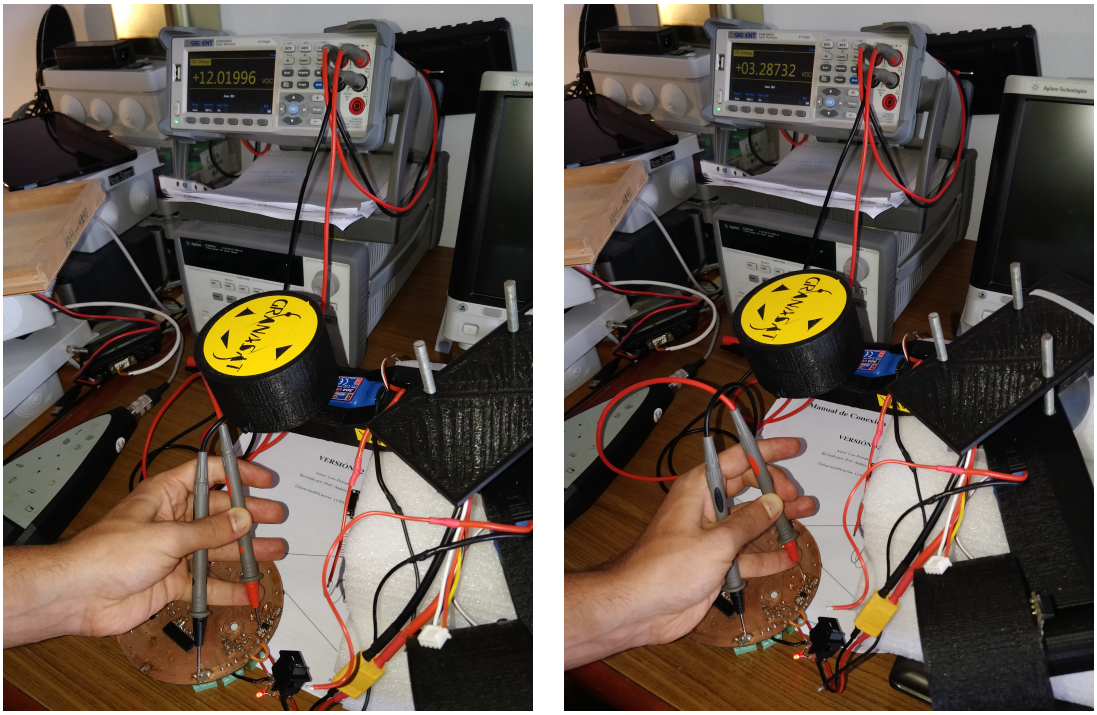
5.0.2.2 Communication

The communication working properly within the HTML interface can be seen in this link.

5.0.3 Power Regulation

As we already know, current must be controlled at all times. That is why we need to make sure the current consumption related to the Inertial Disk speed is below the maximum allowd. A current measurement can be watched in in this link.

We can also make sure that the buck regulator is working properly by measuring the voltage:



(a) *Voltage In* **(b)** *Voltage Out*

Figure 5.4 – *Review of the Buck*

5

Chapter 6

Conclusions & Future Lines

6.1 Conclusions

During this document we have had the pleasure of seen the intense development of an space project. This tough thesis has its foundations on many disciplines, and has been a real challenge for me. New scientific laboratory equipment, everyday challenges regarding mathematical problems, physical equations, new CAD programs I had never used, etc.

I think an experience like this should be mandatory for every engineer. This is as close as a senior project I am going to be in a very long time, so it was a pleasure.

6.2 Future Lines

In a project of these characteristics there is always something more you can add, or enhance. Here are some of the things I recommend doing or things that I was not capable of doing, due to time deadlines or complexity:

- Fix the gyroscope and accelerometer error with the Kalman Filter exposed. It needs patience and wisdom, but it is possible.

- Balance the simulation platform from the very beginning. Keeping the motors on to keep the platform balanced consumes a lot of energy, but for this you will need to calculate precise center of masses, as well as inertial momentum.

- Developing more advanced interfaces, maybe with voice commands or even with GPRS communication, so the need for a WiFi is not necessary any more.

Finally, I end up my degree studies with this sentence. It was a pleasure working

with laboratories, I hope we do not lose the contact.

The future awaits.

References

[1] Low pass filter - wikipedia. Wikipedia, The Free Encyclopedia., July 2005. https://commons.wikimedia.org/wiki/File:Butterworth_response.png.

[2] Kxg1205 datasheet, 2006. https://www.alldatasheet.es/datasheet-pdf/pdf/512368/ETC1/KXG1205.html.

[3] Optimizing power efficiency in point-of-load regulators using sllm (simple light load mode) control, 2009. https://www.rohm.com/documents/11303/41217/100554_BD91x_WP_V5.pdf/c345654d-1307-4a0d-a87d-f454c0a1b608?t=1459534856227.

[4] Combine accelerometer and gyroscope, 2010. https://stackoverflow.com/questions/1586658/combine-gyroscope-and-accelerometer-data.

[5] Reaction wheel actuated satellite dynamics test platform, 2014. https://os.mbed.com/users/DimitriGruebel/code/Reaction-Wheel-Actuated-Satellite-Dynami/.

[6] Reaction wheel actuated satellite dynamics test platform, 2014. https://www.youtube.com/watch?v=FidvWCDA_8w.

[7] Magnetic field detection with arduino and 3144, 2015. https://www.luisllamas.es/detectar-campos-magneticos-con-arduino-y-sensor-hall-a3144/.

[8] I2c communication protocol, 2016. https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/.

[9] Lcd adm1602k, 2016. https://www.iberobotics.com/producto/pantalla-lcd-16x2-1602-hd44780-display-green-backlight/.

[10] Ssd1306, 2016. https://nettigo.eu/products/oled-display-0-96-i2c-128x64-ssd1306-white.

[11] Freertos logo, 2018. https://en.wikipedia.org/wiki/FreeRTOS#/media/File:Logo_freeRTOS.png.

[12] Accelerometer and gyroscope issues, 2021. https://www.analog.com/en/analog-dialogue/raqs/raq-issue-139.html.

[13] Air bearing, 2021. https://en.wikipedia.org/wiki/Air_bearing.

[14] Brushless motor d2826-6, 2021. https://hobbyking.com/es_es/d2826-6-2200kv-outrunner-motor.html.

[15] Cny70 as a tachometer, 2021. https://github.com/fmilburn3/Tachometer_CNY70/blob/master/Tachometer_CNY70.ino.

[16] Cny70 vishay, 2021. https://es.rs-online.com/web/product/sensores-opticos-reflectantes/9190082/.

[17] Esc 30a motor driver, 2021. https://hobbyking.com/es_es/hobbyking-30a-2-4s-esc-3a-ubec.html.

[18] Esp32 dht11 dht22 web server temperature and humidity using arduino ide, 2021. https://randomnerdtutorials.com/esp32-dht11-dht22-temperature-humidity-web-server-arduino-ide/.

[19] Esp32 node mcu, 2021. https://www.amazon.com/-/es/nodemcu-32s-esp-32s-esp32-WiFi-Bluetooth-desarrollo/dp/B0769FZS5M.

[20] Esp32/esp8266: Control outputs with web server and a physical button simultaneously, 2021. https://randomnerdtutorials.com/esp32-esp8266-web-server-physical-button/.

[21] Euler angles, 2021. https://en.wikipedia.org/wiki/Euler_angles.

[22] Euler angles in mems, 2021. https://ez.analog.com/mems/w/documents/4124/adis16480-datasheet-euler-angle-reference.

[23] Euler plane image, 2021. https://www.analog.com/en/analog-dialogue/raqs/raq-issue-139.html.

[24] Gauss e imanes, 2021. https://www.supermagnete.es/faq/Disponen-de-un-iman-con-X-gauss.

[25] Gyroscope, 2021. https://en.wikipedia.org/wiki/Gyroscope.

[26] Gyroscope calibration, 2021. https://learn.adafruit.com/adafruit-sensorlab-gyroscope-calibration?view=all.

[27] Gyroscope mems, 2021. https://en.wikipedia.org/wiki/Vibrating_structure_gyroscope.

[28] Gyroscope mems example, 2021. https://www.asdreports.com/news-6900/key-players-global-mems-gyroscope-market-20152019.

[29] Hall sensor 3144, 2021. https://www.e-ika.com/sensor-de-efecto-hall-a3144-a3144e-oh3144e.

[30] Ina219 programming, 2021. https://www.best-microcontroller-projects.com/ina219.html#Using_the_simplest_equations.

[31] Introduction to rtos part 12 - multicore systems | digi-key electronics, 2021. https://www.youtube.com/watch?v=LPSHUcH5aQc&list=RDCMUCclJCqMDAkyVGsm5oFOTXIQ&index=12.

[32] Kxg1205, 2021. https://www.mpja.com/Active-Magnetic-Buzzer-12VDC/productinfo/30009%20SU/.

[33] Kxg1205 interior, 2021. https://www.cuidevices.com/product-spotlight/piezo-and-magnetic-buzzers.

[34] Ntm prop drive serie 28-30a, 2021. https://hobbyking.com/es_es/propdrive-v2-2830-1000kv-brushless-outrunner-motor.html?queryID=&objectID=59027&indexName=hbk_live_products_analytics.

[35] Passive speaker sound reproduction, 2021. https://www.luisllamas.es/reproducir-sonidos-arduino-buzzer-pasivo-altavoz/.

[36] Reaction wheel, 2021. https://en.wikipedia.org/wiki/Reaction_wheel.

[37] Reaction wheel in a cubesat, 2021. https://newspace-factory.com/product/comat-reaction-wheels-60/.

[38] Shawn hymel digikey profile, 2021. https://www.digikey.com/en/maker/profiles/72825bdd887a427eaf8d960b6505adac.

[39] Wifi issues with adc2 pins, 2021. https://github.com/espressif/arduino-esp32/issues/102.

[40] Allegro. *ESP32 Technical Reference Manual*, July 1999. https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

[41] Allegro. *Hall Sensor 3144 Datasheet*, July 1999. https://www.mpja.com/download/a3144eul.pdf.

[42] Inc., T. I. *7.0 V to 26.0 V Input, 1 A Integrated MOSFET Single Synchronous Buck DC/DC Converter*, December 2017. https://www.mouser.es/datasheet/2/348/bd9e104fj-e-1874366.pdf.

[43] Inc., T. I. *LMR33610 SIMPLE SWITCHER 3.8 V to 36 V, 1 A Synchronous Step down Converter*, October 2019. https://www.ti.com/lit/ds/symlink/lmr33610.pdf?HQS=dis-dk-null-digikeymode-dsf-pf-null-wwe&ts=1616580646121.

[44] Instruments, T. *INA219 Zero Drift, Bidirectional Current Power Monitor With I2C Interface*, December 2015. https://www.ti.com/lit/ds/symlink/ina219.pdf?ts=1623301919611&ref_url=https%253A%252F%252Fwww.google.com%252F.

[45] Molina, J. P. Sistema de simulacion para pruebas de algoritmos deorientacion y control de satelites pequenos. Final masters project, Universidad Nacional Autonoma de Mexico, Mexico D.F., 2007.

[46] QIU, c., and ye. *LCD ADM1602K Datasheet*, October 2008. https://www.sparkfun.com/datasheets/LCD/ADM1602K-NSW-FBS-3.3v.pdf.

[47] ST. *Gyroscope Datasheet*, February 2013. https://www.pololu.com/file/0J563/L3GD20.pdf.

[48] Stark, K. W. *The Design of Various Types of Air Bearing for Stimulating Frictionless Environments*, May 1962. https://play.google.com/books/reader?id=rMtLWP8o5vUC&hl=es&printsec=frontcover&pg=GBS.PA2.

[49] Systech, S. *SSD1306 Datasheet*, April 2008. https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf.

[50] Vishay. *CNY70 Datasheet*, July 2012. https://www.vishay.com/docs/83751/cny70.pdf.

[51] Young, H. D., and Freedman, R. A. *Sears y Zemansky Fisica Universitaria Vol.1*, thirteenth ed. Physics. Pearson, Mexico, 2013.

Appendix A

Kalman Filter

The purpose of this chapter is to explain the Kalman filter and to make a practical approach to our system.

A.1 Technical Description

This filter is a recursive filter that estimates the internal state of a linear dynamic system from a series of noisy measurements. Altogether with the linear quadratic regulator (LQR), it solves the linear quadratic Gaussian control problem (LQG).

A.2 Kalman Discrete Filter Approach for C Implementation

http://blog.tkjelectronics.dk/2012/09/a-practical-approach-to-kalman-filter-and-how-to-implement-it/
https://github.com/TKJElectronics/Example-Sketch-for-IMU-including-Kalman-filter

C implementation of the equations calculated from the web page of TKJ Electronics by Kristian Lauszus

- Step One

$$\hat{\boldsymbol{x}}_{k|k-1} = \boldsymbol{F}\hat{x}_{k-1|k-1} + \boldsymbol{B}\dot{\theta}_k$$

$$\begin{aligned}
\begin{matrix} \theta \\ \dot{\theta}_b \end{matrix}\Big]_{k|k-1} &= \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k \\
&= \begin{bmatrix} \theta - \dot{\theta}_b \Delta t \\ \dot{\theta}_b \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} \Delta t \\ 0 \end{bmatrix} \dot{\theta}_k \\
&= \begin{bmatrix} \theta - \theta_b \Delta t + \theta \Delta t \\ \dot{\theta}_b \end{bmatrix}
\end{aligned}$$

which can be written as

```
        rate = newRate - bias;
angle += dt * rate;

```

- Step Two

$$\boldsymbol{P}_{k|k-1} = \boldsymbol{F}\boldsymbol{P}_{k-1|k-1}\boldsymbol{F}^t + \boldsymbol{Q}_k$$

$$\begin{aligned}
\begin{matrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{matrix}\Big]_{k|k-1} &= \begin{bmatrix} 1 & -\Delta t \\ 0 & 1 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \\
&= \begin{bmatrix} P_{00} - \Delta t P_{10} & P_{01} - \Delta t P_{11} \\ P_{10} & P_{11} \end{bmatrix}_{k-1|k-1} \begin{bmatrix} 1 & 0 \\ -\Delta t & 1 \end{bmatrix} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \\
&= \begin{bmatrix} P_{00} - \Delta t P_{10} - \Delta t \left(P_{01} - \Delta t P_{11}\right) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} \end{bmatrix}_{k-1|k-1} + \begin{bmatrix} Q_\theta & 0 \\ 0 & Q_{\dot{\theta}_b} \end{bmatrix} \Delta t \\
&= \\
&\begin{matrix} P_{00} - \Delta t P_{10} - \Delta t \left(P_{01} - \Delta t P_{11}\right) + Q_\theta \Delta t & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{matrix} \\
&= \begin{bmatrix} P_{00} + \Delta t \left(\Delta t P_{11} - P_{01} - P_{10} + Q_\theta\right) & P_{01} - \Delta t P_{11} \\ P_{10} - \Delta t P_{11} & P_{11} + Q_{\dot{\theta}_b} \Delta t \end{bmatrix}
\end{aligned}$$

which could be transcripted as:

```
        P[0][0] += dt * (dt*P[1][1] - P[0][1] - P[1][0] + Q_angle
 );
P[0][1] -= dt * P[1][1];
P[1][0] -= dt * P[1][1];
P[1][1] += Q_gyroBias * dt;

```

- Step Threeç

$$\begin{aligned} \boldsymbol{y}_k &= \boldsymbol{z}_k - \boldsymbol{H}\hat{x}_{k|k-1} \\ &= \boldsymbol{z}_k - \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} \\ &= \boldsymbol{z}_k - \theta_{k|k-1} \end{aligned}$$

The innovation variable could be transcripted into

```
        y = newAngle - angle;

```

- Step Four

$$\begin{aligned} \boldsymbol{S}_k &= \boldsymbol{H}\boldsymbol{P}_{k|k-1}\boldsymbol{H}^T + \boldsymbol{R} \\ &= \begin{bmatrix} 1 & 0 \end{bmatrix} \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \boldsymbol{R} \\ &= P_{00k|k-1} + R \\ &= P_{00k|k-1} + \mathrm{var}(v) \end{aligned}$$

where the covariance could be implemented like

```
        S = P[o][o] + R_measure;

```

- Step Five

$$\begin{aligned} \boldsymbol{K}_k &= \boldsymbol{P}_{k|k-1}\boldsymbol{H}^T\boldsymbol{S}_k^{-1} \\ \begin{matrix} K_0 \\ K_1 \end{matrix}\Big]_k &= \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} \begin{bmatrix} 1 \\ 0 \end{bmatrix} \boldsymbol{S}_k^{-1} \\ &= \begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1} \boldsymbol{S}_k^{-1} \\ &= \frac{\begin{bmatrix} P_{00} \\ P_{10} \end{bmatrix}_{k|k-1}}{\boldsymbol{S}_k} \end{aligned}$$

This control matrix calculation is not intended in every condition. This is recommended for situations when the Kalman prediction must be under certain conditions, not only the already considered variables.

```
        K[o] = P[o][o] / S;
K[1] = P[1][o] / S;

```

- Step Six

$$\hat{\boldsymbol{x}}_{k|k} = \hat{\boldsymbol{x}}_{k|k-1} + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k$$

$$\begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k} = \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \tilde{\boldsymbol{y}}_k$$

$$= \begin{bmatrix} \theta \\ \dot{\theta}_b \end{bmatrix}_{k|k-1} + \begin{bmatrix} K_0 \tilde{\boldsymbol{y}} \\ K_1 \tilde{\boldsymbol{y}} \end{bmatrix}_k$$

```
      angle += K[0] * y;
bias += K[1] * y;

```

- Step Seven

$$\boldsymbol{P}_{k|k} = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H})\, \boldsymbol{P}_{k|k-1}$$

$$\begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 \\ K_1 \end{bmatrix}_k \begin{bmatrix} 1 & 0 \end{bmatrix} \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

$$= \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_0 & 0 \\ K_1 & 0 \end{bmatrix}_k \right) \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1}$$

$$= \begin{bmatrix} P_{00} & P_{01} \\ P_{10} & P_{11} \end{bmatrix}_{k|k-1} - \begin{bmatrix} K_0 P_{00} & K_0 P_{01} \\ K_1 P_{00} & K_1 P_{01} \end{bmatrix}$$

```
      float Poo_temp = P[0][0];
float Po1_temp = P[0][1];

P[0][0] -= K[0] * Poo_temp;
P[0][1] -= K[0] * Po1_temp;
P[1][0] -= K[1] * Poo_temp;
P[1][1] -= K[1] * Po1_temp;

```

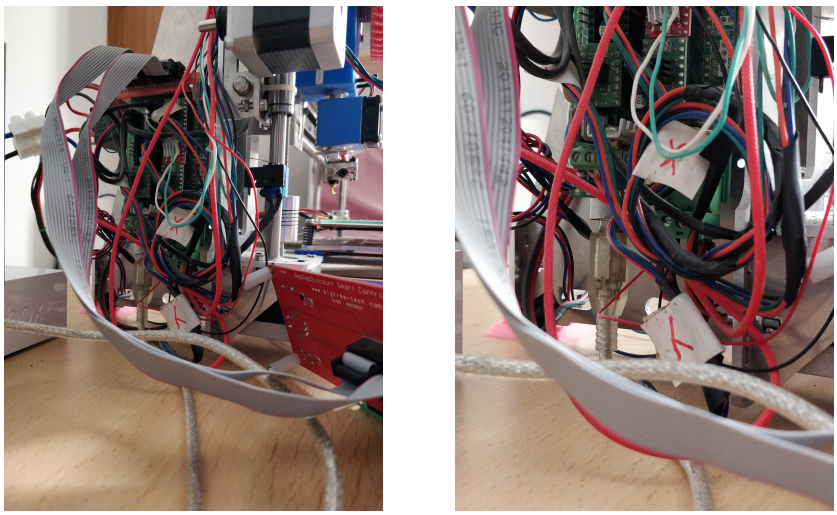
It is important to set the target angle at startup if you need to use the output.

Appendix B

3D Printer Initialization and Calibration Guide

B.1 Initialization

- Turn on the 3D printer. The power source has a **white switcher** where you plug it on and off.
- Once the 3D printer is on, plug the USB-A wire (Figure B.1) in to establish communication between the computer and the printer.



(a) *Normal View* **(b)** *Zoom View*

Figure B.1 – *USB-A plug in*

The program used to print and control the printer is Simplify 3D

- It is paramount to have the necessary drivers installed. In this case, as the 3D printer uses an Arduino, **CH340** Drivers are the ones we need. After the installation we can proceed with the USB connection to the computer. To make sure that our device has been connected successfully, we can go to Device Manager and check if the CH340 appears on the list.

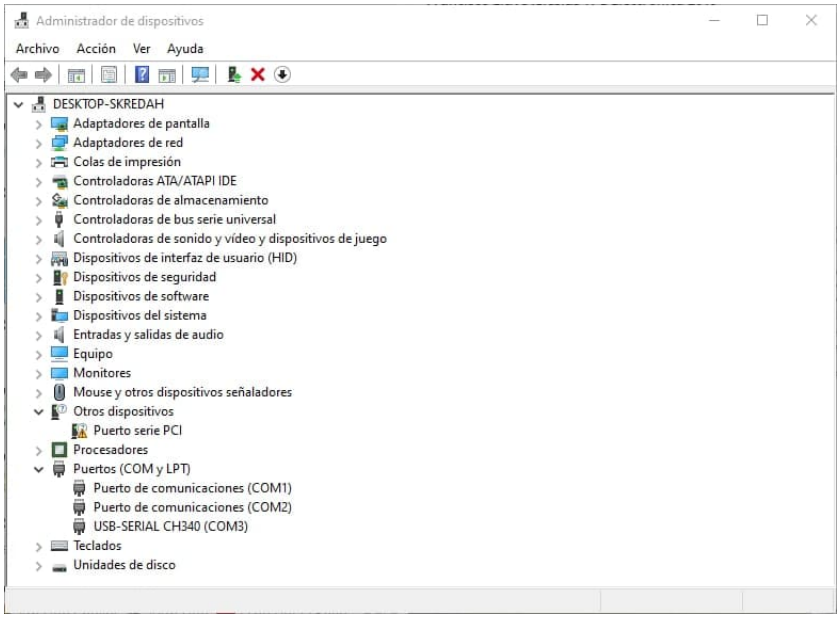


Figure B.2 – *Device Manager Window*

- Having finished with the drivers installation and the plug of our 3D printer, we proceed to establish the connection with our Machine Control Panel, present in the **Tools** tab in the program. The connection must be made at a **baud rate** of $\mathbf{115200}\frac{\mathbf{bits}}{\mathbf{sec}}$ as the program in the Arduino is designed to work at that speed. If the SENT and READ commands are the same as the ones that appear on the figure Figure B.3 **the communication protocol is working successfully**.

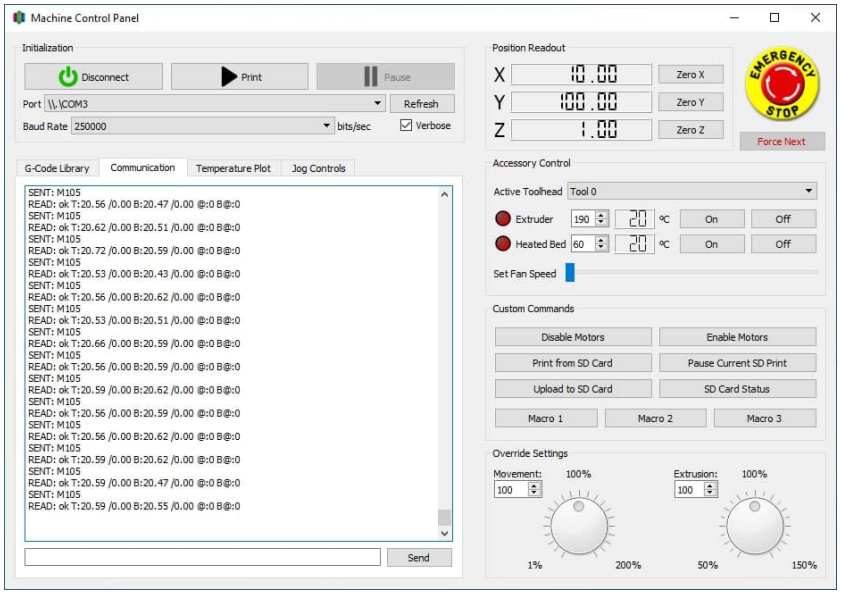


Figure B.3 – *Communication Tab in the Machine Control Panel*

B.2 Calibration

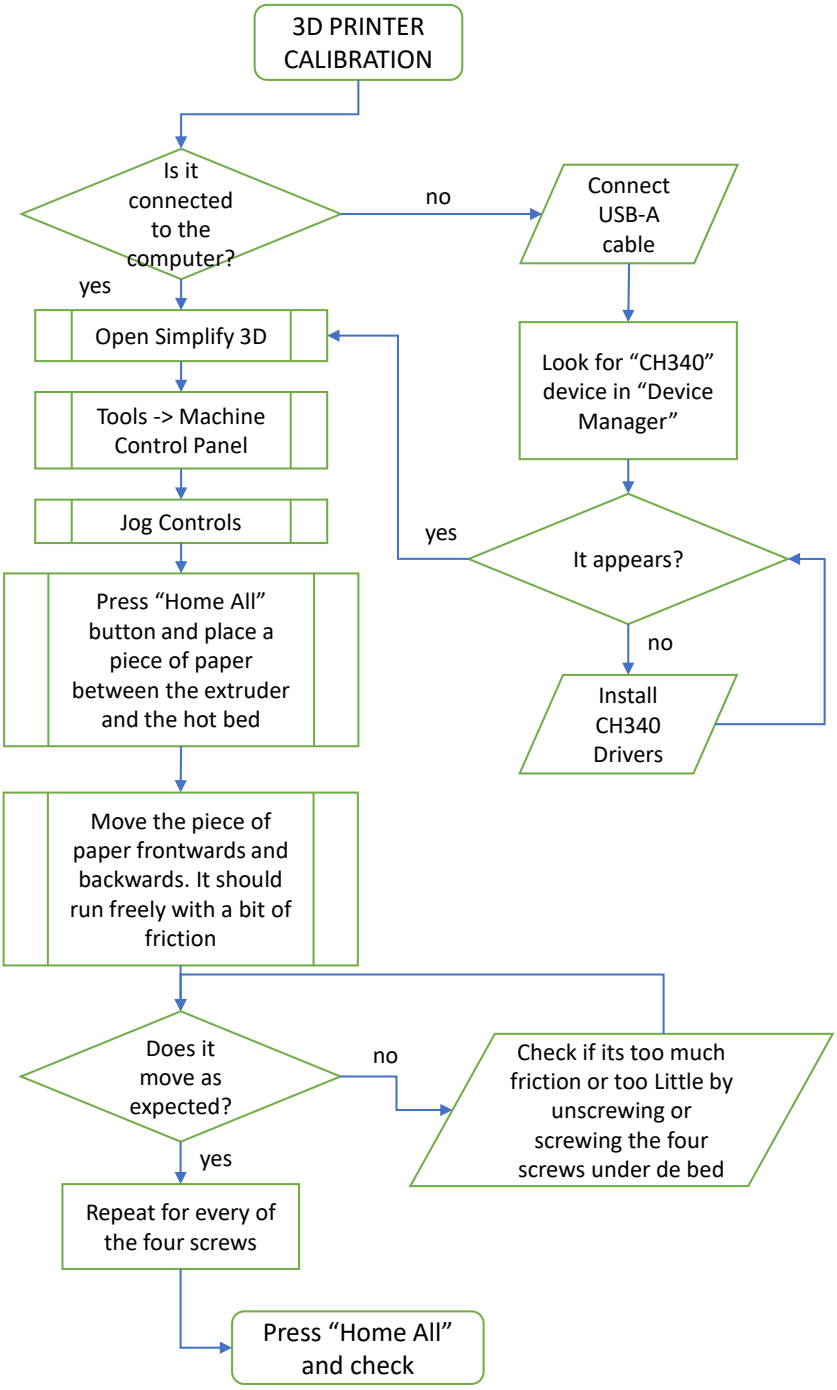


Figure B.4 – *Calibration Diagram for the 3D printer*

- Next to the **Communication** tab, we have the **Jog Controls** tab. If you click on it a window like the one in figure Figure B.5 will appear.

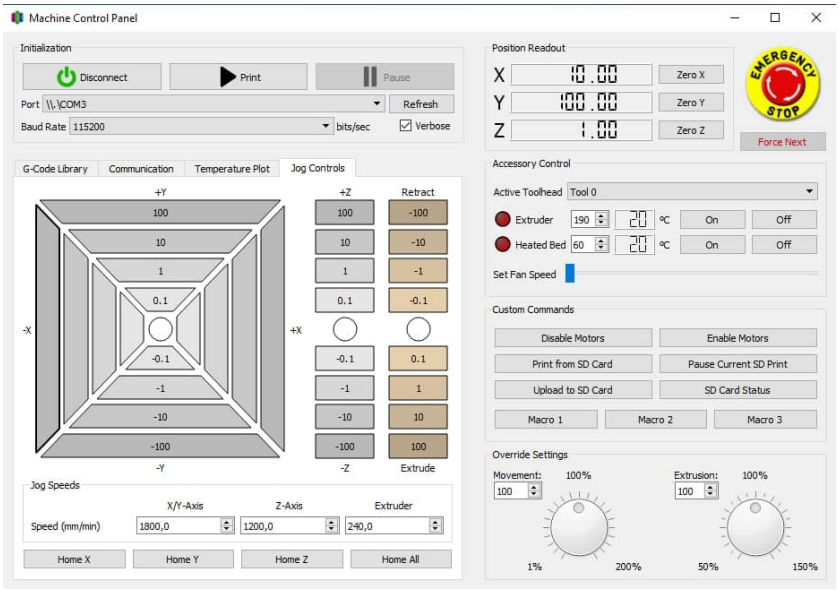
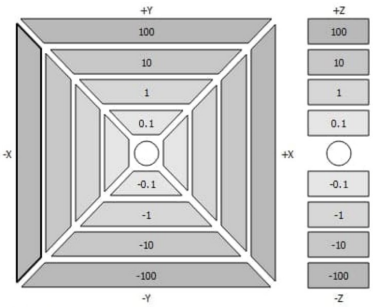


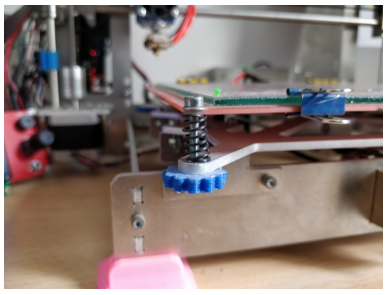
Figure B.5 – *Jog Controls tab in the Machine Control Panel*

Our main purpose in this window is to calibrate the bed. We will do it following a series of steps:

1. **Make sure the Z position is a little bit higher upon the bed**, you could scratch it if the extruder is touching the bed while moving it along.
2. Move the extruder along to each and every one of the four corners with the help from the X and Y coordinates buttons (**??**).



(a) *Coordinates for Jog Controls*



(b) *Corner screws*

3. If at **zero Position Readout** (upper right textbox) at the Z coordinate the extruder does not slightly touch the bed, you will have to **screw (gets the bed down) or unscrew (makes the bed to go up)** (**??**) depending on the **hardness of the paper while slipping** between the extruder and the bed. This seems quite subjective but it is the fastest way. Simply make sure that the paper does **not oppose very much** to slipping.
4. **WARNING**: If for any reason, when you press **Home All** button but the extruder pushes to hard onto the bed or just leaves a big space with the bed, it may be the **lever type switch** that sometimes moves. Just move it very

carefully (first try rotating it, that could be enough to fix it) until the **Home All** button takes the extruder to a proper position.

- Once all of this has been done, the 3D printer is ready for printing. Have in mind that if you are printing something with steep walls it is recommended to start the printing at **slow speed**. This gives the plastic time to **cool down** so it does not affect the printing. 65% speed should be enough.

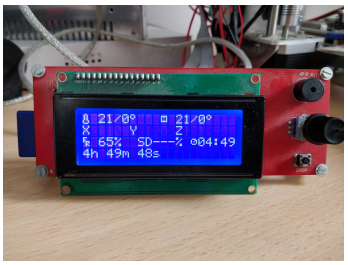


Figure B.6 – *LCD from the 3D printer*

Appendix C

Orientation: A Mathematical Approach

C.1 Rotations & Space Transformation

C.1.1 Change of coordinates in a vector space

Suppose we have a vector space Y upon $\mathbb{R}$ of n dimension.

Letters like a will be used to denote *arrays* in $\mathbf{Y}$. Given an $\mathbf{Y}$ base $\mathbf{v_1}, \cdots, \mathbf{v_n}$ every array $\mathbf{w} \in \mathbf{Y}$ can be represented uniquely as a linear combination:

$$\mathbf{w} = \alpha_1 v_1 + \cdots + \alpha_n v_n. \tag{C.1.1}$$

Column array $(\mathbf{ff_1}, \cdots, \mathbf{ff_n})^{\mathbf{T}} \in \mathbb{R}^{\mathbf{n}}$ is the w coordinate array in the ordered base $\mathbf{V} = (\mathbf{v_1}, \cdots, \mathbf{v_n})$, denoted by $[\mathbf{w}]_{\mathbf{V}}$

$$[\mathbf{w}]_{\mathbf{V}} = \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix}.$$

If we can see formally the base $\mathbf{V} = (\mathbf{v_1}, \cdots, \mathbf{v_n})$ as a row array ($\mathbf{v_j}$ arrays), then Equation C.1.1 can be rewritten as

$$\mathbf{w} = (v_1, \cdots, v_n) \begin{pmatrix} \alpha_1 \\ \vdots \\ \alpha_n \end{pmatrix} = \mathbf{V}[\mathbf{w}]_{\mathbf{V}}.$$

Observation C.1.1.1 *If $\mathbf{Y} = \mathbb{R}^n$, each array from base $\mathbf{v_j}$ is at the same time a column array, so*

in that space $\mathbf{V}$ *it is in fact, a* $n \times n$*.*

Let's suppose this time we have another ordered base, $\mathbf{U} = (\mathbf{u_1}, \cdots, \mathbf{u_n})$*, from* $\mathbf{Y}$*, and we want to determine the coordinates array* $[\mathbf{w}]_\mathbf{U}$ *from* $\boldsymbol{w}$ *in the new base* $\boldsymbol{U}$

$$\mathbf{U}[\mathbf{w}]_\mathbf{U} = \mathbf{w} = \mathbf{V}[\mathbf{w}]_\mathbf{V}.$$

But each and every element $\mathbf{v_j}$ *from the former* $\boldsymbol{V}$ *base is an element which belongs to* $\mathbf{Y}$*, and* $\boldsymbol{U}$ *is a base which also belongs to* $\mathbf{Y}$*, so as expected*

$$\mathbf{v_1} = \mathbf{U}[\mathbf{v_1}]_\mathbf{U}, \cdots, \mathbf{v_n} = \mathbf{U}[\mathbf{v_n}]_\mathbf{U},$$

which can be merged into a single expression

$$\mathbf{V} = \mathbf{U}\left([\mathbf{v_1}]_\mathbf{U}, \cdots, [\mathbf{v_n}]_\mathbf{U}\right). \tag{C.1.2}$$

Notice that no matter how abstract the vector space $\mathbf{Y}$ *is, the expression*

$$\left([\mathbf{v_1}]_\mathbf{U}, \cdots, [\mathbf{v_n}]_\mathbf{U}\right)$$

is a $\mathbb{R}^{n\times n}$ *matrix, where column* j *is the coordinates array from* $\mathbf{v_j}$ *array, coming from the former base but which belongs in the new base* $\boldsymbol{U}$*. This results in the Base Change Matrix*

$$\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}} = \left([\mathbf{v_1}]_\mathbf{U}, \cdots, [\mathbf{v_n}]_\mathbf{U}\right).$$

With the former notation, identity (??) can be written as

$$\mathbf{V} = \mathbf{U}\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}}.$$

Rearranging former identities into a chain

$$\mathbf{w} = \mathbf{V}[\mathbf{w}]_\mathbf{V} = \left(\mathbf{U}\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}}\right)[\mathbf{w}]_\mathbf{V} = \mathbf{U}\left(\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}}[\mathbf{w}]_\mathbf{V}\right).$$

It has been obtained the expression for the base swap

$$\boxed{[\mathbf{w}]_\mathbf{U} = \mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}}[\mathbf{w}]_\mathbf{V}}$$

C.1.2 Change of coordinates in Euclidean space

Suppose $\mathbf{Y}$ is an Euclidean space which means the dot product $\langle u, v\rangle$ between two arrays $\mathbf{u}, \mathbf{v} \in \mathbf{Y}$ is defined. I.e. the standard definition of dot product in $\mathbb{R}^n$ is

$$\mathbf{u} = \begin{pmatrix} u_1 \\ \vdots \\ u_n \end{pmatrix}, \mathbf{v} = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} \Rightarrow \langle u, v \rangle = u \cdot v = u^T v = \sum_{j=1}^{n} u_j v_j.$$

In an Euclidean space it is paramount to mention the **orthogonal** arrays (where $\mathbf{u} \cdot \mathbf{v} = \mathbf{0}$) and orthonormal arrays (where $\|u\| = \|v\| = 1$, being $\|u\| = \sqrt{\langle u, u \rangle}$). As an orthonormal base, $\mathbf{U} = (\mathbf{u_1}, \cdots, \mathbf{u_n})$ consists of orthonormal arrays, which makes it easy to calculate the coordinates $[\mathbf{w}]_\mathbf{U}$ for whatever array $\mathbf{w} \in \mathbf{Y}$

$$\begin{aligned} \mathbf{w} = \alpha_1 v_1 + \cdots + \alpha_n v_n \quad &\Rightarrow \alpha_j = \mathbf{w} \cdot \mathbf{u_j}, \quad j = 1, \cdots, n \\ &\Rightarrow [w]_U = \begin{pmatrix} \langle w, v_1 \rangle \\ \vdots \\ \langle w, v_n \rangle \end{pmatrix}. \end{aligned}$$

Let's assume we have two ordered bases, ***V and U***, where ***U*** is *orthonormal* and we want to find the change of coordinates matrix from base ***V*** to base ***U***. As we have just discussed,

$$\mathbf{T_{U \leftarrow V}} = ([\mathbf{v_1}]_\mathbf{U}, \cdots, [\mathbf{v_n}]_\mathbf{U}) = \begin{pmatrix} \langle v_1, u_1 \rangle & \langle v_2, u_1 \rangle & \cdots & \langle v_n, u_1 \rangle \\ \vdots & \vdots & \ddots & \vdots \\ \langle v_1, u_n \rangle & \langle v_2, u_n \rangle & \cdots & \langle v_n, u_n \rangle \end{pmatrix}.$$

The former matrix, whose element *(i,j)* is equal to $\langle \mathbf{v_i}, \mathbf{u_j} \rangle$, is called *Gram's Matrix* and it is symmetrical. Any two vectors ***u*** and ***v*** from $\mathbf{Y}$,

$$\cos \angle(\mathbf{u}, \mathbf{v}) = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\| \|\mathbf{v}\|},$$

so if both bases ***V and U*** consist of unitary vectors (and ***U*** is orthonormal), then that matrix can be geometrically described as

$$\mathbf{T_{U \leftarrow V}} = \begin{pmatrix} \cos \angle(v_1, u_1) & \cos \angle(v_2, u_1) & \cdots & \cos \angle(v_n, u_1) \\ \vdots & \vdots & \ddots & \vdots \\ \cos \angle(v_1, u_n) & \cos \angle(v_2, u_n) & \cdots & \cos \angle(v_n, u_n) \end{pmatrix}. \tag{C.1.3}$$

This former matrix is known in engineering as **DCM**, which means *direction cosine matrix*. Therefore, it is a change of base matrix from an orthonormal base to another base with unitary vectors (these may or may not be orthogonals).

If base ***V*** were orthonormal, then $\mathbf{T_{U \leftarrow V}}$ would be an orthogonal matrix (its inverse

matrix equals its transpose matrix)

$$\begin{aligned}\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}} &= \begin{pmatrix}\langle v_i, u_1\rangle \\ \vdots \\ \langle v_i, u_n\rangle\end{pmatrix} \cdot \begin{pmatrix}\langle v_j, u_1\rangle \\ \vdots \\ \langle v_j, u_n\rangle\end{pmatrix} = \sum_{k=1}^{n}\langle v_i, u_k\rangle\langle v_j, u_k\rangle = \\ &= \sum_{k=1}^{n}\langle v_j, \langle v_i, u_k\rangle u_k\rangle = \langle v_j, \sum_{k=1}^{n}\langle v_i, u_k\rangle u_k\rangle = \langle v_j, v_i\rangle = \begin{cases}0, & \text{if } i \neq j \\ 1, & \text{if } i = j\end{cases}\end{aligned} \tag{C.1.4}$$

In $\mathbb{R}^n$, the coordinates transformation which keeps the orthonormality of the bases is an space rotation (Euler's Theorem). For this particular case, the DCM are in fact *rotation matrices*. For $\mathbf{Y} = \mathbb{R}^n$, DCM look like

$$\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}} = \begin{pmatrix}\cos\angle(v_1, u_1) & \cos\angle(v_2, u_1) & \cos\angle(v_3, u_1) \\ \cos\angle(v_1, u_2) & \cos\angle(v_2, u_2) & \cos\angle(v_3, u_2) \\ \cos\angle(v_1, u_3) & \cos\angle(v_2, u_3) & \cos\angle(v_3, u_3)\end{pmatrix}.$$

Adopting every $\angle(v_i, u_i)$ is **small**, a reasonable approximation would be

$$\cos\angle(v_i, u_i) \approx 1.$$

That means the rest of the angles

$$\angle(v_i, u_i) \approx \frac{\pi}{2}, \quad if i \neq j$$

If we say

$$\angle(v_2, u_1) = \frac{\pi}{2} - \theta_{12}, \quad (\theta_{12} \approx 0) \quad \Rightarrow \cos\angle(v_2, u_1) = \sin\theta_{12} \approx \theta_{12}.$$

and on the other hand,

$$\angle(v_1, u_2) = \frac{\pi}{2} + \theta_{21}, \quad (\theta_{21} \approx 0) \quad \Rightarrow \cos\angle(v_1, u_2) = -\sin\theta_{21} \approx -\theta_{21}.$$

Doing the same with the rest of the matrix elements we come up with the fact that *in case of a disturbance from the primal orthonormal system $\boldsymbol{V}$, which keeps the unitary normals of the base (without necessarily maintaining orthogonality)*, the DCM could be approximated by the following expression

$$\mathbf{T}_{\mathbf{U}\leftarrow\mathbf{V}} \approx \begin{pmatrix}1 & \theta_{12} & -\theta_{13} \\ -\theta_{21} & 1 & \theta_{23} \\ \theta_{31} & -\theta_{32} & 1\end{pmatrix} \equiv \mathbf{I} + \begin{pmatrix}0 & \theta_{12} & -\theta_{13} \\ -\theta_{21} & 0 & \theta_{23} \\ \theta_{31} & -\theta_{32} & 0\end{pmatrix} \tag{C.1.5}$$

Where I is the identity matrix.

C.1.3 Euler angles

Rotation arrays are easy to use, but its elements seem difficult to read. Let's narrow it down to $\mathbb{R}^3$. Euler claimed that every rotation in $\mathbb{R}^3$ can be obtained as a composition from three consecutive elementary rotations, just by choosing the order from the coordinates axis and rotating around that specific axis, in a consecutive way. The three respective angles shape the *Euler's angles*. There are $3! = 12$ ways of arranging 3 axis, each space rotation has 12 differents triplets of *Euler's angles*, which is a drawback.

However, if we happen to know the *Euler's angles*, DCM is easy to get, just with the consecutive product between the *elementary rotation matrices* (ERM). I.e., a rotation around the axis OX an angle θ_x is determined by its DCM

$$\mathbf{R_x} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & \sin\theta_x \\ 0 & -\sin\theta_x & \cos\theta_x \end{pmatrix}.$$

Likewise, the matrices whose rotation around the OY axis (OZ resp.,) with an angle of θ_y (θ_z resp.,)

$$\mathbf{R_y} \approx \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & 0 \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix} \quad \& \quad \mathbf{R_z} \approx \begin{pmatrix} \cos\theta_z & \sin\theta_z & 0 \\ -\sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix},$$

Then, a transformation conducting a rotation of $\cos\theta_x$ angle around the $\boldsymbol{OX}$ axis first, followed by a rotation of $\cos\theta_y$ angle around the new $\boldsymbol{OY}$ axis, and eventually a rotation of $\cos\theta_z$ angle around the new $\boldsymbol{OZ}$, would lead us to the DCM

$$\mathbf{R} = \mathbf{R_x}\mathbf{R_y}\mathbf{R_z}.$$

Assuming the θ rotation angles are small, a reasonable approximation would be $\cos\theta \approx 1, \sin\theta \approx \theta$

$$\mathbf{R_x} \approx \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & \theta_x \\ 0 & -\theta_x & 1 \end{pmatrix}, \quad \mathbf{R_y} \approx \begin{pmatrix} 1 & 0 & \theta_y \\ 0 & 1 & 0 \\ -\theta_y & 0 & 1 \end{pmatrix} \quad \& \quad \mathbf{R_z} \approx \begin{pmatrix} 1 & \theta_z & 0 \\ -\theta_z & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}.$$

Performing the product between $R_x R_y R_z$ and rejecting the terms of higher order (trivial), we have

$$\mathbf{R} = \begin{pmatrix} 1 & \theta_z & \theta_y \\ -\theta_z & 1 & \theta_x \\ -\theta_y & -\theta_x & 1 \end{pmatrix}.$$

This result could be seen graphically for our MEMS purpose this way:

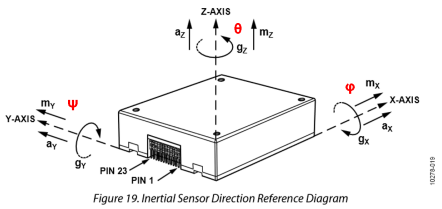


Figure 19. Inertial Sensor Direction Reference Diagram

Figure C.1 – *Euler in IMU*

Appendix D

Main Program Code

```
/*
* Programmers:
*   Prof. Andrés Roldán       01/03/2019 (amroldan@ugr.es)
*   Francisco Llave Iglesias 11/02/2021 (francisco.
llaveiglesias@gmail.com)
*
*   Uso:
*
*    Esta versión inicial de uso del IMU 10 DOF  https://www.
adafruit.com/product/1604
*    que tiene disponible:
*        L3GD20H 3-axis gyroscope:  250 ,  500 , or  2000  degree-
per-second scale
*        LSM303 3-axis compass:  1 .3 to  8 .1 gauss magnetic field
scale
*        LSM303 3-axis accelerometer:  2g / 4g / 8g / 16g
selectable scale
*        BMP180 barometric pressure/temperature: -40 to 85  C , 300
- 1100hPa range, 0.17m resolution
*
*   Scanning... I2C devices:  Estos son los dispositivos encontrados
.
*   I2C device found at address 0x19  !     #define
LSM303_ADDRESS_ACCEL   (0x32 >> 1)  // 0011001x
*   I2C device found at address 0x1E  !     #define
LSM303_ADDRESS_MAG     (0x3C >> 1)  // 0011110x
*   I2C device found at address 0x69  !
*   I2C device found at address 0x77  ! BMP085_ADDRESS
*   I2C device found at address 0x3C  ! SSD1306_ADDRESS
*   done
*
* Versions:
*
```

```
25  *  V05:  13/06/2021
26  *    WiFi with Interface implementation through HTML, CSS and
    JavaScript programming
27  *
28  *  V04:  20/05/2021
29  *    Including the MCPWM commands, altogether with the RTOS
    configuration xTasks, which allow us to
30  *    compute all of the tasks simultanously.
31  *    Also the entrance of the tachometer configuration reading, as
    well as the battery voltage measure.
32  *
33  *  V03:  02/05/2021
34  *    Fixed problem with readings from the Adafruit IMU 10dof. The
    accelerometer and
35  *    magnetometer are fusioned to deploy a better reading.
36  *    Also, I found a library for the L3G4200D gyroscope which gives
     us a more accurate and precise
37  *    measurement
38  *
39  *  V02:  16/03/2021
40  *    I2C communication with the Adafruit sensors to show data in
    the SSD1306 OLED screen.
41  *    Added buzzer system and "balanced" string sign in the oled
    when positioned  x=0, y=0, z=10.
42  *
43  *  V01:  11/02/2021
44  *    Metido la cabecera y la descripción de emails.
45  *    Versión portable del código para ver los valores de los
    sensores incluidos en el IMU,
46  *    preparada por el prof. Andrés Roldán
47  *
48  */
49
50 /*-----------------------------------------------------*/
51 /*------------------ LIBRARIES ----------------------*/
52 /*-----------------------------------------------------*/
53 #include <Wire.h>
54 /*-------------------SSD1306--------------------*/
55 #include <Adafruit_GFX.h>
56 #include <Adafruit_SSD1306.h>
57 /*--------------------WIFI--------------------*/
58 #include <ESPAsyncWebServer.h>
59 #include <AsyncTCP.h>
60 /*------------------ANDRES--------------------*/
61 #include <Adafruit_Sensor.h>
62 #include <Adafruit_LSM303_U.h>
63 #include <Adafruit_BMP085_U.h>
64 #include <L3G4200D.h>
65 #include <Adafruit_10DOF.h>
66 /*------------------MCPWM--------------------*/
```

```
#include "driver/mcpwm.h"

/*-----  CONDITIONAL COMPILATION  ------*/
//#define DEBUG TRUE    //  Comment this line if you want to print messages through the serial monitor.
//#define CABLES_GUARRETAS TRUE  // Comment this line if you're using the PCB_GUARRETA.

#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 64 // OLED display height, in pixels

#ifdef CABLES_GUARRETAS
  #define GPIO_PWM0A_OUT 12   //Declara GPIO D12 como PWM PITCH
  #define GPIO_PWM1A_OUT 26   //Declara GPIO D26 como PWM ROLL
  #define GPIO_PWM2A_OUT 25   //Declara GPIO D25 como PWM YAW
#else
  #define GPIO_PWM0A_OUT 23   //Declara GPIO D4 como PWM PITCH
  #define GPIO_PWM1A_OUT 5   //Declara GPIO D5 como PWM ROLL
  #define GPIO_PWM2A_OUT 18  //Declara GPIO D18 como PWM YAW
#endif

/*------------------CORE SELECTION FROM ESP32--------------------*/
#if CONFIG_FREERTOS_UNICORE
  static const BaseType_t app_cpu = 0;
#else
  static const BaseType_t app_cpu = 1;
#endif

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C for 128x32

/*------------------SENSORS ID----------------------*/
Adafruit_LSM303_Accel_Unified accel = Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   mag   = Adafruit_LSM303_Mag_Unified(30302);
Adafruit_BMP085_Unified       bmp   = Adafruit_BMP085_Unified(18001);
Adafruit_10DOF                dof   = Adafruit_10DOF();
L3G4200D gyro;

  // Update this with the correct SLP for accurate altitude measurements
    float seaLevelPressure = SENSORS_PRESSURE_SEALEVELHPA;

  // Read values FROM 10DOF CALCULATIONS
    sensors_event_t accel_event;      // event ask for accelerometer data
```

```
108     sensors_event_t mag_event;          // event ask for magnetometer
    data
109     sensors_event_t bmp_event;          // event ask for barometer
    data
110     sensors_event_t event;              // event ask for gyroscope
    data
111     sensors_vec_t   orientation;        // orientation vector pointer
112     GyroDPS gDPS;                       // gyro request for degrees
    per second
113   // TAKEN FROM ADAFRUIT 10DOF LIBRARY
114
115   // Variables definition:
116     //ACCELEROMETER & MAGNETOMETER
117       // Variable string preparation for sending data through WiFi
118       static String accel_x, accel_y, accel_z;
119       static String fusion_pitch, fusion_roll, fusion_heading;
120     //BAROMETER
121     static float temperature; static float altitude;
122       // Variable string preparation for sending data through WiFi
123       static String temp_str, alt_str;
124     //GYROSCOPE
125     static double xi,yi,zi;     // Variables for integration
126     static unsigned long cT;    // Current time
127     static unsigned long pT;    // Previous time
128     static unsigned long dT;    // Discrete time difference
129     double angleF_roll; double angleF_pitch;   //Fusioned angles
130       // Variable string preparation for sending data through WiFi:
131       static String gyro_x, gyro_y, gyro_z;
132       static String filter_roll, filter_pitch;
133
134 /*-----------------CNY70-TACHOMETER--------------------*/
135 #define lowLevel  2000          // Low level threshold for falling
    edge
136 #define lowLevel_R 2300
137 #define highLevel 3700          // High level threshold for rising
    edge
138
139 static bool fallen_y, fallen_p, fallen_r = false;  // Flag for
    falling/rising edge detection
140 static unsigned long lastTime_y, lastTime_p, lastTime_r = 0;  //
    Last time a rising edge was detected
141 static unsigned long thisTime_y, thisTime_p, thisTime_r = 0;  //
    Current time
142 static uint16_t thisReading_y, thisReading_p, thisReading_r = 0;  //
     Analog reading, maximum 4095
143 static uint16_t RPM_p, RPM_y, RPM_r, rads_p, rads_y, rads_r,
    rads_p_f, rads_r_f, rads_y_f = 0;
144 static int16_t ang_accel_r, ang_accel_p, ang_accel_y = 0;
145
146 //Variables needed for WiFi communication string transformation
```

```
147 static String rpm_p_str, rpm_r_str, rpm_y_str;
148 static String ang_acc_p_str, ang_acc_r_str, ang_acc_y_str;
149
150 // Specify the pins where the analog signal from CNY70 will be read
    from
151 #define PITCH_PIN 36
152 #define YAW_PIN 39
153 #define ROLL_PIN 34
154
155 // timer for cny70 rpm data reading
156 static unsigned long startMillis, currentMillis;
157 static const unsigned long period = 50;  //the value is a number of
    milliseconds
158
159 /*------------------BUZZER---------------------*/
160 #define freq_buzzer       2400  // maxima frecuencia para maximo
    volumen
161 #define channel_buzzer    0     // porque va por PWM
162 #define resolution_buzzer 8     // no necesitamos mucha resolución
    porque va al máximo
163 #define duty_c_buzzer_on  125   // definimos el duty cycle al 50%
164 #define duty_c_buzzer_off 0     // apagar el buzzer
165
166 /*-------------BATTERY MEASURE VOLTAGE DIVIDER---------------*/
167 #define BAT_PIN 32
168
169 static uint8_t bat_smpl = 20;
170 static uint16_t batt_raw = 0;
171 static uint32_t sum_bat;
172 static float bat_norm, bat_perc;
173 static String bat_perc_str;      // Variable string preparation for
    sending data through WiFi
174
175 // timer for battery voltage data reading
176 static unsigned long bat_timer, bat_start;
177 static const unsigned long bat_period = 2000;  //the value is a
    number of milliseconds
178
179 /*-------------WIFI CONFIG---------------*/
180 // Replace with your network credentials
181 #define MIWIFI TRUE
182 #ifdef MIWIFI
183 const char* ssid = "MOVISTAR_88CD";
184 const char* password = "qj889fyys3B322L5jszv";
185 #else
186 const char* ssid = "IMUDS_E";
187 const char* password = "IMUDS_E2019*";
188 #endif
189 // Create AsyncWebServer object on port 80
190 AsyncWebServer server(80);
```

```
191
192 // For the button communication
193 #define buttonPin 33
194 #define integrated_LED 2
195 const char* BUTTON_INPUT = "state";
196
197 static bool activateMotors = false;
198 static int button_read, button_state;
199 static int LED_motor, lastButtonState = LOW;
200 // In order to avoid overflow, unsigned long is the best option
201 static unsigned long lastDebounceTime = 0;
202 static unsigned long debounceDelay = 70;
203
204
205 // Definition, design and communication configuration between server
206 // and ESP32 data reading
207 const char index_html[] PROGMEM = R"rawliteral(
208 <!DOCTYPE HTML><html>
209 <head>
210   <meta name="viewport" content="width=device-width, initial-scale=1
    ">
211   <link rel="stylesheet" href="https://use.fontawesome.com/releases/
    v5.7.2/css/all.css" integrity="sha384-
    fnmOCqbTlWIlj8LyTjo7mOUStjsKC4pOpQbqyi7RrhN7udi9RwhKkMHpvLbHG9Sr"
    crossorigin="anonymous">
212   <style>
213     html {
214      font-family: Arial;
215      display: inline-block;
216      margin: 0px auto;
217      text-align: center;
218     }
219     h2 { font-size: 2.0rem; }
220     p { font-size: 2.0rem; }
221
222
223     temp { position:absolute; left:100px; top:100px;}
224     bat { position:absolute; left:800px; top:100px;}
225
226     imu_fusion_x { position:absolute; left:100px; top:175px;}
227     imu_fusion_y { position:absolute; left:550px; top:175px;}
228     imu_fusion_z { position:absolute; left:800px; top:175px;}
229
230     imu_gyro_x { position:absolute; left:100px; top:225px;}
231     imu_gyro_y { position:absolute; left:550px; top:225px;}
232     imu_gyro_z { position:absolute; left:800px; top:225px;}
233
234     filter_imu_r { position:absolute; left:100px; top:275px;}
235     filter_imu_p { position:absolute; left:600px; top:275px;}
236
```

```
237     cny_rpm_r { position:absolute; left:100px; top:325px;}
238     cny_rpm_p { position:absolute; left:550px; top:325px;}
239     cny_rpm_y { position:absolute; left:850px; top:325px;}
240
241     cny_ang_r { position:absolute; left:100px; top:375px;}
242     cny_ang_p { position:absolute; left:550px; top:375px;}
243     cny_ang_y { position:absolute; left:850px; top:375px;}
244
245     button { position:absolute; left:100px; top:425px;}
246
247     .switch {position: relative; display: inline-block; width: 120px
    ; height: 68px}
248     .switch input {display: none}
249     .slider {position: absolute; top: 0; left: 0; right: 0; bottom:
    0; background-color: #ccc; border-radius: 34px}
250     .slider:before {position: absolute; content: ""; height: 52px;
    width: 52px; left: 8px; bottom: 8px; background-color: #fff; -
    webkit-transition: .4s; transition: .4s; border-radius: 68px}
251     input:checked+.slider {background-color: #2196F3}
252     input:checked+.slider:before {-webkit-transform: translateX(52px
    ); -ms-transform: translateX(52px); transform: translateX(52px)}
253
254     .units { font-size: 1.2rem; }
255     .dht-labels{
256       font-size: 1.5rem;
257       vertical-align:middle;
258       padding-bottom: 15px;
259     }
260   </style>
261 </head>
262 <body>
263   <h2>REACTION WHEEL ESP32 MANAGEMENT INTERFACE</h2>
264
265   <p><temp>
266     <i class="fas fa-thermometer-half" style="color:#059e8a;"></i>
267     <span class="dht-labels">Temperature: </span>
268     <span id="temperature">%TEMPERATURE%</span>
269     <sup class="units">&deg;C</sup>
270   </temp></p>
271   <p>
272     <i class="fas fa-mountain" style="color:#ffoooo;"></i>
273     <span class="dht-labels">Altitude: </span>
274     <span id="altitude">%ALTITUDE%</span>
275     <span class="dht-labels">meters</span>
276   </p>
277   <p><bat>
278     <i class="fas fa-battery-full" style="color:#oooooo;"></i>
279     <span class="dht-labels">Battery (V)</span>
280     <span id="battery">%BATTERY%</span>
281     <sup class="units">&percnt;</sup>
```

```
282   </bat></p>
283
284   <p><imu_fusion_x>
285     <i class="fas fa-compass" style="color:#0000ff;"></i>
286     <span class="dht-labels">ORIENTATION X: </span>
287     <span id="fusion_x">%FUSION_X%</span>
288     <sup class="units">rad</sup>
289   </imu_fusion_x></p>
290   <p><imu_fusion_y>
291     <span class="dht-labels"> Y: </span>
292     <span id="fusion_y">%FUSION_Y%</span>
293     <sup class="units">rad</sup>
294   </imu_fusion_y></p>
295   <p><imu_fusion_z>
296     <span class="dht-labels"> Z: </span>
297     <span id="fusion_z">%FUSION_Z%</span>
298     <sup class="units">rad</sup>
299   </imu_fusion_z></p>
300
301   <p><imu_gyro_x>
302     <i class="fas fa-compass" style="color:#00ff00;"></i>
303     <span class="dht-labels">GYROSCOPE ROLL: </span>
304     <span id="gyro_x">%GYRO_X%</span>
305     <sup class="units">rad</sup>
306   </imu_gyro_x></p>
307   <p><imu_gyro_y>
308     <span class="dht-labels"> PITCH: </span>
309     <span id="gyro_y">%GYRO_Y%</span>
310     <sup class="units">rad</sup>
311   </imu_gyro_y></p>
312   <p><imu_gyro_z>
313     <span class="dht-labels"> YAW: </span>
314     <span id="gyro_z">%GYRO_Z%</span>
315     <sup class="units">rad</sup>
316   </imu_gyro_z></p>
317
318   <p><filter_imu_r>
319     <i class="fas fa-exclamation-triangle" style="color:#ffoooo;"></
    i>
320     <span class="dht-labels">ACCU. ERR. ROLL:  </span>
321     <span id="filter_roll">%FILTER_ROLL%</span>
322     <sup class="units">rad</sup>
323   </filter_imu_r></p>
324   <p><filter_imu_p>
325     <span class="dht-labels"> PITCH: </span>
326     <span id="filter_pitch">%FILTER_PITCH%</span>
327     <sup class="units">rad</sup>
328   </filter_imu_p></p>
329
330   <p><cny_rpm_r>
```

```
    <i class="fas fa-tachometer-alt" style="color:#00ff00;"></i>
    <span class="dht-labels">SPEED ROLL: </span>
    <span id="rpm_roll">%RPM_R%</span>
    <sup class="units">rpm</sup>
  </cny_rpm_r></p>
  <p><cny_rpm_p>
    <span class="dht-labels"> PITCH: </span>
    <span id="rpm_pitch">%RPM_P%</span>
    <sup class="units">rpm</sup>
  </cny_rpm_p></p>
  <p><cny_rpm_y>
    <span class="dht-labels"> YAW: </span>
    <span id="rpm_yaw">%RPM_Y%</span>
    <sup class="units">rpm</sup>
  </cny_rpm_y></p>

  <p><cny_ang_r>
    <i class="fas fa-tachometer-alt" style="color:#800080;"></i>
    <span class="dht-labels">SPEED ROLL: </span>
    <span id="ang_acc_r">%ANGACC_R%</span>
    <sup class="units">rad/s2</sup>
  </cny_ang_r></p>
  <p><cny_ang_p>
    <span class="dht-labels"> PITCH: </span>
    <span id="ang_acc_p">%ANGACC_P%</span>
    <sup class="units">rad/s2</sup>
  </cny_ang_p></p>
  <p><cny_ang_y>
    <span class="dht-labels"> YAW: </span>
    <span id="ang_acc_y">%ANGACC_Y%</span>
    <sup class="units">rad/s2</sup>
  </cny_ang_y></p>

    <p><button>
    %BUTTONPLACEHOLDER%
    </button></p>
</body>


<script>
setInterval(function ( ) {
  var xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("temperature").innerHTML = this.
 responseText;
    }
  };
  xhttp.open("GET", "/temperature", true);
  xhttp.send();
```

```
380 }, 5000 ) ;
381
382 setInterval(function ( ) {
383   var xhttp = new XMLHttpRequest();
384   xhttp.onreadystatechange = function() {
385     if (this.readyState == 4 && this.status == 200) {
386       document.getElementById("altitude").innerHTML = this.
    responseText;
387     }
388   };
389   xhttp.open("GET", "/altitude", true);
390   xhttp.send();
391 }, 8000 ) ;
392
393 setInterval(function ( ) {
394   var xhttp = new XMLHttpRequest();
395   xhttp.onreadystatechange = function() {
396     if (this.readyState == 4 && this.status == 200) {
397       document.getElementById("battery").innerHTML = this.
    responseText;
398     }
399   };
400   xhttp.open("GET", "/battery", true);
401   xhttp.send();
402 }, 2000 ) ;
403
404 setInterval(function ( ) {
405   var xhttp = new XMLHttpRequest();
406   xhttp.onreadystatechange = function() {
407     if (this.readyState == 4 && this.status == 200) {
408       document.getElementById("fusion_x").innerHTML = this.
    responseText;
409     }
410   };
411   xhttp.open("GET", "/fusion_x", true);
412   xhttp.send();
413 }, 100 ) ;
414 setInterval(function ( ) {
415   var xhttp = new XMLHttpRequest();
416   xhttp.onreadystatechange = function() {
417     if (this.readyState == 4 && this.status == 200) {
418       document.getElementById("fusion_y").innerHTML = this.
    responseText;
419     }
420   };
421   xhttp.open("GET", "/fusion_y", true);
422   xhttp.send();
423 }, 100 ) ;
424 setInterval(function ( ) {
425   var xhttp = new XMLHttpRequest();
```

```
426   xhttp.onreadystatechange = function() {
427     if (this.readyState == 4 && this.status == 200) {
428       document.getElementById("fusion_z").innerHTML = this.
 responseText;
429     }
430   };
431   xhttp.open("GET", "/fusion_z", true);
432   xhttp.send();
433 }, 100 ) ;
434
435 setInterval(function ( ) {
436   var xhttp = new XMLHttpRequest();
437   xhttp.onreadystatechange = function() {
438     if (this.readyState == 4 && this.status == 200) {
439       document.getElementById("gyro_x").innerHTML = this.
 responseText;
440     }
441   };
442   xhttp.open("GET", "/gyro_x", true);
443   xhttp.send();
444 }, 100 ) ;
445 setInterval(function ( ) {
446   var xhttp = new XMLHttpRequest();
447   xhttp.onreadystatechange = function() {
448     if (this.readyState == 4 && this.status == 200) {
449       document.getElementById("gyro_y").innerHTML = this.
 responseText;
450     }
451   };
452   xhttp.open("GET", "/gyro_y", true);
453   xhttp.send();
454 }, 100 ) ;
455 setInterval(function ( ) {
456   var xhttp = new XMLHttpRequest();
457   xhttp.onreadystatechange = function() {
458     if (this.readyState == 4 && this.status == 200) {
459       document.getElementById("gyro_z").innerHTML = this.
 responseText;
460     }
461   };
462   xhttp.open("GET", "/gyro_z", true);
463   xhttp.send();
464 }, 100 ) ;
465
466 setInterval(function ( ) {
467   var xhttp = new XMLHttpRequest();
468   xhttp.onreadystatechange = function() {
469     if (this.readyState == 4 && this.status == 200) {
470       document.getElementById("filter_roll").innerHTML = this.
 responseText;
```

```
471     }
472   };
473   xhttp.open("GET", "/filter_roll", true);
474   xhttp.send();
475 }, 100 ) ;
476 setInterval(function ( ) {
477   var xhttp = new XMLHttpRequest();
478   xhttp.onreadystatechange = function() {
479     if (this.readyState == 4 && this.status == 200) {
480       document.getElementById("filter_pitch").innerHTML = this.
 responseText;
481     }
482   };
483   xhttp.open("GET", "/filter_pitch", true);
484   xhttp.send();
485 }, 100 ) ;
486
487 setInterval(function ( ) {
488   var xhttp = new XMLHttpRequest();
489   xhttp.onreadystatechange = function() {
490     if (this.readyState == 4 && this.status == 200) {
491       document.getElementById("rpm_roll").innerHTML = this.
 responseText;
492     }
493   };
494   xhttp.open("GET", "/rpm_roll", true);
495   xhttp.send();
496 }, 100 ) ;
497 setInterval(function ( ) {
498   var xhttp = new XMLHttpRequest();
499   xhttp.onreadystatechange = function() {
500     if (this.readyState == 4 && this.status == 200) {
501       document.getElementById("rpm_pitch").innerHTML = this.
 responseText;
502     }
503   };
504   xhttp.open("GET", "/rpm_pitch", true);
505   xhttp.send();
506 }, 100 ) ;
507 setInterval(function ( ) {
508   var xhttp = new XMLHttpRequest();
509   xhttp.onreadystatechange = function() {
510     if (this.readyState == 4 && this.status == 200) {
511       document.getElementById("rpm_yaw").innerHTML = this.
 responseText;
512     }
513   };
514   xhttp.open("GET", "/rpm_yaw", true);
515   xhttp.send();
516 }, 100 ) ;
```

```
517
518 setInterval(function ( ) {
519   var xhttp = new XMLHttpRequest();
520   xhttp.onreadystatechange = function() {
521     if (this.readyState == 4 && this.status == 200) {
522       document.getElementById("ang_acc_r").innerHTML = this.
 responseText;
523     }
524   };
525   xhttp.open("GET", "/ang_acc_r", true);
526   xhttp.send();
527 }, 100 ) ;
528 setInterval(function ( ) {
529   var xhttp = new XMLHttpRequest();
530   xhttp.onreadystatechange = function() {
531     if (this.readyState == 4 && this.status == 200) {
532       document.getElementById("ang_acc_p").innerHTML = this.
 responseText;
533     }
534   };
535   xhttp.open("GET", "/ang_acc_p", true);
536   xhttp.send();
537 }, 100 ) ;
538 setInterval(function ( ) {
539   var xhttp = new XMLHttpRequest();
540   xhttp.onreadystatechange = function() {
541     if (this.readyState == 4 && this.status == 200) {
542       document.getElementById("ang_acc_y").innerHTML = this.
 responseText;
543     }
544   };
545   xhttp.open("GET", "/ang_acc_y", true);
546   xhttp.send();
547 }, 100 ) ;
548
549 function toggleCheckbox(element) {
550   var xhr = new XMLHttpRequest();
551   if(element.checked){ xhr.open("GET", "/update?state=1", true); }
552   else { xhr.open("GET", "/update?state=0", true); }
553   xhr.send();
554 }
555 setInterval(function ( ) {
556   var xhttp = new XMLHttpRequest();
557   xhttp.onreadystatechange = function() {
558     if (this.readyState == 4 && this.status == 200) {
559       var inputChecked;
560       var outputStateM;
561       if( this.responseText == 1){
562         inputChecked = true;
563         outputStateM = "On";
```

```
564         }
565         else {
566           inputChecked = false;
567           outputStateM = "Off";
568         }
569         document.getElementById("integrated_LED").checked =
 inputChecked;
570         document.getElementById("outputState").innerHTML =
 outputStateM;
571       }
572     };
573     xhttp.open("GET", "/state", true);
574     xhttp.send();
575   }, 1000 ) ;
576
577 </script>
578 </html>)rawliteral";
579
580 /*********************************************************************/
581 /*!
582     @brief  Initialises all the sensors and the OLED used by the
 system.
583 */
584 /*********************************************************************/
585
586 static void initCompo()
587 {
588   /* Initialise the display */
589   if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
590     Serial.println(F("SSD1306 allocation failed"));
591     for(;;); // Don't proceed, loop forever
592   }
593
594   /* Initialise the sensors BMP085 and LSM303*/
595   if(!bmp.begin())
596   {
597     /* There was a problem detecting the BMP085 ... check your
 connections */
598     Serial.print("Ooops, no BMP085 detected ... Check your wiring or
  I2C ADDR!");
599     while(1);
600   }
601
602   if(!accel.begin())
603   {
604     /* There was a problem detecting the LSM303 ... check your
 connections */
```

```
605     Serial.println(F("Ooops, no LSM303 detected ... Check your
    wiring!"));
606     while(1);
607   }
608
609   if(!mag.begin())
610   {
611     /* There was a problem detecting the LSM303 ... check your
    connections */
612     Serial.println("Ooops, no LSM303 detected ... Check your wiring!
    ");
613     while(1);
614   }
615 }
616
617 /**************************************************************************/
618 /*!
619     @brief  MCPWM setup configuration for the motor velocity
    management.
620 */
621 /**************************************************************************/
622
623 static void MCPWM_setup(){
624   //mcpwm_gpio_init(Unit PWM 0-1, saida (Timer 0-2, Pair A-B), GPIO)
625   mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM0A, GPIO_PWM0A_OUT);
626   //------------- Adding for two motors
627   mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM1A, GPIO_PWM1A_OUT);
628   mcpwm_gpio_init(MCPWM_UNIT_0, MCPWM2A, GPIO_PWM2A_OUT);
629   // configuration of the pwm signal characteristics
630   mcpwm_config_t pwm_config;
631   pwm_config.frequency = 50;                              //frequency =
    50Hz,
632   pwm_config.cmpr_a = 0.0;                                  // (duty
    cycle) do PWMxA = 0
633   pwm_config.cmpr_b = 0.0;                                  // (duty
    cycle) do PWMxb = 0
634   pwm_config.counter_mode = MCPWM_UP_COUNTER;             //Para MCPWM
    assimetric
635   pwm_config.duty_mode = MCPWM_DUTY_MODE_0;               //duty cycle
    high
636   //Inicia(Unit , Timer, Config PWM)
637   mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_0, &pwm_config); //Define
    PWM0A & PWM0B with above configurations
638   mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_1, &pwm_config); //Define
    PWM0A & PWM0B with above configurations
639   mcpwm_init(MCPWM_UNIT_0, MCPWM_TIMER_2, &pwm_config); //Define
    PWM0A & PWM0B with above configurations
640 }
```

```
641
642 /**************************************************************************/
643 /*!
644     @brief  Function which configures the operator's A MCPWM (Unit, Timer, Percentage)
645             Duty Cycle.
646 */
647 /**************************************************************************/
648
649 static void brushed_motor_forward(mcpwm_unit_t mcpwm_num, mcpwm_timer_t timer_num , float duty_cycle)
650 {
651   mcpwm_set_duty(mcpwm_num, timer_num, MCPWM_OPR_A, duty_cycle);
652 }
653
654 /**************************************************************************/
655 /*!
656     @brief  Battery measurement system designed with a voltage divider.
657             A series resistance with a low pass filter (R, R//C) allow usto determine
658             the remain voltage of the battery, thereby knowing how much
659             battery we have left before causing a deep discharge of the battery.
660 */
661 /**************************************************************************/
662
663 void BatteryMeasure(void* parameters)
664 {
665   while(1)
666   {
667     batt_raw = analogRead(BAT_PIN);  //max value 4095
668     sum_bat = 0;
669     for (uint8_t i = 0; i < bat_smpl; ++i)
670     {
671       sum_bat += batt_raw; //max sum value 4095*100 = 409500
672     }
673     bat_norm = 3.818181 * (3.3/4095) * (sum_bat/bat_smpl);   // real voltage battery value.
674     bat_perc = ((12.6-bat_norm)/3) * 100; bat_perc_str = String(bat_perc);
675     bat_timer = millis();  //get the current "time"
676     if (bat_timer - bat_start >= bat_period)
677     {
678       #ifdef DEBUG
```

```
679       Serial.print(F("Battery Voltage: ")); Serial.print(bat_norm);
 Serial.print(F(" % ; "));
680       Serial.print(F(" Battery Voltage Percentage: ")); Serial.print
 (bat_perc, 2); Serial.println(F(" % "));
681       #endif
682       bat_start = bat_timer;  // Save the start time of the current
 state.
683     }
684   }
685 }
686
687 /*************************************************************************/
688 /*!
689     @brief  Function which reads the value from the button to see if
  we can
690             activate the motors.
691             This value will be communicated to our WiFi interface.
692 */
693 /*************************************************************************/
694
695 void ButtonMotor(void* parameters)
696 {
697   while(1)
698   {
699     // Check to see the value of the button
700     button_read = digitalRead(buttonPin);
701     // Depending on the change, it can be LOW to HIGH o HIGH to LOW,
  that's why we use change:
702     if (button_read != lastButtonState) {
703         lastDebounceTime = millis();   // update the value of
 debounce from button
704     }
705     // Once the debounce delay has passed (consider 50 ms)
706     if ((millis() - lastDebounceTime) > debounceDelay) {
707       if (button_read != button_state) {    // if the value read is
 different from the
708         button_state = button_read;         // former state value of
  the button, update it.
709         if (button_state == HIGH) {         // If it's HIGH, change
 the state of the variable
710             activateMotors == true;   // For the motors, TRUE or
 FALSE
711             LED_motor = !LED_motor;             // For the LED
 indicator, HIGH or LOW
712         }
713       }
714     }
715   // set the LED:
```

```
716   digitalWrite(integrated_LED, LED_motor);
717   lastButtonState = button_read;  // in order to compare if it has
    changed in the next iteration
718   vTaskDelay(100/ portTICK_PERIOD_MS);
719   }
720 }
721
722 /*************************************************************************/

723 /*!
724     @brief  Function which manages the PWM sending to the motors
725             Duty Cycle.
726 */
727 /*************************************************************************/

728
729 void MotorMovement(void* parameters)
730 {
731   while(1)
732   {
733     // vTaskDelay(2000 / portTICK_PERIOD_MS);
734     if(activateMotors == true){
735       // Acceleration from initial velocity to medium velocity
736       // Min == 5.346%     Max == 9.516%     Values from RC Controller
737       for(float i=5.0 ; i<=7.0 ; i+= 0.1){
738         brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, i);
739         brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_1, i);
740         brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_2, i);
741         //Serial.print(F("ACCEL: ")); Serial.print(i); Serial.
    println(F(""));
742         Serial.println(F("SPEEDING"));
743         vTaskDelay(300 / portTICK_PERIOD_MS);
744       }
745       // Delay between accel and decel not to overimpose one onto
    the other
746       vTaskDelay(300 / portTICK_PERIOD_MS);
747       // Deceleration from maximum velocity to initial velocity
748       for(float j=7.0 ; j>=5.0 ; j-= 0.1){
749         brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_0, j);
750         brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_1, j);
751         brushed_motor_forward(MCPWM_UNIT_0, MCPWM_TIMER_2, j);
752         //Serial.print(F(" DECEL: ")); Serial.print(j); Serial.
    println(F(""));
753         Serial.println(F("BRAKING"));
754         vTaskDelay(300 / portTICK_PERIOD_MS);
755       }
756       // Change the state variable so it does not repeat until told
    by the button
757       activateMotors == false;
758     }
```

```
759   }
760 }
761
762 /*************************************************************************/
763 /*!
764     @brief  Tachometer function. With the CNY70 and this program below we are
765             able to read the analog signal received as an RPM signal.
766
767             Since there is only one rising edge per rotation on this
768             tachometer, and only the rising edges are being used in the RPM
769             calculation, the elapsed time since the last reading is equal to the
770             revolutions per microsecond.
771 */
772 /*************************************************************************/
773
774 static void Tachometer_P(void* parameter)
775 {
776   while(1)
777   {
778     thisReading_p = analogRead(PITCH_PIN);    // Take a reading check to see
779     if (thisReading_p > highLevel)            // if above  the high level
780     {                                         // threshold, if so, it was low on
781       if (fallen_p == true)                   // previous pass which means a
782       {                                       // rising edge was detected
783         thisTime_p = micros();
784         // 1 rev/ usec * 1e6 usec/1 sec *60 sec/1 min == [revs/min] == RPM
785         RPM_p = 60000000 / (thisTime_p - lastTime_p);       // RPM = Revolutions per minute
786         // rad/s because we need angular acceleration for our torque design response
787         // 1 rev/ min * 1 min/60 sec * 2pi rad/1 rev == [rad/sec] == w
788         rads_p = (RPM_p*2*PI) / 60;
789         // Angular acceleration is dw/dt which can be discretised into w2-w1/t2-t1:
790           ang_accel_p = (rads_p - rads_p_f) / (thisTime_p - lastTime_p);
```

```
791           rpm_p_str = String(RPM_p); ang_acc_p_str = String(
    ang_accel_p);
792           rads_p_f = rads_p;
793         lastTime_p = thisTime_p;              // Set the time to "
    last time" and
794         fallen_p = false;                     // the edge to false
    so we won't count this rise again
795       }
796     }
797     if (thisReading_p < lowLevel)             // When the edge
    first falls below the
798     {                                         // low level
    threshold, make fallen true
799       if (fallen_p == false)
800       {
801         fallen_p = true;
802       }
803     }
804   }
805 }
806
807 static void Tachometer_Y(void* parameter)
808 {
809   while(1)
810   {
811     thisReading_y = analogRead(YAW_PIN);
812     if (thisReading_y > highLevel)
813     {
814       if (fallen_y == true)
815       {
816         thisTime_y = micros();
817         RPM_y = 60000000 / (thisTime_y - lastTime_y); rads_y = (
    RPM_y*2*PI) / 60;
818           ang_accel_y = (rads_y - rads_y_f) / (thisTime_y -
    lastTime_y);
819           rpm_y_str = String(RPM_y); ang_acc_y_str = String(
    ang_accel_y);
820           rads_y_f = rads_y;
821         lastTime_y = thisTime_y;
822         fallen_y = false;
823       }
824     }
825     if (thisReading_y < lowLevel)
826     {
827       if (fallen_y == false)
828       {
829         fallen_y = true;
830       }
831     }
832   }
```

```
833 }
834
835 static void Tachometer_R(void* parameter)
836 {
837   while(1)
838   {
839     thisReading_r = analogRead(ROLL_PIN);
840     if (thisReading_r > highLevel )
841     {
842       if (fallen_r == true)
843       {
844         thisTime_r = micros();
845         RPM_r = 60000000 / (thisTime_r - lastTime_r); rads_r = (
 RPM_r*2*PI) / 60;
846           ang_accel_r = (rads_r - rads_r_f) / (thisTime_r -
 lastTime_r);
847           rpm_r_str = String(RPM_r); ang_acc_r_str = String(
 ang_accel_r);
848           rads_r_f = rads_r;
849         lastTime_r = thisTime_r;
850         fallen_r = false;
851       }
852     }
853     if (thisReading_r < lowLevel)
854     {
855       if (fallen_r == false)
856       {
857         fallen_r = true;
858       }
859     }
860   }
861 }
862
863 /* static void RPM_data(void* parameters)
864 {
865   while(1)
866   {
867     currentMillis = millis();  //get the current "time"
868     if (currentMillis - startMillis >= period)
869     {
870       #ifdef DEBUG
871       Serial.print(F("RPM Pitch: ")); Serial.print(RPM_p); Serial.
 print(F("  ,"));
872       Serial.print(F("rad/s: ")); Serial.print(ang_accel_p,2);
 Serial.print(F("  ;"));
873       Serial.print(F("RPM Yaw: ")); Serial.print(RPM_y); Serial.
 print(F("  ,"));
874       Serial.print(F("rad/s: ")); Serial.print(ang_accel_y,2);
 Serial.print(F("  ;"));
```

```
875       Serial.print(F("RPM Roll: ")); Serial.print(RPM_r); Serial.
 print(F("  ,"));
876       Serial.print(F("rad/s: ")); Serial.print(ang_accel_r); Serial.
 print(F("  ;"));
877       Serial.println(F(" "));
878       #endif
879       startMillis = currentMillis;  // Save the start time of the
 current state.
880     }
881   }
882 } */
883 
884 /*************************************************************************/
885 /*!
886     @brief  Function which reads the gravity, pitch, roll, heading
 and altitude.
887             It shows the data on the display (due to size
 restrictions only
888             gravity on three axis and pitch, roll and heading) and
 in the
889             Serial monitor.
890             Baud rate is 115200.
891 */
892 /*************************************************************************/
893 
894 static void SensorResults(void * parameters)
895 {
896   while(1)
897   {
898     display.clearDisplay();        // Clear former data shown in
 OLED
899     accel.getEvent(&accel_event);  // Get event from accelerometer
900     mag.getEvent(&mag_event);      // Get event from magnetometer
901     accel_x = String(accel_event.acceleration.x);
902     accel_y  = String(accel_event.acceleration.y);
903     accel_z = String(accel_event.acceleration.z);
904     /*-----------------ACCELEROMETER--------------------*/
905     // Display the accelerometer results (acceleration is measured
 in m/s^2)
906       display.setFont(); display.setTextSize(2); display.setCursor
 (0,0);
907       display.print("ACCEL");
908       display.setTextSize(1); display.setCursor(0, 16); display.
 print("X: ");
909       display.print(accel_x);
910       display.setTextSize(1); display.setCursor(0,26); display.print
 ("Y: ");
911       display.print(accel_y);
```

```
912       display.setTextSize(1); display.setCursor(0,36); display.print
("Z: ");
913       display.print(accel_z);
914       display.setTextSize(1); display.setCursor(0,46); display.print
("m/s2");
915     /*Serial print of the prior sensor, to see clearly the behaviour
*/
916   #ifdef DEBUG
917     Serial.print(F("X grav: "));
918     Serial.print(accel_event.acceleration.x); Serial.print(F("; "));
919     Serial.print(F("Y grav: "));
920     Serial.print(accel_event.acceleration.y); Serial.print(F("; "));
921     Serial.print(F("Z grav: "));
922     Serial.print(accel_event.acceleration.z); Serial.print(F(" m/s2;
 "));
923   #endif
924
925     /*------------------GYROSCOPE----------------------*/
926     // Measurements from gyroscope in DPS (degrees per second)
927     gDPS = gyro.readGyroDPS();
928     gyro_x = String(gDPS.x); gyro_y = String(gDPS.y); gyro_z =
String(gDPS.z);
929     /*
930       Measurements from gyroscope integrating dps -> distance
931       This distance is the sum (or substract, depending on the
932       twist direction) of the angles tilted
933       from the original position.
934       dT represents the discrete time differentiation.
935     */
936     cT = micros(); dT = cT - pT; pT = cT;
937     xi = xi + gDPS.x*(dT/1000000.0);
938     yi = yi + gDPS.y*(dT/1000000.0);
939     zi = zi + gDPS.z*(dT/1000000.0);
940   #ifdef DEBUG
941     // Print gyroscope degrees per second values
942     Serial.print(F("X dps: ")); Serial.print(gyro_x); Serial.print(F
("  ; "));
943     Serial.print(F("Y dps: ")); Serial.print(gyro_y); Serial.print(F
("  ; "));
944     Serial.print(F("Z dps: ")); Serial.print(gyro_z); Serial.print(F
("  deg/sec; "));
945     // Print gyroscope integral of previous values, to see if there'
s a lot of noise or not
946     Serial.print(F("Xi: ")); Serial.print(xi); Serial.print(F("; "))
;
947     Serial.print(F("Yi: ")); Serial.print(yi); Serial.print(F("; "))
;
948     Serial.print(F("Zi: ")); Serial.print(zi); Serial.print(F(" sum;
 "));
949   #endif
```

```
950
951     /*------------------ORIENTATION---------------------*/
952     if (dof.fusionGetOrientation(&accel_event, &mag_event, &
    orientation))
953       {
954       fusion_pitch = String(orientation.pitch);
955       fusion_roll  = String(orientation.roll);
956       fusion_heading = String(orientation.heading);
957       // Display the orientation, which is the pitch, roll and yaw
    in degrees
958       display.setFont(); display.setTextSize(2); display.setCursor
    (70,0);
959       display.print("ORTN");
960       display.setTextSize(1); display.setCursor(70, 16); display.
    print("P: ");
961       display.print(fusion_pitch);
962       display.setTextSize(1); display.setCursor(70, 26); display.
    print("R: ");
963       display.print(fusion_roll);
964       display.setTextSize(1); display.setCursor(70, 36); display.
    print("Y: ");
965       display.print(fusion_heading);
966       display.setTextSize(1); display.setCursor(70, 46);display.
    print("deg");
967       /*
968         This next expression computes the difference between the
    accelerometer
969         and the gyroscope, showing how much error has been
    accumulating
970         since the beginning of the data collection.
971       */
972       angleF_roll = 0.95*(angleF_roll + gDPS.x*(dT/1000000.0)) +
    0.05*orientation.roll;
973       angleF_pitch = 0.95*(angleF_pitch + gDPS.y*(dT/1000000.0)) +
    0.05*orientation.pitch;
974       filter_roll = String(angleF_roll); filter_pitch = String(
    angleF_pitch);
975
976     #ifdef DEBUG
977       // Serial display of data
978       Serial.print(F("Roll: ")); Serial.print(orientation.roll);
    Serial.print(F("; "));
979       Serial.print(F("Pitch: ")); Serial.print(orientation.pitch);
    Serial.print(F("; "));
980       Serial.print(F("Heading: ")); Serial.print(orientation.heading
    );
981       Serial.print(F(" deg; "));
982       Serial.print(F("Roll fus: ")); Serial.print(angleF_roll);
    Serial.print(F("; "));
```

```
983       Serial.print(F("Pitch fus: ")); Serial.print(angleF_pitch);
Serial.print(F("; "));
984     #endif
985       }
986
987     /*----------------ALTITUDE & TEMPERATURE
----------------------*/
988     /* Calculate the altitude using the barometric pressure sensor
*/
989     bmp.getEvent(&bmp_event);
990     if (bmp_event.pressure)
991     {
992       /* Get ambient temperature in C */
993       bmp.getTemperature(&temperature);
994       temp_str = String(temperature);
995       /* Convert atmospheric pressure, SLP and temp to altitude
*/
996       altitude = bmp.pressureToAltitude(seaLevelPressure, bmp_event.
pressure, temperature);
997       alt_str = String(altitude);
998       display.setTextSize(1); display.setCursor(70,56); display.
print("ALT:");
999       display.print(altitude);
1000
1001      #ifdef DEBUG
1002      Serial.print(F("Alt: ")); Serial.print(altitude); Serial.print
(F(" m; "));
1003      Serial.print(F("Temp: ")); Serial.print(temperature); Serial.
print(F(" m; "));
1004      #endif
1005    }
1006
1007    // Acoustic sign of having reached the balanced position with
BUZZER
1008    /* if ((accel_event.acceleration.z > 9.2) && ( abs(accel_event.
acceleration.x) < 0.3) && (abs(accel_event.acceleration.y) < 0.3))
1009    {
1010      display.setTextSize(1); display.setCursor(0,56); display.print
("BALANCED");
1011      ledcWriteTone(channel_buzzer, freq_buzzer);
1012      ledcWrite(channel_buzzer, duty_c_buzzer_on);
1013    } else{
1014            display.setTextSize(1); display.setCursor(0,56); display
.print("UNBALANCED");
1015            ledcWrite(channel_buzzer, duty_c_buzzer_off);
1016          } */
1017
1018    display.display();
1019    vTaskDelay(300 / portTICK_PERIOD_MS);
1020  }
```

```
1021 }
1022
1023 String outputState(){
1024   if(digitalRead(integrated_LED)){
1025     return "checked";
1026   }
1027   else {
1028     return "";
1029   }
1030   return "";
1031 }
1032
1033 /*********************************************************************/

1034 /*!
1035     @brief  Processor function creates the gateway between our main
     program
1036             data and the HTML variable where it will be saved and
     afterwards
1037             sent to our server.
1038
1039 */
1040 /*********************************************************************/

1041 String processor(const String& var){
1042   //Serial.println(var);
1043   if(var == "BUTTONPLACEHOLDER"){
1044     String buttons ="";
1045     String outputStateValue = outputState();
1046     buttons+= "<h4>Output - GPIO 2 - State <span id=\"outputState
     \"></span></h4><label class=\"switch\"><input type=\"checkbox\"
     onchange=\"toggleCheckbox(this)\" id=\"output\" " +
     outputStateValue + "><span class=\"slider\"></span></label>";
1047     return buttons;
1048   }
1049   else if(var == "TEMPERATURE"){ return temp_str;}
1050   else if(var == "ALTITUDE"){ return alt_str;}
1051   else if(var == "BATTERY"){ return bat_perc_str;}
1052
1053   else if(var == "FUSION_X"){ return fusion_roll;}
1054   else if(var == "FUSION_Y"){ return fusion_pitch ;}
1055   else if(var == "FUSION_Z"){ return fusion_heading;}
1056   else if(var == "GYRO_X"){ return gyro_x;}
1057   else if(var == "GYRO_Y"){ return gyro_y; }
1058   else if(var == "GYRO_Z"){ return gyro_z;}
1059   else if(var == "FILTER_ROLL"){ return filter_roll;}
1060   else if(var == "FILTER_PITCH"){ return filter_pitch;}
1061
1062   else if(var == "RPM_R"){ return rpm_r_str;}
1063   else if(var == "RPM_P"){ return rpm_p_str;}
```

```
1064    else if(var == "RPM_Y"){ return rpm_y_str;}
1065
1066    else if(var == "ANGACC_R"){ return ang_acc_r_str;}
1067    else if(var == "ANGACC_P"){ return ang_acc_p_str;}
1068    else if(var == "ANGACC_Y"){ return ang_acc_y_str;}
1069
1070    return String();
1071  }
1072
1073  void WiFi_HTML_setup()
1074  {
1075    // Connect to Wi-Fi
1076    WiFi.begin(ssid, password);
1077    while (WiFi.status() != WL_CONNECTED) {
1078      delay(1000);
1079      Serial.println("Connecting to WiFi..");
1080    }
1081
1082    // Print ESP32 Local IP Address
1083    Serial.println(WiFi.localIP());
1084
1085    // Route for root / web page
1086    server.on("/", HTTP_GET, [](AsyncWebServerRequest *request){
1087      request->send_P(200, "text/html", index_html, processor);
1088    });
1089    // Request to get temperature and altitude from the barometer
1090    server.on("/temperature", HTTP_GET, [](AsyncWebServerRequest *
   request){
1091      request->send_P(200, "text/plain", temp_str.c_str());
1092    });
1093    server.on("/altitude", HTTP_GET, [](AsyncWebServerRequest *request
   ){
1094      request->send_P(200, "text/plain", alt_str.c_str());
1095    });
1096    // Request to get the battery voltage level in percentage
1097    server.on("/battery", HTTP_GET, [](AsyncWebServerRequest *request)
   {
1098      request->send_P(200, "text/plain", bat_perc_str.c_str());
1099    });
1100    // Request to get the fusion orientation from the IMU
1101    server.on("/fusion_x", HTTP_GET, [](AsyncWebServerRequest *request
   ){
1102      request->send_P(200, "text/plain", fusion_roll.c_str());
1103    });
1104    server.on("/fusion_y", HTTP_GET, [](AsyncWebServerRequest *request
   ){
1105      request->send_P(200, "text/plain", fusion_pitch.c_str());
1106    });
1107    server.on("/fusion_z", HTTP_GET, [](AsyncWebServerRequest *request
   ){
```

```
1108      request->send_P(200, "text/plain", fusion_heading.c_str());
1109    });
1110    // Request to get the gyro information
1111    server.on("/gyro_x", HTTP_GET, [](AsyncWebServerRequest *request){
1112      request->send_P(200, "text/plain", gyro_x.c_str());
1113    });
1114    server.on("/gyro_y", HTTP_GET, [](AsyncWebServerRequest *request){
1115      request->send_P(200, "text/plain", gyro_y.c_str());
1116    });
1117    server.on("/gyro_z", HTTP_GET, [](AsyncWebServerRequest *request){
1118      request->send_P(200, "text/plain", gyro_z.c_str());
1119    });
1120    // Request to get the filtered signal obtained with the
     accelerometer and the gyroscope
1121    server.on("/filter_roll", HTTP_GET, [](AsyncWebServerRequest *
     request){
1122      request->send_P(200, "text/plain", filter_roll.c_str());
1123    });
1124    server.on("/filter_pitch", HTTP_GET, [](AsyncWebServerRequest *
     request){
1125      request->send_P(200, "text/plain", filter_pitch.c_str());
1126    });
1127    // Request to get the RPM value from the inertial disks
1128    server.on("/rpm_roll", HTTP_GET, [](AsyncWebServerRequest *request
     ){
1129      request->send_P(200, "text/plain", rpm_r_str.c_str());
1130    });
1131    server.on("/rpm_pitch", HTTP_GET, [](AsyncWebServerRequest *
     request){
1132      request->send_P(200, "text/plain", rpm_p_str.c_str());
1133    });
1134    server.on("/rpm_yaw", HTTP_GET, [](AsyncWebServerRequest *request)
     {
1135      request->send_P(200, "text/plain", rpm_y_str.c_str());
1136    });
1137    // Request to get the angular acceleration from the inertial disks
1138    server.on("/ang_acc_r", HTTP_GET, [](AsyncWebServerRequest *
     request){
1139      request->send_P(200, "text/plain", ang_acc_r_str.c_str());
1140    });
1141    server.on("/ang_acc_p", HTTP_GET, [](AsyncWebServerRequest *
     request){
1142      request->send_P(200, "text/plain", ang_acc_p_str.c_str());
1143    });
1144    server.on("/ang_acc_y", HTTP_GET, [](AsyncWebServerRequest *
     request){
1145      request->send_P(200, "text/plain", ang_acc_y_str.c_str());
1146    });
1147
```

```
1148   /*-----FOR THE BUTTON (PHYSYCAL & HTML INTERFACE) COMMUNICATION
       ------*/
1149   // Send a GET request to <ESP_IP>/update?state=<inputMessage>
1150   server.on("/update", HTTP_GET, [] (AsyncWebServerRequest *request)
       {
1151     String inputMessage;
1152     String inputParam;
1153     // GET input1 value on <ESP_IP>/update?state=<inputMessage>
1154     if (request->hasParam(BUTTON_INPUT)) {
1155       inputMessage = request->getParam(BUTTON_INPUT)->value();
1156       inputParam = BUTTON_INPUT;
1157       digitalWrite(integrated_LED, inputMessage.toInt());
1158       LED_motor = !LED_motor;
1159     }
1160     else {
1161       inputMessage = "No message sent";
1162       inputParam = "none";
1163     }
1164     Serial.println(inputMessage);
1165     request->send(200, "text/plain", "OK");
1166   });
1167
1168   // Send a GET request to <ESP_IP>/state
1169   server.on("/state", HTTP_GET, [] (AsyncWebServerRequest *request)
       {
1170     request->send(200, "text/plain", String(digitalRead(
       integrated_LED)).c_str());
1171   });
1172
1173   // Start server
1174   server.begin();
1175 }
1176
1177 /*
       ********************************************************************
       */
1178 /*setup*/
1179 /*
       ********************************************************************
       */
1180 void setup(void)
1181 {
1182   Serial.begin(115200); Wire.begin();
1183   // LED indication for motors' startup
1184   pinMode(integrated_LED, OUTPUT); digitalWrite(integrated_LED, LOW)
       ;
1185   // Button configuration setup
1186   pinMode(buttonPin, INPUT);
1187   // Buzzer config:
1188   ledcSetup(channel_buzzer, freq_buzzer, resolution_buzzer);
```

```
1189   ledcAttachPin(13, channel_buzzer);
1190   // WiFi setup and initialization
1191   WiFi_HTML_setup();
1192   // Tachometer config:
1193   pinMode(PITCH_PIN, INPUT_PULLDOWN);
1194   pinMode(YAW_PIN, INPUT_PULLDOWN);
1195   pinMode(ROLL_PIN, INPUT_PULLDOWN);
1196   // MCPWM setup configuration function:
1197     MCPWM_setup();
1198
1199   // Initialise the sensors and the display function:
1200     initCompo();
1201   // Gyro setup code:
1202     gyro.init(-120.1, -563.64724, -0.48845);
1203     pT = 0;
1204     xi = yi = zi = 0;
1205     delay(1500);   //Time to wait before calibration to open the
      serial
1206     gyro.printCalibrationValues(150);
1207
1208   // Initialize the OLED display:
1209   display.clearDisplay();
1210   display.setTextColor(WHITE);
1211
1212   /* // Start serial read task
1213   xTaskCreatePinnedToCore(  // Use xTaskCreate() in vanilla FreeRTOS
1214             readSerial,     // Function to be called
1215             "Read Serial",  // Name of task
1216             1024,           // Stack size (bytes in ESP32, words in
      FreeRTOS)
1217            NULL,           // Parameter to pass
1218             1,              // Task priority (must be same to
      prevent lockup)
1219             NULL,           // Task handle
1220             app_cpu);       // Run on one core for demo purposes (
      ESP32 only) */
1221
1222   // Start Motor PWM signal creation task:
1223   xTaskCreatePinnedToCore( MotorMovement,"Motor PWM signal",102400,
      NULL,1, NULL,app_cpu);
1224   // Start IMU data reading task:
1225   xTaskCreatePinnedToCore( SensorResults,"IMU data reading",102400,
      NULL,1, NULL,app_cpu);
1226   // Start Tachometer for Pitch function
1227   xTaskCreatePinnedToCore( Tachometer_P,"Tachometer Pitch", 10240,
      NULL,1, NULL,app_cpu);
1228   // Start Tachometer for Yaw function
1229   xTaskCreatePinnedToCore( Tachometer_Y, "Tachometer Yaw",  10240,
      NULL,1, NULL,app_cpu);
1230   // Start Tachometer for Roll function
```

```
1231   xTaskCreatePinnedToCore( Tachometer_R, "Tachometer Roll", 10240,
       NULL,1, NULL,app_cpu);
1232   // Timer for serial data RPM management function
1233   // xTaskCreatePinnedToCore( RPM_data,"RPM serial data",      1024,
        NULL,1, NULL,app_cpu);
1234   // Read Battery Voltage:
1235   xTaskCreatePinnedToCore( BatteryMeasure,"Voltage battery measure"
       ,10240, NULL,1, NULL,app_cpu);
1236   // LED and Motor control WIfi:
1237   xTaskCreatePinnedToCore( ButtonMotor,"LED and Motor Controll
       through WIFI",10240, NULL,1, NULL,app_cpu);
1238
1239   // Delete "setup and loop" task
1240   vTaskDelete(NULL);
1241 }
1242
1243 void loop()
1244 {
1245   // nothing here
1246 }
1247
```

Code D.1 – *Main Code of the Simulation System*

Appendix E

Project Budget

E.1 Materials and hardware

In this section, project cost regarding materials and hardware sections will be detailed. Each one of the different hardware subsections is differentiated. Human resources area also included.

For the **SMD components, screws and nuts** we will consider the average price of each kind of component, as the budget would become unnecessarily long.

E.1.1 Air Bearing Platform

The Air Bearing Platform manufacturing was produced by Ángel Paredes. Here is an approximation of the budget needed.

Item	Units	Cost/u. (€)	Cost (€)
Aluminum Round Cylinder	1	85.11	85.11
CNC Milling Machine Renting	1	35.00 (€/h)	175.00
Electricity Bill	8 kWh	0.22 (€ kW/h Taxes inc.)	8.8
Material Subtotal (before 21 % VAT):			**260.11 €**
Total (VAT included):			**323.53 €**

Table E.1 – *Air Bearing Manufacturing*

E.1.2 Simulation Platform

E.1.2.1 Mechanical Platform

The mechanical structure has been studied in Table E.1.

E.1.2.2 CU

The cost of the Control Unit is outspread into the electronic components, machine renting and materials, as follows.

E.1.2.2.1 Bill of Materials

Item	Units	Cost/u. (€)	Cost (€)
Buzzer KXG1205	1	0.55	0.55
SSD1306 Display	1	4.99	4.99
Switch Button	2	0.25	0.50
LiPo 2200 mAh Battery	2	14.59	29.18
Female 15 Single Socket	1	1.06	1.06
Male 15 Single Socket	1	1.08	1.08
Battery SMD Bornes	3	1.25	3.75
Adafruit 10-DoF IMU	1	17.00	17.00
Switch	1	2.56	2.56
Resistor SMD	8	0.67	5.36
Capacitor SMD	10	0.89	8.9
Inductance	1	1.56	1.56
Regulator BD9E104FJ	3	0.89	2.67
INA219	2	2.51	5.02
Wires (m)	2.5m	0.85 €/m	2.13
Node MCU ESP32	2	6.75	13.5
CNY70	4	1.20	4.80
Subtotal (before 21 % VAT):			**104.61 €**
Total (VAT included):			**126.58 €**

Table E.2 – *Control Unit Components*

E.1.2.2.2 Manufacturing

The manufacturing of four 2 layer PCBs in GranaSAT facilities is outspread over the following table

Item	Units	Cost/u. (€)	Cost (€)
Copper Clad Board	2	4.00	8.00
Nail Polish	1	1.85	1.85
Soldering Tin (100g)	1	9.99	9.99
CNC PCB Milling Renting	20 h	25.00 €/h	500.00
Press Drill Renting	15 h	15.00 €/h	225.00
Electricity Bill (Total kWh, Tax inc.)	5 kWh	0.22 €/kWh	1.10
Subtotal (before 21 % VAT):			**744.84 €**
Total (VAT included):			**902.36 €**

Table E.3 – *PCB Manufacturing Budget*

E.1.2.3 Gripping Arms & Reaction Wheels & Battery Platform

E.1.2.3.1 Bill of Materials

Item	Units	Cost/u. (€)	Cost (€)
PLA 1.75mm Plastic 1 Kg	1	13.00	13.00
Screws	24	0.40	9.60
Nuts	9	0.30	2.70
Vinyl Stickers	1	3.56	3.56
Subtotal (before 21 % VAT):			**28.86 €**
Total (VAT included):			**34.92 €**

Table E.4 – *Inertial System Components*

E.1.2.3.2 Manufacturing

Item	Units	Cost/u. (€)	Cost (€)
3D Prusa Printer Renting	30 h	20.00 €/h	600.00
Vinyl Machine Renting	1 h	15.00 €/h	15.00
Electricity Bill (Total kWh, Tax inc.)	0.7 kWh	0.22 €/kWh	0.16
Subtotal (before 21 % VAT):			**615.00 €**
Total (VAT included):			**744.31 €**

Table E.5 – *Inertial System Manufacturing*

The programs used to design the PCB and the gripping arms were Altium Designer and SolidWorks, which in order to be used there must be made a payment.

Item	Units	Cost/u. (€)	Cost (€)
Altium Designer 19	3	165.88 €/month	497.64
SolidWorks Student Edition	1	99.00	99.00
Subtotal (before 21 % VAT):			**596.64 €**
Total (VAT included):			**721.94 €**

Table E.6 – *IT Program Budget*

E.2 Human Resources Cost

The manufacturing was made by different people, such as an intern from a summer internship programme. Also, there were collaborators during the process. This will all be showed in the table

Finally, the project needed most of the time two figures, a junior Intern Engineer and a Project Supervisor, also senior engineer. The wages of these workers were $10_{\overline{h}}$ and $40_{\overline{h}}$, respectively

Post	Time (Hours)	Cost (€)	
Junior Engineer	1080	10800	
Senior Engineer	150	6000	
Subtotal (before 35 % Taxes):			**16800.00 €**
Total (Taxes included):			**22680.00 €**

Table E.7 – *Human Resources Cost*