GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL

“Scale boat guidance through mobile application”

AUTHOR:

Carlos Albacete Fuentes

SUPERVISED BY:

Prof. Andrés Roldán Aranda

DEPARTMENT:

Electronics and Computers Technologies

ACADEMIC COURSE:

2022/2023

“Scale boat guidance through a mobile application”

Carlos Albacete Fuentes

KEYWORDS:

Altium Designer® 19, H bridge, DC motor, servo motor, GPS, IMU, PCB design, C++, Arduino, ESP32, LED, MCU, GPRS, GSM, LTSpice®.

ABSTRACT:

Posidonia oceanica is a seagrass species that grows in Mediterranean regions and it is typically distributed in meadows at depths ranging from the water surface to thirty-five meters. Its importance lies in its capability to constitute a valuable ecosystem where a great variety of fauna and flora can settle and to indicate the environmental quality of the littoral.

This plant is periodically studied by the CEI · MAR · UGR divers and, in some cases, they can locate these meadows at first sigh from the shore or the water surface. However, in days when the seabed visibility is poor, this localization task can be quite challenging since the divers do not have any reference point that can guide them to the correct place.

The goal of this thesis is to provide divers with a maritime guidance device that facilitates the localization process. This will be implemented in an already-existing remote-controlled bait boat provided by our client, from which we will extract its original circuitry and replace it with a brand-new design, more modern and adapted to our project’s requirements, while taking advantage of some of its included features.

This design will allow the user to remotely indicate the GPS coordinates of the desired location to the boat through a mobile application via GPRS so it can navigate autonomously to said location, acting as a buoy that indicates the divers the correct immersion place. This circuit will be implemented in a multi-board PCB and controlled by an ESP32 MCU programmed in C++ using the Arduino framework.

“Scale boat guidance through a mobile application”

Carlos Albacete Fuentes

PALABRAS CLAVE:

Altium Designer® 19, puente H, motor DC, servomotor, GPS, IMU, diseño de PCB, C++, Arduino, ESP32, LED, MCU, GPRS, GSM, LTSpice®.

RESUMEN:

La Posidonia oceánica es una planta marítima que crece en el Mediterráneo y normalmente se distribuye en praderas cuya profundad varía desde la superficie hasta los treinta y cinco metros. Su importancia reside en su capacidad para consituir un valioso ecosistema donde puede instalarse una gran variedad de fauna y flora y en su capacidad para indicar el estado medioambiental litoral.

Esta planta es periódicamente estudiada por los submarinistas del Aula del Mar CEI · MAR · UGR y, en ocasiones, estas praderas pueden localizarse facilmente desde la costa o desde la superficie. Sin embargo, en los días en los que visibilidad del fondo marino no es buena, esta tarea de localización puede ser compleja ya que los buceadores no cuentan con ningún punto de referencia que les pueda guiar hasta el lugar correcto.

El objetivo de este Trabajo de Fin de Grado es proporcionar a los submarinistas un dispositivo de guiado marítimo que facilite el proceso de localización. Este será implementado en un barco cebador teledirigido ya existente que nos ha sido proporcionado por nuestro cliente, del cual extraeremos su electrónica original y la reemplazaremos por un nuevo diseño, más moderno y adaptado a las necesidades de nuestro proyecto, a la vez que aprovechamos algunos de los elementos ya incluidos en él.

Este diseño le permitirá al usuario indicar las coordenadas GPS de la localización deseada remotamente a través de una aplicación móvil por GPRS para que pueda navegar autónomamente a dicha localización, actuando como una boya que indica el lugar correcto de inmersión. Este circuito será implementado en una PCB multi-board y será controlado mediante el MCU ESP32, programado en C++ usando el framework de Arduino.

Agradecimientos:

Este trabajo ha sido posible gracias a un gran número de personas, a las cuales debo un gran agradecimiento.

En primer lugar a toda mi familia. Sin su educación, sus valores y su apoyo incondicional jamás habría llegado hasta donde estoy ahora mismo. Al igual que en el resto de mi vida, a lo largo de este proyecto han sabido aconsejarme, ayudarme y aportarme una visión más positiva en los momentos de mayor desesperación.

A mis amigos, por sus consejos, su apoyo, su comprensión y sobretodo por las risas, los buenos momentos y las lecciones que me llevo de cada uno. Sin ellos todo este trabajo se habría hecho mucho más cuesta arriba.

A todos mis compañeros de Granasat, por prestarme toda su ayuda y atención cuando más la necesitaba y por acompañarme y amenizar el trabajo durante incontables horas en el laboratorio.

Por último, he de agradecer a mi tutor, Andrés Roldán Aranda, que pese a mi falta de experiencia al inicio del proyecto, se mantuvo paciente y me ofreció la ayuda y la motivación necesarias para que todo saliese adelante.

Cada uno de los mencionados son partícipes de este trabajo. A todos ellos, y a los que no pudieron verlo, muchas gracias.

Contents

List of Figures

List of Tables

Glossary

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

Arduino Open-source hardware and software company that manufactures development boards that can be programmed in C++ using the Arduino language API..

C++ General purpose object-oriented programming language.

Charge Current (CA) Unit that expresses the charging or discharge current of a battery in relation to its nominal capacity. The current in Amperes can be obtained by multiplying the charge current by the nominal capacity..

Clamping voltage Maximum voltage that can pass through an surge protector or electrical circuit breaker before it restrics further voltage from passing through the circuit..

Dropout voltage Difference between the input and output voltage of the regulator.

Fourth-generation cellular network (4G) Cellular network technology standards completely based on the IP protocol. It offers data transfer rates higher than 100 Mbit/s in movement and 1 Gbit/s in rest..

GSM AT commands Set of instructions in the format of text strings that are used to control modems..

Hypertext Preprocessor (PHP) General-purpose server scripting language commonly use for website development.

Integrated circuit Circuit composed of passive and semiconductor elements that is fabricated on the surface of wafer and later encapsulated..

LTE 4G wireless standard that provides increased network capacity and speed for cellular devices compared to the previous cellular network technology generation..

LTSpice® Free electronic circuit simulation software produced by Analog Devices.

Microcontroller IC that contains one or more CPUs, memories and programmable input/output peripherals.

NMEA data Standard data format transmitted in ASCII strings that can be interpreted by most of the GPS receivers.

Reverse engineering Process trough which it is attempted to understand the principles of a system through analysis of its structure, function, and operation.

Saturn PCB Design Toolkit Freeware dedicated to PCB related calculations.

Second-generation cellular network (2G) Group of technology standards employed for cellular networks and it is characterized by its digital operation and smaller size..

Structured Query Language (SQL) Programming language used to manage data in relational databases..

Third-generation cellular network (3G) Network based on the IMT-2000 specifications of the International Telecommunication Union that offers a faster data transfer and better call quality than the previous generation..

Zener diode Type of diode capable of conducting currents either in a forward or reverse direction..

Acronyms

ABS Acrylonitrile Butadiene Styrene.

ADC Analog to Digital Converter.

BJT Bipolar Junction Transistor.

CDC Communication Device Class.

CEMF Counter-electromotive force.

COLREGs Convention on the International Regulations for Preventing Collisions at Sea.

CPU Central Processing Unit.

DAC Digital to Analog Converter.

EEPROM Electrically Erasable Programmable Read-Only Memory.

EMF Electromotive Force.

EMI Electromagnetic Interference.

EPROM Erasable Programmable Read-Only Memory.

ESD Electrostatic Discharge.

FSM Finite State Machine.

GPIO General Purpose Input/Output.

GPRS General Packet Radio Services.

GPS Global Positioning System.

GSM Global System for Mobile Communications.

HTTP Hypertext Transfer Protocol.

I2C Inter-Integrated Circuit.

I2S Inter-IC Sound.

IC Integrated Circuit.

IMU Inertial Measurement Unit.

IP Internet Protocol.

LCD Liquid Crystal Display.

LDO Low-Dropout Regulator.

LED Light-Emitting Diode.

MCU Microcontroller Unit.

MEMS Micro-Electromechanical Systems.

MOSFET Metal-oxide-semiconductor field-effect transistor.

OLED Organic Light-Emitting Diode.

PCB Printed Circuit Board.

PLED Polymer Light Emitting Diode.

PWM Pulse-Width Modulation.

ROM Read-Only Memory.

RTC Real Time Clock.

SMD Surface Mount Devices.

SPI Serial Peripheral Interface.

SRAM Static Random Access Memory.

UART Universal Asynchronous Receiver-Transmitter.

UGR University of Granada.

USB Universal Serial Bus.

VRLA Valve-Regulated Lead Acid.

Chapter 1

Introduction

The following Bachelor's Thesis was written as a the culmination of a four year degree in Industrial Electronics and as a continuation of previous thesis done by other students of the same degree. The goal of this project is to provide the CEI · MAR UGR divers with a device that can guide them to underwater Posidonia meadows.

This thesis originated and was developed in Granasat, a multidisciplinary group from the University of Granada (UGR) originally focused on aerospace electronics, made up entirely for students and under the supervision of the professor Dr. Andrés María Roldán Aranda.

Figure 1.1 – *Granasat logo*

The Granasat project started in 2013, when several students who were interested in the aerospace field decided to participate in the BEXUS/REXUS programme, which allows students from universities and higher education colleges across Europe to carry out scientific and technological experiments on research rockets and balloons.

Since then, Granasat's scope has widen, collaborating and involving in wide ranging projects with students and professionals from different areas of science and engineering. This is the case of this project, which originated as a request from the CEI · MAR UGR program.

1.1 Motivation

Since offering a fully functional and validated product in a year's time lapse is not realistic due to the learning process that this project requires for an inexperienced student and the encountered obstacles, our main motivation is to make as much progress as possible while minimizing the errors, providing a reliable and tested starting point for following advances.

From a personal perspective, this project is an opportunity to apply and test all the theoretical knowledge I have acquired through my degree, specially in the circuit design, firmware development and mechanical fields, and what is more important, to keep learning and improve my engineering and problem-solving skills.

1.2 Project objectives

The task of developing an autonomous navigation ship is complex and can be overwhelming when considered as a whole. For that reason, when developing a large project it is important to divide it into smaller tasks, which must be specific and realistic so we can evaluate our progress and avoid losing track. Some of these tasks are completely independent from the others, however others rely on data that has to be previously obtained so it is important to take into consideration the realization order.

From a technical standpoint, the project goals in order are:

- Boat's detailed analysis: the scale boat that we are using is a remote controlled bait boat called Pondskater Orca. Most of its circuitry has been removed so it can be replaced by ours but some of its elements such as the motors and the LEDs will be used for our application. In the previous thesis this elements were not carefully analyzed so their technical specifications were estimated or directly supposed. It is essential to dispose of a precise characterization of every component when it comes to the circuit design since the overestimation or underestimation of some of their parameters could permanently damage the component or the full circuit.

- DC motor modelling: a mathematical model of the motor will be obtained , which is essential achieve a precise and efficient control. It will allow us to estimate the motor's response to different inputs, to estimate the values of different parameters without the necessity to experimentally measure them and moreover, it will provide us with a better understanding of the motor operation.

- Schematic revision: since the parameters of the boat's components may be mistaken it is possible that those mistakes have been translated into the circuit design. In addition, due to the time constraints of the previous thesis it is possible that there are some unnoticed design errors that could compromise the correct operation of the device.

- Firmware development: This process must be done throughout the project, preferably in different parts so they can be used to test the different schematics and, once every separate part is complete, joint them all in a single program.

- Establishing communication between the boat and the mobile app: it is one of the most crucial parts of the firmware and the project, it will be achieved through a GSM/GPRS module. It will also require the creation of a mobile app and a server acts as a broker between the module and the app.

- Schematic testing: once the schematics have been corrected, they must be tested within our possibilities to ensure their proper operation. Since we do not dispose of the finished circuit at this stage only certain portions can be tested and has to be done separately. In the cases that require testing but it cannot be done in practice a circuit simulation software will be used.

- PCB revision: due to the corrections and modifications of our circuit it will be mandatory to redesign the PCB. Moreover, it will be revised to detect possible design errors and potentially find a more optimal organization of the components.

- PCB manufacturing and testing: the PCB board will be manufactured and delivered to our laboratory. However, all the components must be manually welded and it must be tested to ensure that it works properly from an electrical standpoint.

1.3 Chapter Description

Following the objectives listed above , the project was developed after planning the next structure:

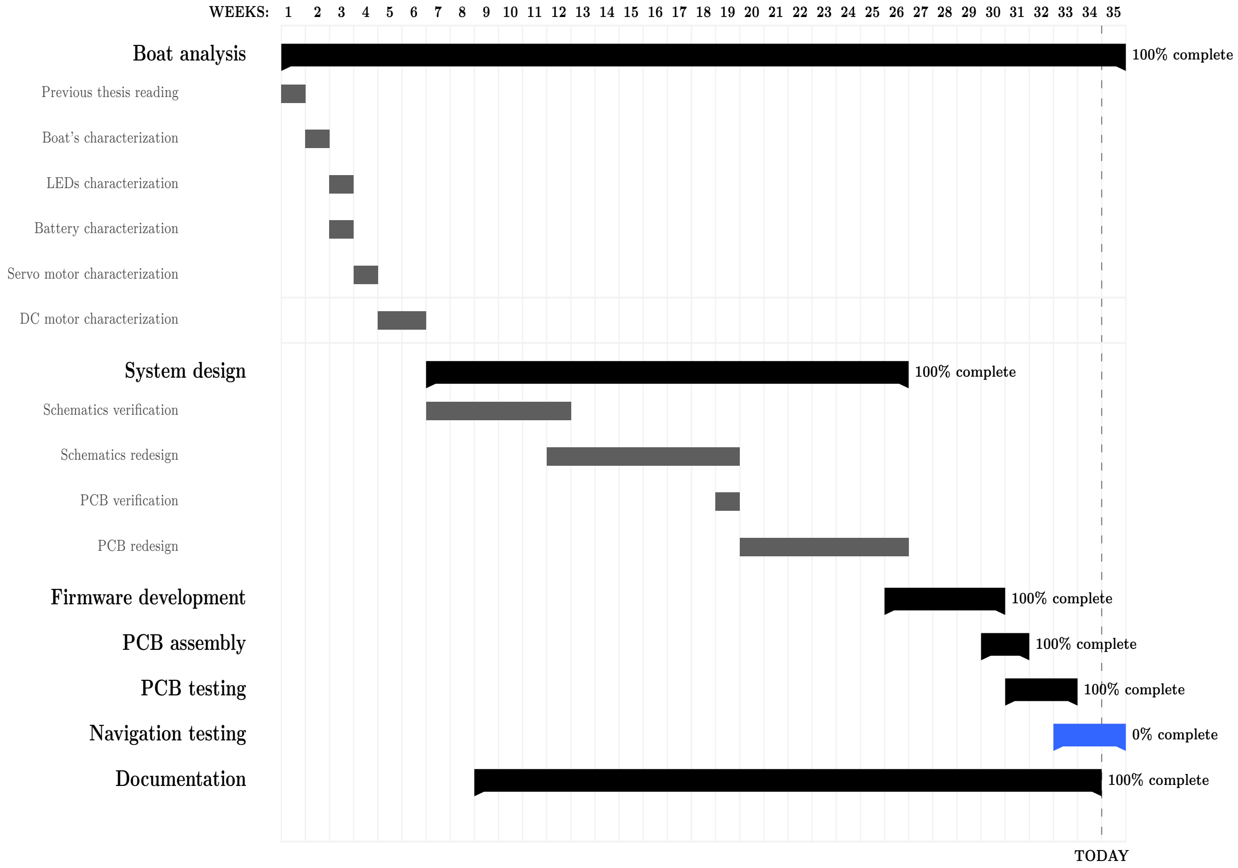
- **Chapter 2: Project planning.**
  In this chapter, the specific tasks that are required to accomplish our objectives and the time that will be dedicated to them will be described through a Gantt diagram.
- **Chapter 3: Boat analysis.**
  In this chapter we will analyze the boat which will be used as our first prototype and its different electrical and mechanical elements. The main purpose is to check if these elements are in good conditions and obtain their technical specifications, either by their documentation or by experimentation.
- **Chapter 4: System requirements and constraints.**
  Following the technical characterization of the bait boat previous study, this chapter will identify our product's requirements. These will be split in functional, performance and design requirements and will serve as a guide during the design process.
- **Chapter 5: System description and design.**
  In this chapter the design of the product will be described thoroughly. The main parts will be electronics, PCB and firmware. For each of these sections, we will explain the design process and describe the testing and verification results when possible.

  It is important to keep in mind that during this chapter, we will primarily focus on the design process and the reasoning behind our decisions, while leaving the most specific details and final results for the Appendix.
- **Chapter 6: Conclusions, future work and lessons learned.**
  In the last chapter, we will reflect on the project and draw conclusions. Moreover, future lines of work and improvements that need to be done in further theses will be summarized.

Chapter 2

Project planning

Before fully immersing in the project, it is mandatory to define the specif set of tasks that must be accomplished before the deadline and the amount of time that it will be dedicated to them. This can be visually represented with a Gantt diagram.



Chapter 3

Boat analysis

3.1 The boat

This prototype is based in an already existing scale boat, the Pondskater Orca, shown in figure 3.1. This was originally designed as a remote controlled bait boat, but for our prototype all the original circuitry will be removed and replaced by ours, more modern and with additional functionalities. Even though the original boat's design could act as an useful guidance device in some occasions it is not the most optimal solution since it has to be manually controlled by the user until it reaches the desired location, which is difficult to determine since it did not feature GPS tracker. Its main characteristics are listed in table 3.1.

Name	Pondskater Orca
Type	Bait boat
Dimensions	55 x 27 x 28 cm^3
Weight	3.4 kg (without the battery)
Material	ABS and reinforced nylon
Battery	Lead acid 6 V 7Ah

Table 3.1 – Pondskater Orca specifications

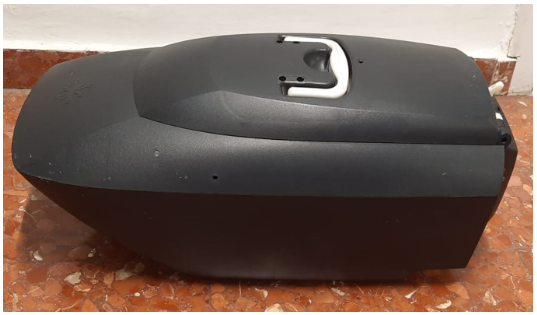


Figure 3.1 – Pondskater Orca

This boat contains certain components that will be used in our design: a DC motor, a servo motor, a 6 V lead acid battery and various LEDs. The exact model of some of these components is unknown and cannot be found neither in the Ponskater Orca documentation nor online so in order to obtain their exact technical specifications they will be experimentally tested.

3.2 The LEDs

This boat contains eight LEDs: one of them is a high power LED and the rest are 5 mm LEDs. The high power one is white and placed in the bow, it will be useful to light up the boat's path and to make the it visible from other vessels and from the shore at night. Since this LED is a high power one, it cannot be tested with the diode mode of the multimeter. For that reason, this LED will be tested by placing a low value resistor in series with the LED an applying a sufficiently high voltage for the diode to conduct. The test circuit is shown in figure 3.2.

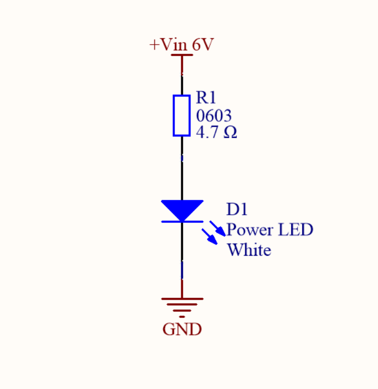
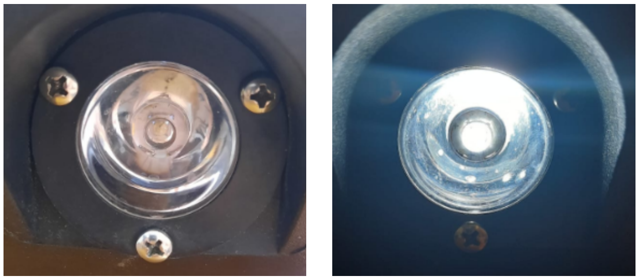


Figure 3.2 – High power LED test circuit

When the source voltage is higher than 2.5 V the LED turns on, verifying that it works (figure 3.3). With this circuit, by varying the source voltage we can obtain the forward current vs forward voltage curve of the LED, displayed in figure 3.4.



(a) LED off (b) LED on

Figure 3.3 – Bow's high power LED

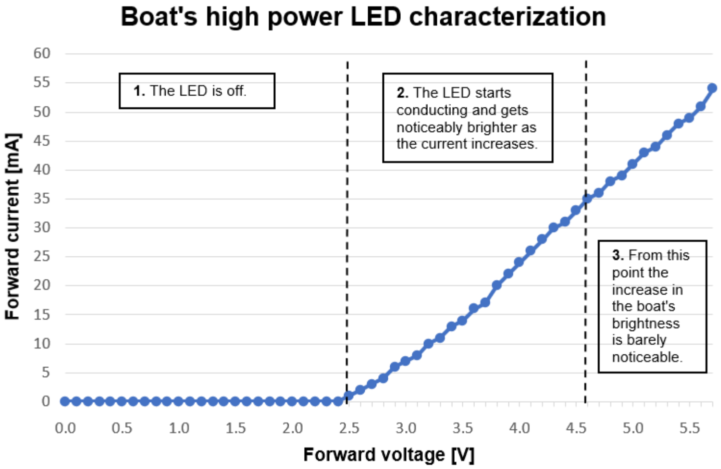


Figure 3.4 – Forward current vs forward voltage curve of the high power LED

We observe that the diode starts conducting at 2.5 V indeed and as the voltage is increased so does the current. The last detail that must be checked is the brightness, we must ensure that we obtain the maximum brightness with the lowest current possible. In order to achieve that, the source voltage will be increased gradually and the light brightness will be observed, the brightness will increase up to a certain point where the changes in luminosity as the voltage increases will be barely noticeable. The current at which this happens is the current that will be used to design this LED circuit. This happens at 35 mA, which will be taken into consideration during the circuit design process (section 5.2.6).

Regarding the 5 mm LEDs, there are seven and they will be useful to alert our boat's presence to other vessels. They are distributed as it follows:

- A blue and a red LED on the port side.
- A green LED on the starboard.
- An orange and a red LED on the port side of the stern and an orange and a green LED in the starboard of the stern. This differs from the previous thesis, that stated that those orange LEDs were white.

All these LEDs are regular 5 mm LEDs so they can be tested with the diode function of the multimeter. The results are:

- The blue LED on the port side works fine, with its anode is connected to a black cable and the cathode connected to a red cable, the red LED does not work.
- The green LED on the starboard does not work.
- All the LEDs on the stern work fine: the orange LEDs are connected in parallel with their cathodes connected to a red cable and their anodes connected to a black cable, the green LED has its cathode connected to a brown cable and its anode connected to a white cable and the red LED has its cathode connected to a white cable and its anode connected to a black cable.

The broken LEDs will be removed and replaced by new high intensity LEDs, so they are visible from a longer distance.

3.3 The battery

The boat contains a 6 V and 7 Ah lead acid battery that will supply all the circuitry. The exact model is the MS7-6 from MHB, which is a Valve-Regulated Lead Acid (VRLA) battery. Lead acid batteries are one of the most widely used rechargeable batteries due to their affordable price, their capability to provide currents up to hundreds of amperes and their good charge retention qualities. One of the downsides of this battery type is that they have a relatively low energy density compared to others [12], however, in our application this will not be a problem since the boat is has plenty of space inside.

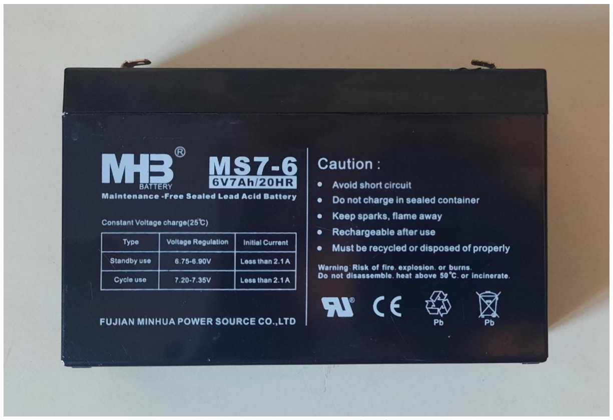

MHB®
BATTERY
MS7-6
6V7Ah/20HR
Maintenance -Free Sealed Lead Acid Battery
Constant Voltage charge(25℃)
Type
Voltage Regulation
Initial Current
Standby use
6.75-6.90V
Less than 2.1 A
Cycle use
7.20-7.35V
Less than 2.1 A
FUJIAN MINHUA POWER SOURCE CO.,LTD
Caution :
Avoid short circuit
Do not charge in sealed container
Keep sparks, flame away
Rechargeable after use
Must be recycled or disposed of properly
Warning: Risk of fire. explosion. or burns.
Do not disassemble. heat above 50℃. or incinerate.
CE
Pb
Pb

Figure 3.5 – Boat's battery

In general terms, this battery is formed by a positive lead dioxide electrode, a lead negative one and a sulfuric acid electrolyte diluted in water. When it is discharges, the electrodes convert into Lead(II) sulfate and the electrolyte converts into water. When it charges, the reaction is the same but reversed [20]. This has a problem: the conversion of water into sulfuric acid during the charge generates vapour that can escape the battery cell, this is solved with the VRLA type batteries. This type incorporates a mechanism that coverts the vapour into water and since they are sealed, it is not necessary to refill them with water.

The technical specifications of our battery are listed in table 3.2.

Name	MS7-6 Datasheet
Type	VRLA battery
Nominal voltage	6 V
Capacity	7 Ah
Dimensions	15.1 x 9.8 x 3.5 cm^3
Weight	1.17 kg
Charge voltage	Cycle use: 7.2 to 7.5 V Float use: 6.75 to 6.9 V
Self discharge	2% of capacity decrease per month

Table 3.2 – MS7-6 battery specifications

Even though the nominal voltage is indicated to be 6 V, it must be taken into consideration that its operating range actually goes from 6.4 to 7.5 V. Its battery should never be lower than 6.4 V as it could cause irreversible damages.

The battery was replaced last year so we could assume that it works, nevertheless, it has been tested by measuring the voltage between the terminals with a multimeter and by charging it with the VOLTCRAFT ALC 8500 EXPERT charging station. The voltage of the battery was 6.4 V and it reaches 7.5 V when it is being charged as it is specified its the datasheet so we can conclude that the battery is in good conditions.

3.4 The servo motor

It is a type of motor that implements a feedback loop which allows it to rotate with high precision. It is typically used to achieve a rotation at a specific speed or angle and stay in that position. Their feedback loop is implemented through an encoder, which measures the position and the rotational speed and a servo drive that reads the encoder's measurements and makes control adjustments [33].

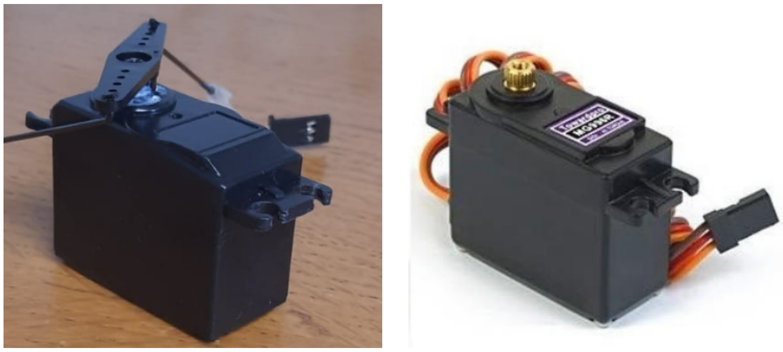
This boat contains two servo motors, one of them is placed on the top part of the boat and it opens and closes the compartment where the bait is stored, the other one is inside the boat and controls the rudder. The first one will not be used for this prototype since this boat will only be used for guidance purposes but the second one is essential for our application as it controls the boat's direction. Moreover, it is important to mention that these two motors only have a 180º rotation range.

The characteristics of the motor that are required for our project are:

- The supply voltage.
- The control pin voltage.
- The current consumption both outside and under the water, as well as the stall current.
- The maximum angles that the rudder can reach.
- The relationship between the angles that will be specified when programming the motor using the servo.h library and the angle that the rudder actually reaches.

At a first sight, we can observe that the manufacturer and the model are indicated in a sticker, the manufacturer is EOGB and the model is T600A. Nevertheless, after an exhaustive online search it has been impossible to find any datasheets nor documentation for this model. For this reason, all those characteristics will have to be obtained experimentally.

In relation to the supply voltage, the most simple method would be to supply the motor at arbitrary voltages, vary them and get its operating range. However, this could be dangerous because we could choose a higher voltage than its absolute maximum and provoke permanent damages to the motor. The safest way to proceed is to find similar servo motors models and check their specifications, which can serve as a good reference for the voltage range that we could use to test the motor. After searching for motors with similar dimensions to ours we have come across the MG996R model, which has the exact same dimensions than ours and a very similar appearance. A comparison of these two motors is displayed in figure 3.6 and the most relevant specifications of the MG996R are listed in table 3.3.



(a) Boat's servo motor **(b)** MG996R

Figure 3.6 – Servo motors comparison

Name	MG996R Datasheet
Type	Servo motor
Supply voltage range	4.8 to 7.2 V
Running current at 6 V	500 to 900 mA
Stall current at 6 V	2.5 A
Dimensions	40.7 x 19.7 x 42.9 mm^3

Table 3.3 – MG996R specifications

In the MG996R datasheet [2] it is specified that it can operate in a 4.8 to 7.2 V supply voltage range but there is no available information regarding the control voltage logic level. Typically, the same voltage level is used for the power supply and the control pin but this would be harder to achieve since the microcontroller that will be used can only provide up to 3.3 V (this will be explained in detail in 5.2.1). For this reason it is required to check if the motor can work correctly controlling it with 3.3 V, we can try this experimentally because applying a lower voltage than the operating range could possibly slow down its operation but not damage it.

Knowing this, we have tested the motor at 6 V (the battery's nominal voltage) with a NodeMCU-32 board and a simple Arduino code (appendix C.14).

The MG996R datasheet also mentions that the running current ranges from 500 to 900 mA and the stall current is 2.5 A at 6 V. Assuming that this data corresponds to our motor's operation (which could not be the case) it could serve as an useful reference if the motor worked outside the water, however, when it is introduced in the water, the mechanical drag load increases and so does the current, then that data would not be valid. Therefore, it is mandatory to measure the current consumption of our motor in underwater conditions.

To measure the current the N6705A DC power analyzer will be used, which allows us to supply up to 600 W and has the capability of an oscilloscope and data logger [4]. The test will consist in moving the motor from 0 to 180º (its full motion range) **gradually** and then back to 0º **abruptly** in different conditions:

- Without external mechanical drag load: this refers to the boat outside the water. The results are shown in figure 3.7.

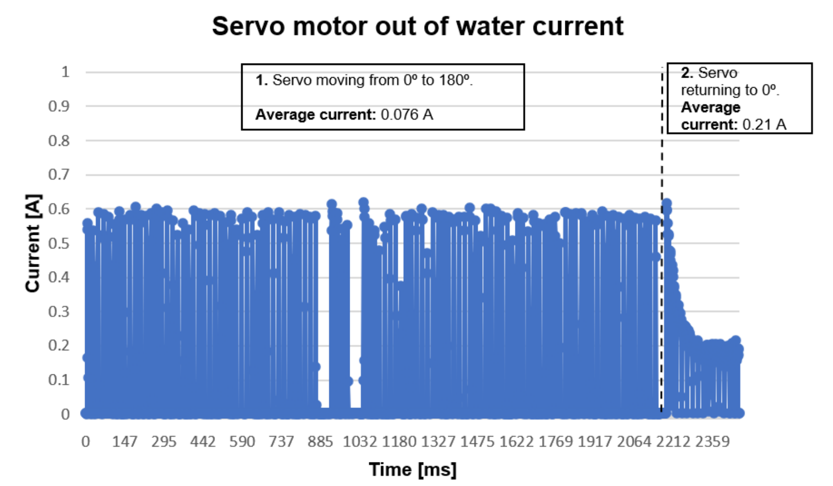


Figure 3.7 – Servo motor current without external mechanical drag load

We can appreciate that the current pulses' amplitude when it moves from 0º to 180º are higher than when it returns to 0º. The reason for this is that the Electromotive Force (EMF) is proportional to the rotational speed, therefore, the higher the motor's speed is, the larger the EMF will become and the smaller the voltage will be across the motor's armature resistance resulting in a lower current. Although the current's pulses are lower as the speed increases, the pulse rate becomes higher since it requires more power to overcome the load torque. The average current when the motor is moving slowly is 76 mA and it is 220 mA when the motor moves as fast as possible. The maximum peak current is 0.6 A.

- With the expected mechanical drag load: this refers to the typical operating conditions of the boat, meaning that the rudder is underwater without movement restriction. The results are displayed in figure 3.8.

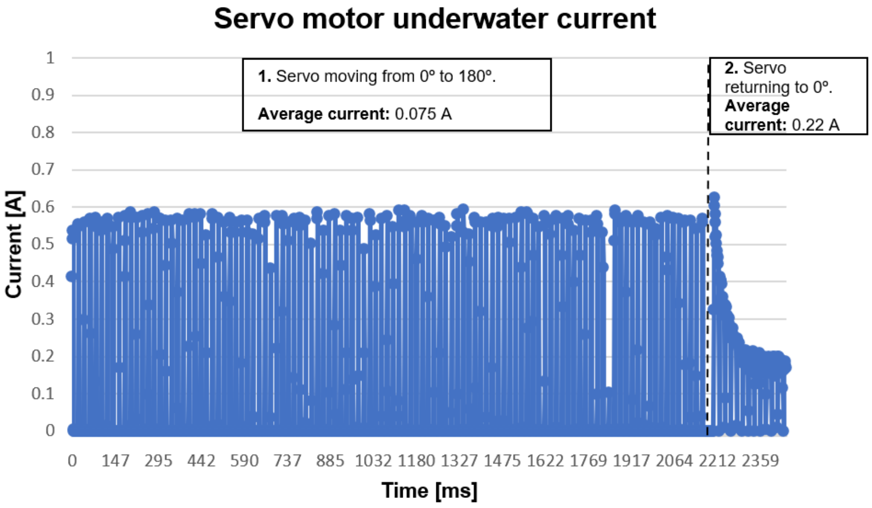


Figure 3.8 – Servo motor current consumption with the rudder underwater

In view of the obtained results, we can conclude that the effect of the water's friction will not have remarkable effects the servo motor's consumption, only a 1% decrease in the average current when it is moving gradually and a 4.7% increase when it is moving fast.

- With the maximum mechanic drag load: this refers to the case when the rudder is immobilized, this current is referred as the stall current. This simulates a situation where the rudder has been blocked by an object or an obstacle. The results are shown in figure 3.9.

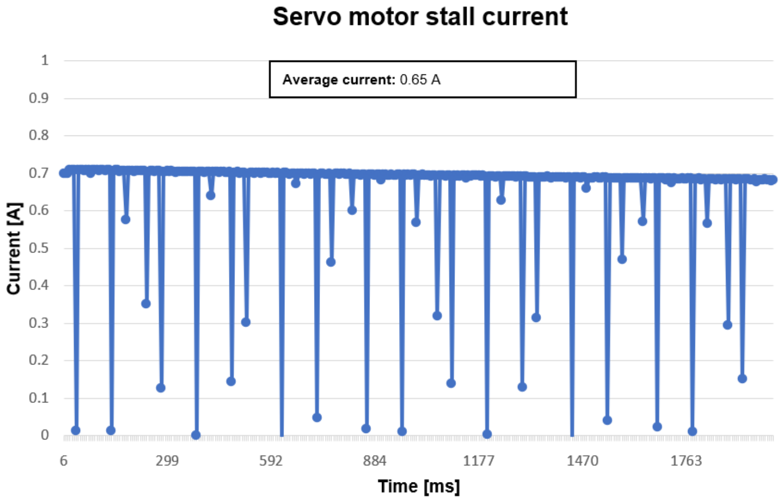


Figure 3.9 – Servo motor current consumption with the rudder blocked

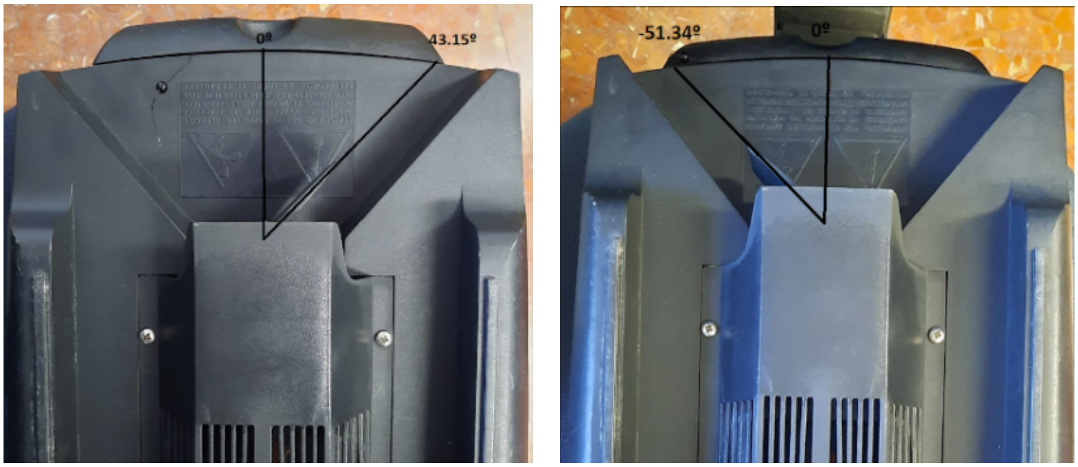
In this case, there is no rotational speed so the current will considerably increase and it will be almost constant since the load torque is too large. The average current is 0.7 A.

In conclusion, the peak running currents are in the same range as the MG669R model but we have determined the average currents, which are much lower than the peaks, and moreover, the stall current, which is significantly lower than the MG669R's. The technical specifications of our boat's servo are listed in table 3.4.

Name	EOGB T600A
Type	Servo motor
Supply voltage	6 V
Peak running current at 6 V	Outside water: 600 mA Underwater: 600 mA
Maximum average running current	Outside water: 210 mA Underwater: 220 mA
Stall current at 6 V	Average: 650 mA Peak: 700 mA
Dimensions	40.7 x 19.7 x 42.9 mm^3

Table 3.4 – Boat's servo motor specifications

Regarding the rudder's angles range, we have tested it by measuring its angles when the servo motor is set to 0º and when it is set to 180º. When the motor is set to 0º the rudder angle is is 43.15º and when it is set to 180º is -51.34º as it can be observed in figure 3.10.



(a) Rudder when the servo motor is set to 0º **(b)** Rudder when the servo motor is set to 180º

Figure 3.10 – Rudder's angle range

The maximum angle's range is not symmetrical, then we have to limit the range to ± 43.15º so the boat does not turn to one side more than the other when manoeuvring. We have experimentally found the servo motor angle for the rudder to be at -43.15º and for it to be at 0º and the results are presented in table 3.5.

Servo.h library angle	Rudder's angle
0º	43.15º
180º	-51.34º
70º	0º
145º	-43.15º

Table 3.5 – Relationship between the servo and the rudder angle

3.5 The DC motor

It is a rotating electrical machine capable of transforming electrical energy in the form of a DC current into mechanical energy. They are widely used because of how easy it is to control their speed, just by varying the current, and to control their direction of rotation, just by inverting the direction of the current flow. Generally, all DC motors are composed of the following parts, which can be observed in figure 3.11:

- Stator: fixed part of the motor, usually formed by magnets or by windings of copper wire over an iron core.
- Rotor: rotating mobile part of the motor, also composed of windings and a core.
- Air gap: separates stator from rotor, allowing this last one to move.
- Brushes: they act as an electric contact between the stator and the rotor.
- Commutators: they switch the direction of the magnetic field in order to keep a constant torque direction.

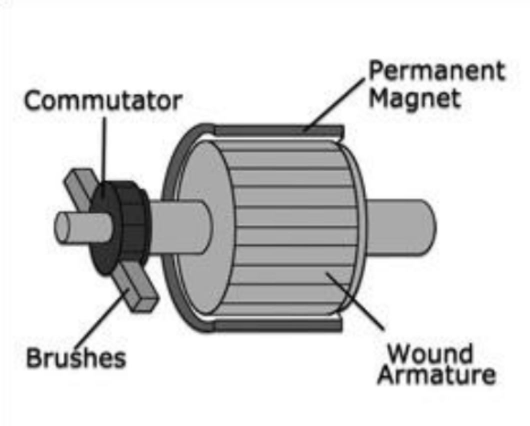


Figure 3.11 – Parts of a brushed DC motor [50]

The boat contains one DC motor which is used to propel the boat. It transfers the rotational speed to the propeller through a drive shaft and this shaft is connected to the motor through a metallic ring (see figure 3.12). In order to analyze it properly it will be dismantled from its mount (figure 3.13). Looking carefully we can observe that it has two 100 nF capacitors welded between the terminals (figure 3.14), they are used to soften the start and the stopping by absorbing current when the motor starts and giving up current when it stops (this will be discussed in more detail in section 5.2.12). Moreover, connected to each terminal, there are two coils whose value will be obtained in section 3.5.1. Their purpose is to filter out the high frequency noise at the motor input.

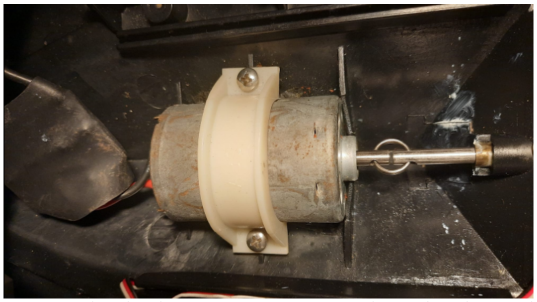


Figure 3.12 – Our DC motor inside the boat, it is connected to the drive shaft through a metallic ring

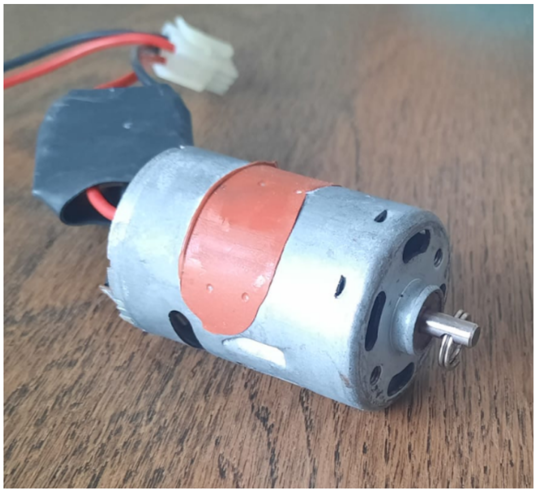
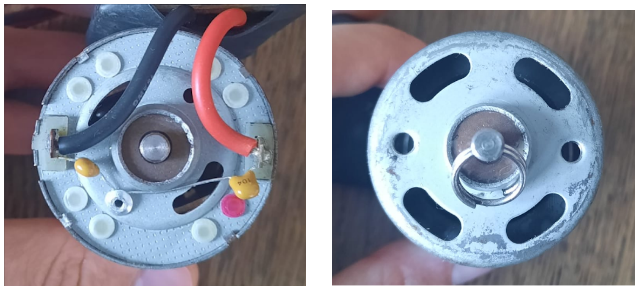


Figure 3.13 – Our DC motor outside the boat



(a) Front view **(b)** Rear view

Figure 3.14 – DC motor from different angles

The DC motor's dimensions can be seen in figure 3.15, they were extracted from the K_3SFN series model, which has the same standardized dimensions as our motor.

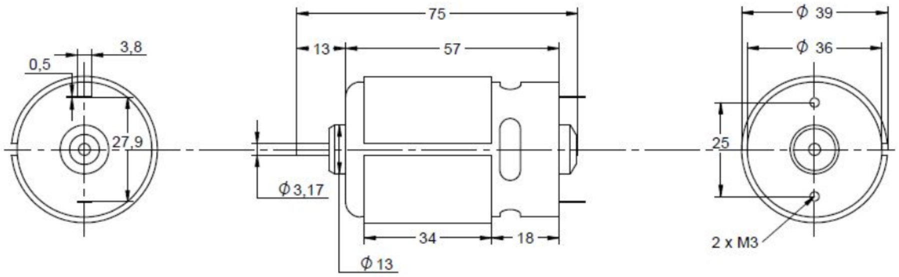


Figure 3.15 – DC motor dimensions [30]

This motor does not have any labels or marks that can help identify its exact model and there is no information about it in the boat's documentation either. Therefore, it was necessary to obtain most of its characteristics through experimentation or assumption. The technical specifications of this motor that are required for our project are:

- The supply voltage.
- The current consumption both outside and under the water with different mechanical loads.
- The lineal model.

Firstly, the supply voltage of the motor has been supposed to be around 6 V, which is the nominal tension of the battery that supplies the whole boat's circuitry and it is also a typical nominal value for DC motors of these dimensions. The motor has been tested with this voltage and has proven to work fine for long period of times without signs of overheating, which is a good sign.

Regarding the current consumption, it will be measured with the N6705A DC power analyzer. This is a complex parameter because it will depend on the motor's mechanical drag load, the larger this load is, the higher the current will become in order to overcome the load's torque. For our application, we can distinguish three different situations of interest:

- Without external mechanical drag load: this refers to the boat out of the water. Technically, in this case there's actually a light load, the one from the transmission shaft that goes from the motor to the propeller, although by properly greasing it, the resistance that it offers is negligible. In this case the average current of the motor is 1.883 A as it is shown in figure 3.16.

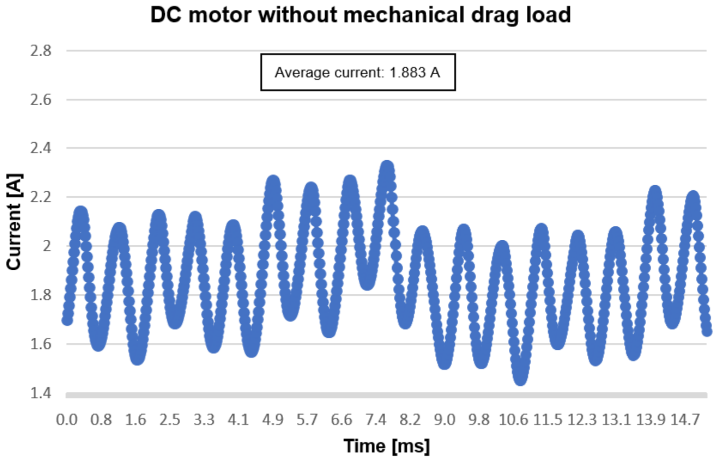


Figure 3.16 – DC motor current without load

- With the expected mechanical drag load: this refers to the typical operating conditions of the boat, meaning that it is in the water without movement restriction and carrying the weight of the battery. In order to measure the current in this conditions, the boat was placed in a sufficiently large pool so it can move freely and the motor was connected to the power analyzer by two five meters long cables. These measurements have been taken in a large enough pool so the boat can move freely.

  When trying to obtain these measurements a problem was encountered: when the boat was placed in the water the motor did not work properly, its speed was too low and could not move through the water. Since the motor had been tested and proven to work well then the problem had to be the drive shaft, specifically in its lubrication. Drive shafts require some type of lubricant (typically grease) to reduce the mechanical drag load and to protect them from water. Our boat's shaft was already lubricated, however, this boat has not been used nor lubricated in years and it is possible that its grease had hardened or deteriorated due to dust accumulation, increasing the shaft's mechanical drag load. This may not be a problem outside the water but in underwater conditions the total mechanical drag load is too high for the motor to overcome the load's torque.

  To fix this, the transmission shaft and the propeller had to be removed and l ubricated. This required to remove the servomotor and the rudder first because they were blocking the propeller's p osition. Once the shaft was removed it revealed a considerable lack of grease (figure 3.17) so it had to be regreased. After researching different types of greases and consulting multiple modeling forums and professionals the best option for this case is white lithium greases as it is durable, water-resistant and prevents rusting and corrosion.

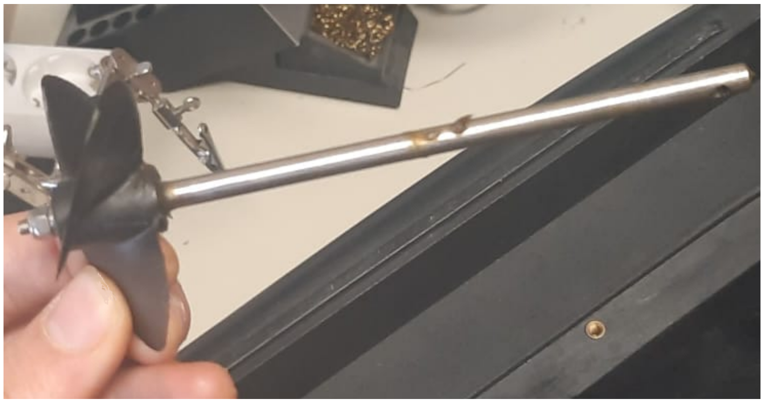


Figure 3.17 – Boat's propeller and drive shaft

After regreasing the motor its performance increased drastically being able to move fast through the water. Now, the average current of the motor is 9.6 A, its curve can be observed in figure 3.18.

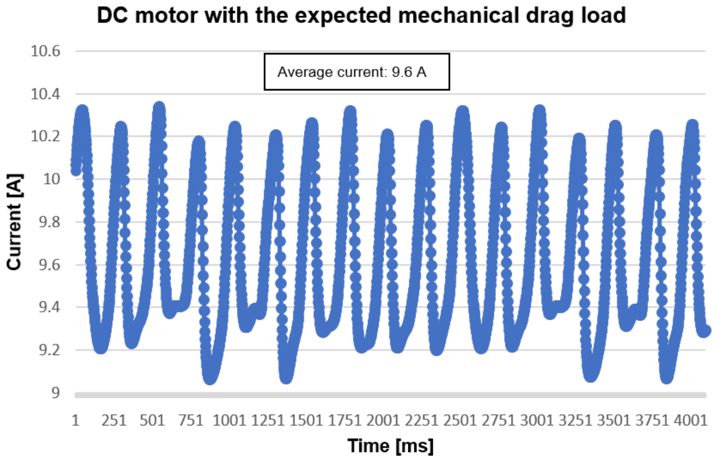


Figure 3.18 – Dc motor current with the expected load

- With the maximum mechanical drag load: this refers to the case when the boat is in the water, adding the weight of the battery but totally restricting its movement. In this case, the load torque that the motor has to overcome is too large, for this reason the rotational speed will decrease and the current will greatly increase. We have tested the motor in this condition limiting the current from the power supply to 11 A and the motor reaches that current so it is probable that the motor would consume even higher currents if we allowed it. We have stopped at that current because that would be 66 W already and we are unaware of the maximum power that the motor can handle, allowing higher currents could be dangerous.

  We must avoid this situation because, when supplying the motor from the battery, it could consume hundreds of watts which would most likely damage the motor due to extreme heating and discharge our battery too quickly.

We have observed that there is a great difference in the DC motors current depending on the mechanical drag load. This difference can be seen in figure 3.19, in which we turned on the motor outside the water and then we introduced it in a pool. These current values differ from the previous thesis that stated that the maximum current was 5 A and did not make any distinctions between these different situations.

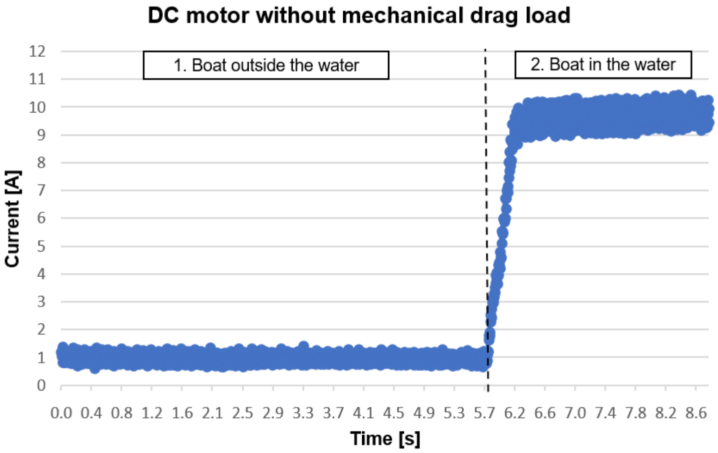


Figure 3.19 – DC motor current difference between mechanical drag loads

Moreover, while testing the DC motor in the pool, the boat's speed in ideal conditions with the motor working at full speed was measured. It covered an 8 m distance in 12.92 s, resulting in a 0.62 m/s speed. This result will be used to estimate the time the boat will take to cover a distance in the navigation algorithm development.

3.5.1 Modeling of the DC motor

Obtaining the mathematical model of the mechanical elements of our system is an essential requirement in order to achieve a precise and efficient control system. It will allow us to estimate the motor's response to different inputs, to estimate the values of different parameters without the necessity to experimentally measure them and moreover, it will provide us with a better understanding of the motor operation from a electromechanical standpoint.

The first step is to determine which type of DC motor we are working with because depending on that the modeling process will differ. As the specifications of our motor are not available we have to determine its type by observing it thoroughly. Through the small holes of the motor housing we can distinguish some of the parts of the motor: in figure 3.20 we can see the commutator, in figure 3.21 we can observe some of the windings, in figure 3.22 we can see the brushes and in figure 3.23 we can see fixed magnets in the stator, which are only found in the permanent magnets DC motors.

Actually, there are different types of permanent magnet motors, they can be brushless or not, in this case it has brushes. What characterizes these kinds of motors is that the magnetic field is continuous and created by those fixed magnets placed in the stator. The windings are only placed in the rotor and connected to the commutators, which are connected to the brushes, and the current will flow from the power source to the armature through the them [24].

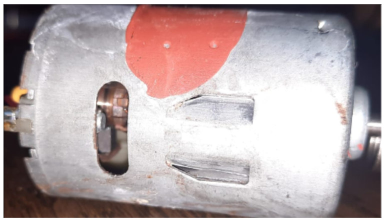


Figure 3.20 – Commutator

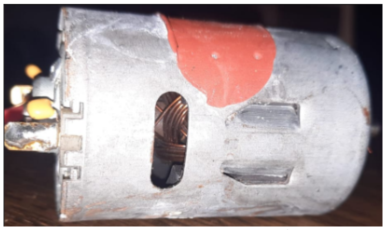


Figure 3.21 – Winding

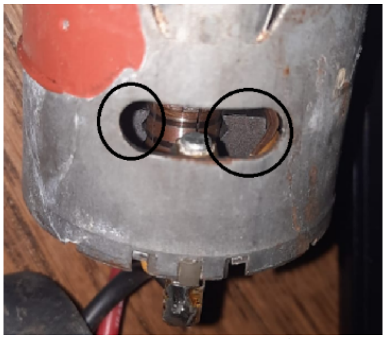


Figure 3.22 – Brushes

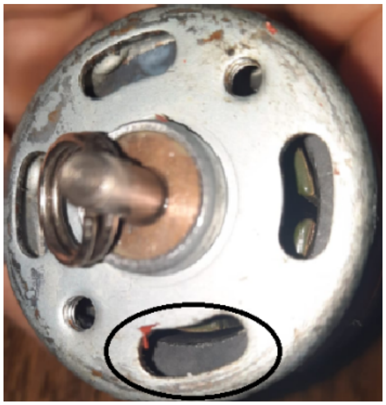


Figure 3.23 – Permanent magnets

Now that the DC motor type is known we can start the modeling process. The model of a DC motor is composed of an electrical part 3.5.2 and a mechanical part 3.5.16. The electrical part represents the inductance, resistance and the EMF where as the mechanical part represents the motor torque, the viscous friction, the rotor's inertial moment and rotational speed. Normally all these values are listed in the datasheet but in this case they will be obtained experimentally. One important consideration is that if we are going to use this model to estimate the motor's behaviour when the boat is on the water, then all these values must be obtained in those conditions since they change brusquely from one conditions to another.

$$\text{Mechanical equation: } L \cdot \frac{\delta\omega(t)}{\delta t} + B \cdot \omega(t) + \tau_f = \tau_m(t) \tag{3.5.1}$$

$$\text{Electrical equation: } L \cdot \frac{\delta i(t)}{\delta t} + R \cdot i(t) + E_a(t) = v(t) \tag{3.5.2}$$

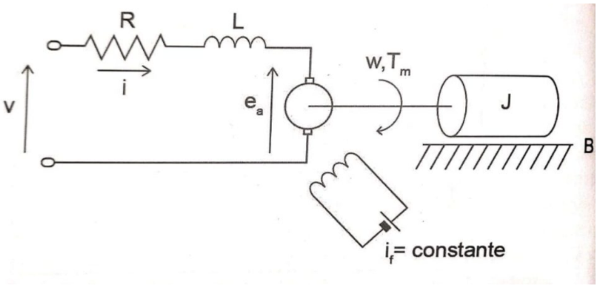


Figure 3.24 – DC motor's model circuit [9]

The required parameters are:

- Armature resistance (R): the armature of a DC motor is a part of the rotor which consists in an iron core that contains the copper windings that carry the current. To measure its resistance a multimeter is placed between the terminals of the motor. However, it has to be taken into consideration that, depending on the rotor's position, we are measuring the contact of a specific winding with the brushes, which may cause differences in the measured resistance due to the presence of different amounts of cinder. The bigger the amounts of cinder, the higher the resistance will be.

  To carry out this measurement correctly, a measurement must be obtained for each position of the rotor and then calculate the average value. In this case sixteen different measurements have been taken (table 3.6) and their average value is 0.51 Ω.

- Armature inductance (L): just as it was done with the armature resistance, different measurements have to be obtained for the different positions of the rotor and then calculate the average value. To measure the inductance we have used the precision LCR meter Keysight E4980A/AL.

  When measuring small inductances, the manual recommends to use the Ls-Q function [22]. Furthermore, we have to select a measuring frequency and current: the frequency will be 20 Hz, the lowest possible and the most similar to the one that the motor will work at, and the current will be 10 mA, the current's value does not matter because we are using 0.5 mm^2 cables that are thick enough to not heat up nor affect the measuring.

  After obtaining 16 measures, the average value of the inductance is 185.41 µH. It has to be taken into consideration that the motor has a 100 nF capacitor welded to each terminal. These have been included in the measure due to the difficulty to remove them and to weld them again.

  As it was mentioned before, there is a 64,17 µH coil in series with each terminal, which reduces the high frequency noise at the input of the motor. These coils will not be included in the motor model, instead they will be added in series with the motor model schematic.

Rotor's position	Resistance [Ω]	Inductance [μH]
1	0.4	191.2
2	0.5	187.68
3	0.5	190.6
4	0.4	172.76
5	0.3	189.4
6	0.6	173.79
7	0.5	183.34
8	0.6	186.17
9	0.5	176.68
10	0.5	192.55
11	0.6	188.52
12	0.6	192.82
13	0.6	187.72
14	0.5	175.37
15	0.5	189.11
16	0.6	189.33
Average	0.51	185.41

Table 3.6 – Armature's resistances and inductances

- Counter-electromotive force (CEMF) constant (K_a): This constant, also known as back-EMF constant, relates the induced CEMF to the rotational speed of the motor. It is typically expressed in $[\frac{V}{\text{rad}\cdot\text{s}}]$ and its formula is:

  $$K_a = \frac{E_a}{\omega} \tag{3.5.3}$$

  Where:
  K_a is the CEMF constant
  E_a is the CEMF
  ω is the rotational speed

  When a current is supplied to the windings of the motor, which are placed in a magnetic field, a torque force is induced on the windings due to Lorentz law. When these start rotating they are cutting the flux of the magnetic field which induces a CEMF in the opposite direction of the applied voltage. The CEMF of a DC motor can be obtained with the following expression [8]:

  $$E_a = V - IR \tag{3.5.4}$$

  Where:
  R is the armature resistance that was previously obtained
  V is the voltage at which the motor is supplied (6 V)
  I is the current that circulates through the motor

  When calculating the counter-electromotive force, the current that will be used is the one that was obtained when the boat was tested with the expected mechanical drag load: 9,6 A. Then, the CEMF will be:

  $$E_a = 6\ V - 9.6\ A \cdot 0.51\ \Omega = 1.104\ V \tag{3.5.5}$$

  The next step is to obtain the rotation speed of the motor. To achieve this, the boat will be placed in water and its current will be limited to 9.6 A by the power supply to simulate the typical conditions in which it will operate. This speed will be measured using the DT-2234B laser light digital tachometer, which is ideal since it allows us to obtain reliable measures (with a precision of ±0.05%) without the necessity of having direct contact [41]. Moreover, its compact dimensions allow us to introduce it inside the boat, whose access is quite restricted, to measure from a closer distance.

Its operation is simple, a small piece of white tape is placed on the part that rotates and when it is lit by the tachometer, the tape will reflect the light, which will be sensed back by the tachometer, allowing it to estimate the amount of revolutions per minute. The easiest part and more reliable part to place the tape would be on one of the screws, however, if the boat is in the water that part will be submerged so we will not have access to it. The second best option is to place the tape on the drive shaft, this is more difficult because the shaft is quite thin and it is not as accessible as the screws, but we managed to do it (figure 3.25). The measured rotational speed was 1831 $rpm = 191.74\ rad/s$.

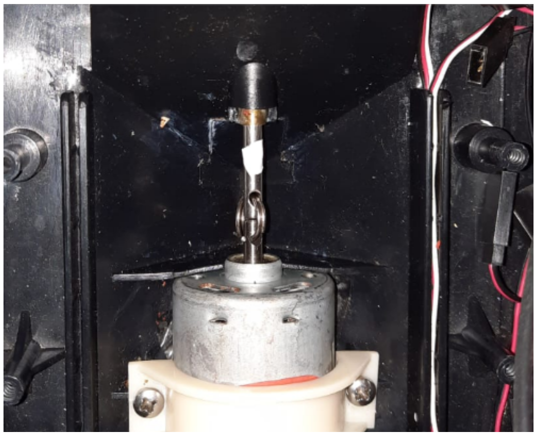


Figure 3.25 – Drive shaft with the tachometer tape

Having the CEMF and the rotational speed, the CEMF constant can be obtain as:

$$K_a = \frac{E_a}{\omega} = 5.76 \cdot 10^{-3} \frac{V}{rad \cdot s} \tag{3.5.6}$$

- Torque constant (K_t): as it was explained before, when a current is applied to the windings that are placed in a magnetic field, a torque force will be induced on the windings. The torque constant (K_t) relates the torque force and the current applied to the motor. The electrical power developed in the motor (P_m) can be calculated as:

$$P_{motor\ electrical} = E_a \cdot I \tag{3.5.7}$$

The mechanical power in the motor can be calculated as:

$$P_{motor\ mechanical} = \tau \cdot \omega \tag{3.5.8}$$

If electrical losses are neglected then we can equal the mechanical and the electrical motor power:

$$E_a \cdot I = \tau \cdot \omega \ ; \frac{E_a}{\omega} = \frac{\tau}{I} \ ; K_a = K_t \tag{3.5.9}$$

This means that the torque constant in $[\frac{Nm}{A}]$ is equal to the CEMF constant in $[\frac{V}{rad \cdot s}]$. This correspondence is only true if the constants are expressed in the SI system. Therefore, the torque constant is:

$$K_t = K_a = 5.76 \cdot 10^{-3} \frac{Nm}{A} \tag{3.5.10}$$

With the torque constant we can also calculate the motor's torque with the following equation:

$$\tau = K_a \cdot I = 55.27 \cdot 10^{-3} N \cdot m \tag{3.5.11}$$

- Mechanical time constant (t_m): when a constant voltage is applied to the motor, the initial rotational speed will be zero and then the current will be at its maximum, resulting in a high torque and acceleration. As the motor starts to turn faster, the CEMF will increase and the current will decrease, which reduces the torque and acceleration rate. The relationship between the rotational speed and the torque can be observed in (figure 3.26), where n is the rotational speed and M the torque.

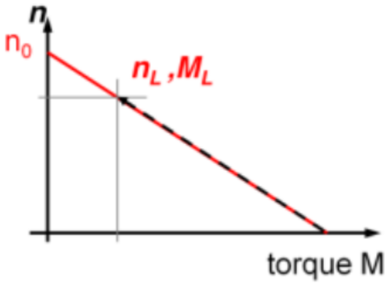
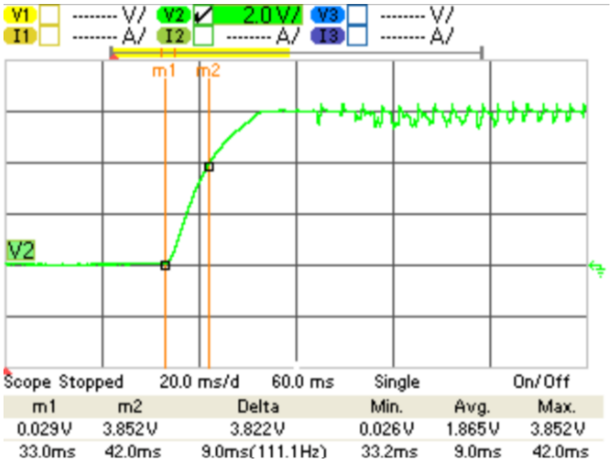


Figure 3.26 – Rotational speed (n) vs torque (M) graph [27]

The mechanical time constant represents the reaction time of the motor speed when a constant voltage is applied. Since the mathematical model of a DC motor presents a direct relationship between the armature tension and the rotational speed [19], we can measure the mechanical time constant of the motor by applying a voltage step to the motor's terminals and obtaining the time that the motor takes to reach 63.2% of the applied voltage, which is how this constant is defined. Applying a 6 V step voltage, the time that the motor takes to reach 3.79 V (63.2% of 6 V) will be 9 ms approximately as it is displayed in figure 3.27.



Scope Stopped 20.0 ms/d 60.0 ms Single On/Off

m1	m2	Delta	Min.	Avg.	Max.
0.029V	3.852V	3.822V	0.026V	1.865V	3.852V
33.0ms	42.0ms	9.0ms(111.1Hz)	33.2ms	9.0ms	42.0ms

Figure 3.27 – Transitory response of the motor to a 6 V step voltage

- Moment of inertia (J): it is a property of a solid that indicates the rotational capacity of the section with respect to an axis [1], relating the applied torque to the angular acceleration:

$$\tau = J\alpha \tag{3.5.12}$$

Where α refers to the angular acceleration.

In a DC motor,it can be calculated through a parametric method based on the already obtained values [19], resulting in the following equation:

$$J = \frac{t_m K_a K_m}{R} = 5.855 \cdot 10^{-7} Kg \cdot m^2 \tag{3.5.13}$$

- Friction torque (τ_f): Friction is a tangential reaction force that is present during the relative motion of surfaces in contact [49]. Friction torque appears due to the sliding of the commutator and the brushes, the drive shaft's friction and the mechanical drag load. As it was mentioned during the torque constant calculation, the torque can be expressed as:

$$\tau = K_t \cdot I \tag{3.5.14}$$

The friction torque is obtained with this same equation but, instead of using the stabilized current, we will use the current at which the motor starts rotating when the voltage is gradually increased from 0 V. This current is the minimum that it requires to overcome the static friction torque. In this case, the motor starts rotating at 1.5 A. Therefore, the friction torque will be:

$$\tau_f = K_t \cdot I_{static\ friction} = 8.64 \cdot 10^{-3} N \cdot m \tag{3.5.15}$$

- Coulomb friction constant (B): Coulomb friction is always present and it is dependant on the direction of motion, opposing to it, the normal force and the properties of the contact surface [48]. It will be calculated in stable conditions from the motor's mechanical model equation, which is:

$$L \cdot \frac{\delta\omega(t)}{\delta t} + B \cdot \omega(t) + \tau_f = \tau_m(t) \tag{3.5.16}$$

This is a linear 1st–order ODE with constant coefficients, however, if we consider stable conditions then the rotational speed will be constant so its derivative will be zero, resulting in the following expression:

$$B \cdot \omega + \tau_f = \tau_m; B = \frac{\tau_m - \tau_f}{\omega} \tag{3.5.17}$$

Using the previously obtained rotational speed, the Coulomb friction constant is:

$$B = 2.43 \cdot 10^{-4} N \cdot m \cdot s \tag{3.5.18}$$

All the parameters are listed in table 3.7.

Parameter	Value
Armature resistance (R)	0.51 Ω
Armature inductance (L)	185.41 µH
Counter-electromotive constant (K_a)	$5.76 \cdot 10^{-3} \frac{V}{rad \cdot s}$
Torque constant (K_a)	$5.76 \cdot 10^{-3} \frac{Nm}{A}$
Mechanical time constant (t_m)	9 ms
Moment of inertia (J)	$5.855 \cdot 10^{-7} Kg \cdot m^2$
Friction torque (τ_f)	$8.64 \cdot 10^{-3} N \cdot m$
Coulomb friction constant (B)	$2.43 \cdot 10^{-4} N \cdot m \cdot s$

Table 3.7 – DC motor model parameters

With these parameters, we can now substitute them in equations 3.5.16 and 3.5.2, obtain the required transfer functions and study our system's response.

Chapter 4

System requirements and constraints

Once that we have are familiarized with the technical features of our boat, we must identify our device's requirements. Some of these requirements will be given by our client and some of them will be defined by ourselves; based on our knowledge in the electronics field and considering the operation conditions of our device.

4.1 Functional requirements

These refers to the features must be included in our device:

- Propulsion and maneuverability system: this will be achieved through the DC motor and the servo motor that is connected to rudder. Both of these are already included in the boat, nevertheless, in order for the propulsion system to operate forward and backwards, an H bridge must be added.
- Battery current and voltage monitoring: when working with lead acid batteries, it essential to measure the voltage because if the battery is discharged below certain voltage, called "end voltage", it will lose its ability to be recharged. Moreover, by measuring both current and voltage we can estimate the state of charge and detect current peaks caused by a DC motor malfunction that could be dangerous for the circuit.
- Battery charger: the charging process of a lead acid battery is complex and has different parts, which requires a specific charger. By integrating this charger into our board it will not be necessary to remove the battery from the boat in order to charge it with an external charger.
- Reverse battery and charger polarity protection: occasionally batteries and chargers are placed with their polarity reversed due to human error, which could damage partially or totally the circuit they are connected to. To avoid these risks, a protection circuit that disconnects the battery or the charger when it is incorrectly placed must be added.
- Power switch button: this allows the user to turn on or off the boat's circuitry by pressing one button without manually connecting or disconnecting the battery, which is a tedious process.
- Orientation system: in order to calculate the route to the desired location and make corrections in the trajectory, it is necessary to know the boat's position and its orientation. These will be provided by a GPS tracker and a IMU.
- Wireless communication: this will allow the user to control the boat from distances ranging from hundreds to a few thousands of meters and to receive messages or warning from the boat. In order to

establish this communication a GSM module will be included in the boat and an mobile app will be provided to the user.

- Navigation lights: this boat must fulfill the visibility specifications described in the Convention on the International Regulations for Preventing Collisions at Sea (COLREGs). According to rule 23, section C part II: “a power-driven vessel of less than 7 meters in length whose maximum speed does not exceed 7 knots may in lieu of the lights prescribed in paragraph (a) of this Rule exhibit an all-round white light and shall, if practicable, also exhibit sidelights”. An all-round light is a white navigation light that indicates the position and can be seen over an arc of 360º and the sidelights area green light on the starboard side and a red light on the port side [11].

4.2 Performance requirements

These requirements refer to the device’s operation and will translate into firmware and schematics design considerations:

- Battery duration and consumption: since the boat is supplied from a lead acid battery and it has no access to any other power sources, it is mandatory to minimize the power consumption so the battery duration is enough to reach the desired position and to stay there during the immersion.
- Waterproofing: this is an indispensable attribute since our device will be used to sail on water, which can cause short circuits and permanently damage the electrical components. Moreover, it causes corrosion, leading to malfunctioning and lifespan shortening.
- External conditions flexibility: when operating outdoors, specially at sea, meteorological conditions must be taken into consideration. External conditions such as wind or waves can alter the boat’s trajectory, complicate its advance or even capsize it. For these reasons, it is important to include a feedback loop that allows trajectory and position correction and a IMU that detects if the boat has been capsized.
- Easiness: the operation of the boat must be simple and easy to use, as well as the app’s human machine interface.

4.3 Design requirements

These requirements will be mainly related to the circuit and PCB design:

- Microcontroller: it is essential to control, coordinate and interface with all the components and modules of our circuit.
- Debugging: since this will not be the final version of the product but a prototype, we must include some debugging elements that facilitate the testing process.
- Voltage regulators: given the amount of features that our boat will have, a big variety of components and modules will be required. Each component has a different operating voltage range and it may differ from the battery voltage, so voltage regulators will be used.
- Size and shape: the PCB size and shape must adapt to the inside of the boat, including its internal components such as the battery, the cables and the servomotor. Moreover, it must be fixed and stable so it does not move nor collide with other components when the boat moves brusquely.
- Corrosion-resistant: as we mentioned before, corrosion can have devastating effects on the circuit’s performance. Every PCB is vulnerable to corrosion, but in this case our device will operate in humid

environments, thus the corrosion possibility is even higher. For this reason, we will use tented vias, which reduces their exposure to air.

- Optimized component positioning: this is an essential requirement in every PCB, but in this case we must be careful because the user must be able to manipulate and see different parts of the circuit without extracting it from the boat nor being blocked by other internal elements. Moreover, we must ensure that the parts of the circuit that are connected to other internal components of the boat are as close as possible to them and that the cables are not bent.

- Test points: we will be included some pads that enables us to easily check the voltage at some of the most crucial parts of the circuit.

- LEDs: just like the test points, some LEDs must be added to indicate if certain parts of the circuit are correctly supplied. Moreover, some of them will be used to make visual indications related to the program upload and flow.

- Power dissipation: as we have checked in Chapter 3, the DC motor has a high power consumption in the order of 60 W. It is important to take this into consideration when designing the circuit, choosing components and sizing the PCB tracks and vias, if not, the board will overheat and eventually it could burn.

- Board information and logo: as in the majority of Granasat's projects, the PCB will be manufactured by JLCPCB, whose minimum order is five boards. In each of this board, will mount different parts of the circuit so we can test them separately. Therefore, it is useful to leave a white box where we can write titles to differentiate them. Adding the logo is not a priority but it will be include as long as there is enough space.

4.4 Firmware requirements

They are strongly related to the MCU model features, but in general terms, they are:

- Program size: in MCUs, firmware is stored in one of the internal memories, typically in an EPROM or a flash memory. The size of these memories usually ranges from a KB to a few MB, which sometimes can be limiting. Therefore, it is important to pay attention to the maximum program size of our board and try to optimize it as much as possible so there is room for new modifications and additions.

- Debugging: the debugging elements that were included will translate to additional firmware features that will allow us to visualize and modify the program state and visualize sensor's data.

- Computational load: it refers to the amount of operations that the MCU performs. If it is overloaded its response will be slower and its power consumption will increase, thus, it must be reduced to the extend possible.

4.5 Constraints

Whenever a real project is carried out, not every idea nor requirement can always be accomplish due to constrains of all kinds. These are the initial constraints that we have considered, however, it is likely that other constraints can appear along the process:

- Deadline: this is the most obvious constraint and, just like in the job market, the project deadline is not determined by ourselves. As we mentioned in section 1.2, when organizing the project we must divide it in smaller tasks and consider which are the most important ones and discard the less critical ones if time is running low.

- Price: when designing a product, the price must be one of the most important considerations. When choosing components and the different design options we have to find a balance between price and functionalities and, when it is possible, use the already-available components in our lab.
- Components availability: due to the global chip shortage that is present since 2020, a noticeable percentage of the components production has been stopped for months and, in some cases, indefinitely. This will complicate the component selection and force us to search for different options.
- Components mounting type: our laboratory's equipment, although varied and powerful, does not provide the means to solder components whose packaging is too small or it is less common, such as BGA or PGA. This restricts a significantly large percentage of components.

Chapter 5

System description and design

5.2 Electronics design

As it was mentioned in Chapter 1, this thesis is the continuation of a previous one. In that previous thesis, the author provided an electronic design with its associated PCB supposedly ready to manufacture and to be programmed. Nevertheless, that design was developed under serious time constraints, which could have led to mistakes or inconsistencies. Moreover, it would not be prudent to manufacture the circuit without having previously analyzed it and verifying it. For this reason, in this section we will go through each part of the circuit, study them, make corrections, suggest possible design alternatives and, in some cases, completely redesign them.

When working with circuits that are too large to be displayed and analyzed as a whole, the design process is split into different sections, which will be the case in this project. In this section, each part of the circuit will be analyzed and redesigned.

5.2.1 Microcontroller

First of all, we need some kind of data processing system that controls the circuit's operation. In embedded systems, this is typically achieved through a Microcontroller Unit (MCU), which contains internal memories and peripheral interfaces, as opposed to microprocessors, whose memories and peripheral interfaces are implemented with external chips.

There is a wide variety of MCUs, nevertheless, we have focused on the most relevant low-cost ones among the teaching community to simplify the learning process since we are already familiarized with them: the ATmega328p, present in some Arduino boards, the ESP8266EX present in WeMos WiFi boards and the ESP32-WROOM-32D, which is the successor to the ESP8266 and has additional and improved features. Of all of them, the ESP32-WROOM-32D was chosen last year as it presents more and superior characteristics, listed in table 5.1. However, we have noticed that this board is being discontinued and not recommended for new design so we will use the manufacturer's recommended replacement, the ESP32-WROOM-32E-N4.

Name	ESP32-WROOM-32D-N4 Datasheet	ESP32-WROOM-32E-N4 Datasheet
CPU	Xtensa dual-core 32-bit LX6	Xtensa dual-core 32-bit LX6
SRAM	520 KB	520 KB
ROM	448 KB	448 KB
Flash	4 MB	4 MB
GPIOs	38	38
ADC pins	18 (12 bits)	18 (12 bits)
DAC pins	2 (8 bits)	2 (8 bits)
Clock frequency	80 to 240 Hz	80 to 240 Hz
Wi-Fi	Yes	Yes
Bluetooth	Yes	Yes
Interfaces	UART, I2C, I2S, SPI	UART, I2C, I2S, SPI
Supply voltage	3 to 3.6 V	3 to 3.6 V
Supply current	Typical: 20 to 68 mA Maximum: 500 mA	Typical: 20 to 68 mA Maximum: 500 mA
Dimensions	18 × 25.5 × 3.1 mm^3	18 × 25.5 × 3.1 mm^3
Price	3.77 € (DigiKey)	2.82 € (Mouser)
Choice	✗	✓

Table 5.1 – ESP32 trade-off table

Even though the ESP32-WROOM-32D is not recommended for new designs it will be used for this prototype since it was already available in our laboratory, but it will be replaced in the newer versions. Both models have almost the same characteristics and their footprint and GPIO distribution is the same.

In particular, the ESP32-WROOM-32UE will be used since it contains an on-board antenna for Wi-Fi and Bluetooth, which are not required in this prototype but they could be useful if we wanted to add additional features.

This board has 38 GPIOs, of which six of them are not recommended to use since they are connected to the internal flash memory where the programs are uploaded, these are GPIOs 6 to 11. However, their behaviour has been analyzed and they showed that their voltage toggles from high to low during the program upload but, after that, the user can program them as regular outputs (figure 5.1). Since they are not as reliable as the other pins, they will only be used if the rest are busy and mainly for the LEDs.

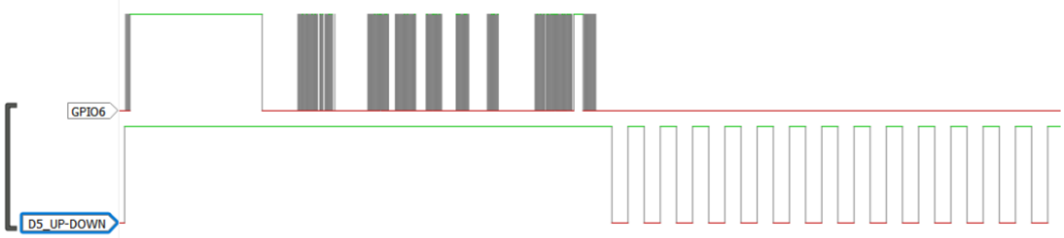


Figure 5.1 – GPIO6's behaviour during the upload and after it with a blink test [25]

This was not taken into consideration in the previous thesis and these pins were connected as if they were normal ones. For this reason, all the pin assignations will be completely rearranged.

The only external components that are recommended for the ESP32 are two capacitors: a 100 nF one for decoupling and a 10 μF one to avoid power rail collapses when the board is operating in transmission mode. The circuit can be consulted in sheet 3 of Appendix D.

The total cost of this circuit is calculated in table 5.2:

Component	**Price per unit [€]**
ESP32-WROOM-32D x1	3.77 (DigiKey)
10 μF 0603 capacitor x1	0.20 (Mouser)
100 nF 0603 capacitor x1	0.09 (Mouser)
Total cost [€]	4.06

Table 5.2 – ESP32 circuit cost

5.2.2 GPS tracker

This device is required to provide the boat's coordinates in real-time, essential to calculate and make corrections in the boat's trajectory. The most commonly used GPS module is the GY-NEO6MV2 module due to its low price and small size. There are smaller modules, such as the Quectel L86 GPS module, but its price is much higher and it offers the same functionality. For that reason, we will choose the GY-NEO6MV2 module, which was the one chosen in the previous thesis.

Name	GY-NEO6MV2 module IC's datasheet	Quectel L86 GPS module Datasheet
Supply voltage	3.3 to 6 V	3 to 4.3 V
Current	Typical: 51.85 mA Maximum: 74.75 mA	Typical: 26 mA Maximum: 26 mA
Interface	UART	UART
Sensitivity	162 dBm	165 dBm
Dimensions	36 x 26 x 4 mm^3 but it does not include an antenna	18.4 x 18.4 x 6.45 mm^3 with an integrated ceramic antenna
Price	2.26 € (AliExpress)	7.49 € (AliExpress)
Choice	✓	✗

Table 5.3 – GPS modules trade-off table

After studying the previous thesis, we have noticed that some of the GY-NEO6MV2 module's characteristics were mistaken for the NEO-6Q/M's. Due to this confusion, the supply voltage and current consumption specified were wrong. To clarify things, GY-NEO6MV2 module is the one that will be implemented in our circuit and it includes external elements such as a 3.3 V voltage regulator, an EEPROM, a red SMD LED that toggles when it locks on to GPS satellites, a 3 V Li-ion button battery and an antenna connector, whereas the NEO-6Q/M is the GPS receiver that is contained in the GY-NEO6MV2 module.

When it comes to the currents, in the NEO-6Q/M datasheet it is specified that the average current for complete data acquisition is 47 mA and the maximum supply current is 67 mA. To that, it must be added the current consumed by the 24AA32A EEPROM and the charging current of the Li-ion battery.

Regarding the charging current of the Li-ion battery, the diode is fully unknown and it is connected to a 100 Ω resistor (not the same as the schematic). Knowing that the voltage of the battery is 3 V and it is connected to 3.3 V through a diode, it can be assumed that the forward voltage of the diode will not be higher than 0.2 V, then the charging current is approximately:

$$I_{bat} = \frac{3.3\ V - 0.2\ V - 3\ V}{100\ \Omega} = 1\ mA \tag{5.2.1}$$

Lastly, in the 24AA32A EEPROM datasheet it is specified that the typical current during writing is 0.1 mA and the maximum is 3 mA. A summary of the current consumption of the GY-NEO6MV2 module can be observed in table 5.4:

GY-NEO6MV2 module's component	Current consumption
NEO-6Q/M	Average: 47 mA Maximum: 67 mA
Li-ion battery charger	1 mA
24AA32A EEPROM	Typical: 0.1 mA Maximum: 3 mA
Total Consumption	Typical: 48.1 mA Maximum: 71 mA

Table 5.4 – Current consumption of the GY-NEO6MV2 module

Observing the module, it has been determined that it uses a XC6222B331MR-G 300mA high speed LDO voltage regulator. There are different models of this regulator depending on the use of the CE pin, nevertheless, in all of them the required dropout voltage can be calculated as:

$$V_{dropout} = I_{out} \cdot (V_{out} + 1\ V) \tag{5.2.2}$$

In the worst-case scenario, when the supply current is maximum, the voltage dropout is:

$$V_{dropout} = 74.75\ mA \cdot (3.3\ V + 1\ V) = 0.305\ V \tag{5.2.3}$$

Thus, a 3.6 to 6 V input voltage range is required for the GY-NEO6MV2 to work properly in the case where the current consumption is maximum and, since this regulator is a LDO regulator, its input current will be almost the same as the output one. This was not taken into consideration in the previous thesis, so this module was supplied at 3.3 V, which should be enough for it to work under normal conditions. However, by supplying this module from the 4 V voltage regulator that was already included in the previous thesis' circuit (see section 5.2.13), its performance will be more reliable. The modified circuit is displayed in sheet 4 of Appendix D.

Regarding the antenna, the one that came with the module will be used, which is a ceramic patch antenna. These types of antennas are often used in GPS systems due to their high gain, reduced size and low price [46]. Its exact characteristics are unknown because the provided antenna's model is not specified. However, observing the NEO-6Q/M datasheet, we can obtain the antenna recommendations (table 5.5), which should be fitting for the antenna that comes with the module.

Type	It can be passive or active
Gain	15 to 50 dB
Maximum noise figure	1.5 dB

Table 5.5 – NEO-6Q/M's antenna specifications

5.2.3 Wireless communication

In order to establish wireless communication between the user and the boat, a SIM800L GSM/GPRS module was selected in the previous theses, whose technical specifications are listed in table 5.6.

Name	SIM800L GSM/GPRS Module Datasheet
Features	Quad-band GSP, GPRS, SMS, phone calls, antenna, SIM socket
Voltage supply range	3.4 to 4.4 V
Current consumption	Sleep mode : 0.7 mA Maximum peak current : 2 A
Dimensions	25 x 25 x 8 mm^3
Interface	UART
Price	1.93 € (AliExpress)

Table 5.6 – SIM800L module specifications

The reasons for using this module are its low price, small size, the great amount of information and documentation available since it is one of the most widely used modules and, the most important factor, it was already available in our laboratory. Its main downside in our project is that a 4 V regulator is required because it cannot be supplied at 3.3 V. Moreover, it is obvious that this module is quite outdated since it only supports 2G networks. This could cause some serious problems in the future as 2G/3G networks are slowly being shut down in some countries and will eventually disappear [23]. For that reason it is a good idea to consider newer options that are 4G compatible. In the trade-off table 5.7 two alternatives are explored:

Name	SIM7600E LTE CAT1 Module Description	SIM7070 Module Description
Features	LTE-FDD, LTE-TDD, HSPA+, WCDMA, EDGE, GPRS, GSM, 2 antennas, SIM socket	NB-IoT, GPRS, antenna, SIM socket
Voltage supply range	4 to 8 V	3.2 to 4.2 V
Current consumption	Typical: 4.6 to 650 mA Maximum peak current: 2 A	Sleep: 0.4 mA Idle: 5.6mA Maximum peak current: 2 A
Dimensions	40 x 40 x 8 mm^3	35 x 28.4 x 8 mm^3
Interface	UART, USB2.0	UART, USB, I2C, SPI
Price	1.64 € (AliExpress)	4,19 € (AliExpress)
Choice	✓	✗

Table 5.7 – SIM800L alternatives

The SIM7600E LTE CAT1 Module has some inconveniences such as its higher size or the necessity of a 4 V regulator, but it allows 2G/3G/4G connectivity and it is more affordable. For these reasons, it is suggested for the following prototypes in case 2G networks are not available.

Regarding the SIM800L module schematic design, the proposed circuit in the previous thesis is shown in figures 5.2 and 5.3. This device was supplied from the 4 V regulator and used three bidirectional level shifters, which allow to adapt a voltage level to another one and vice versa, in this case to adapt the TX, RX and RST signals from the ESP32's voltage (3.3 V) to the SIM800L supply voltage (4 V) and vice versa.

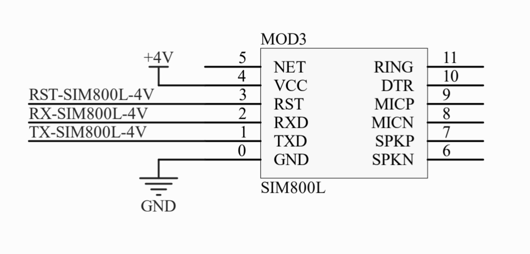


Figure 5.2 – SIM800L connections

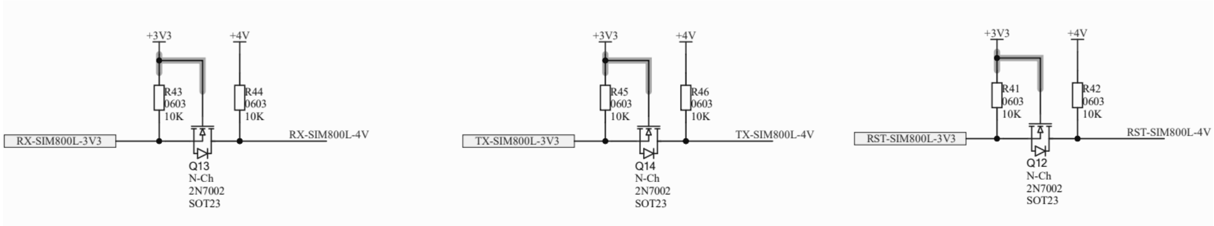


Figure 5.3 – SIM800L level shifters

This design is correct but it can be optimized by removing the level shifters as the SIM800L logic level ranges from 3 to 5 V [21]. Since the source for this information is not an official datasheet, it has been experimentally tested and we can confirm that it works fine. The money that will be saved with this modification is displayed in table 5.8.

Component	**Price per unit [€]**
2N7002 x3	0.34 € (Mouser)
10 kΩ 0603 resistor x6	0,09 € (Mouser)
Total saved money [€]	1.56

Table 5.8 – SIM800L circuit saved money

The typical power consumption, which is required for the power budget, is not specified in the datasheet as it depends on the module's operation and features that are being used. Because of that, it has been experimentally measured with the N6705A DC power analyzer while emulating the typical operation of the module in this project, obtaining the results shown in figure 5.4. In view of this experimental measurement, we can estimate the average current as the consumption when it is operating in the HTTP services, approximately 125 mA and the maximum peak current as 2 A approximately.

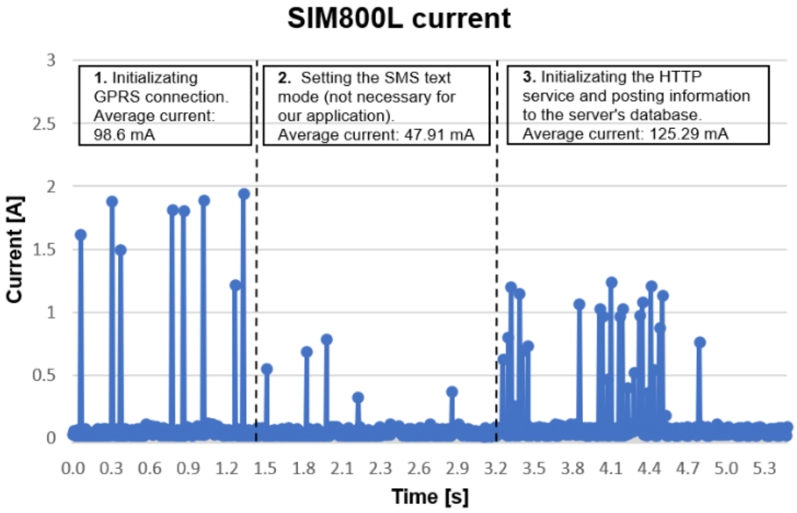


Figure 5.4 – SIM800L current consumption

5.2.4 Inertial Measurement Unit (IMU)

It will be required to estimate the boat's orientation, mainly working as a digital compass. This, combined with the GPS coordinates, will allow us to develop an algorithm that guides the boat to a desired location. Moreover, an IMU will also provide the angular velocity and the acceleration, and in some cases, some additional data such as the temperature or the altitude. This could be useful to detect if the boat has capsized or has got stuck in an obstacle.

In the previous thesis, the Adafruit 10-DOF IMU module was chosen because it was already available in our laboratory, but not only that, it is small, easy to interface as all their sensors can communicate through I2C and easy to program since there is a great amount of information about this module and Adafruit provides official libraries that abstract the programmer from the low-level operation of each sensor. This module is composed of three different sensors:

- L3GD20H: it is a MEMS gyroscope that provides angular velocity in three axes. This velocity will be given in degrees per second (dps) and, by default, its full scale is ±245 dps, although it can be changed to ±500 or ±2000 dps.

- LSM303: it is a MEMS 3-axis magnetometer and accelerometer. The acceleration will be provided in g (9.81 m/s^2) and, by default, its full scale will be ±2g, although it can be increased up to ±16g. Regarding the magnetometer, the magnetic field will be provided in Gauss and, by default, its full scale will be ±1.3, but can be increased up to ±8.1 g.

- BMP180: it is a digital barometric pressure and temperature piezo-resistive sensor. It provides the barometric pressure within a 300 to 1100 hPa range in 1 Pa steps and provides temperature within a -40 to 85 ºC range in 0.1ºC steps.

The technical specifications of this module are listed in table 5.9. Despite of the positive features of this module, it has some major inconveniences: It is quite expensive ($29.95), includes functionalities that are not required for our application and the most important one, its production is discontinued. For this prototype, this module will be still used because it is available in our laboratory but other options should be considered to replace it in newer prototypes. Two of this alternatives are compared in table 5.10.

Name	Adafruit 10-DOF module Datasheets
Features	Three interrupt pins, two "ready" pins, magnetometer, accelerometer, gyroscope, thermometer, barometric pressure sensor
Voltage supply range	3 to 5 V
Current consumption	Typical: 6.21 mA Maximum: 7.21 mA
Dimensions	38 x 23 x 3 mm^3
Interface	I2C
Price	$29.95 (Adafruit)

Table 5.9 – Adafruit 10-DOF technical specifications. Note: the current has been estimated by adding the current consumption of each sensor.

Name	BNO085 Datasheet	BMM150 Datasheet
Features	Magnetometer, accelerometer, gyroscope, thermometer, 32-bit ARM® Cortex™-M0+ microcontroller	Magnetometer
Voltage supply range	3 to 5 V	1.62 to 3.3 V
Current consumption	Maximum current: 7.5 mA	Low power preset: 170 μA Regular preset: 0.5 mA Enhanced preset: 0.8 mA
Dimensions	3.8 x 5.2 x 1.1 mm^3	1.56 × 1.56 × 0.6 mm^3
Interface	I2C/UART/SPI/UART-RVC	SPI/I2C
Price	15.96 € (DigiKey)	1,96 € (Mouser)
Choice	✓	✗

Table 5.10 – Adafruit 10-DOF module alternatives

The BMM150 was proposed because the main purpose of the IMU was to estimate the yaw position of the boat. This device is a three-axis magnetic field sensor so it would not provide the rest of the unnecessary measurements, saving money, power and space in the PCB. If this option was chosen, we could detect if the boat had got stuck in an obstacle or has been removed from the water with a current monitor. As it was mentioned in 3.5, the motor's current will greatly increase when the boat's movement is blocked and will decrease as the mechanical drag load does. Then, by detecting a current higher than 10 A we can assume that the boat is stuck and detecting a current lower than 5 A we can safely assume that the boat is out of the water. The problem is that we could not confidently detect if the boat had capsized. For this situation we need a 9-Axis IMU such as the BNO055.

The BNO085 has been proposed because it contains a magnetometer, accelerometer, gyroscope, thermometer and a 32-bit ARM® Cortex™-M0+ Microcontroller all in the same package. This is optimal since the included MCU contains by default a fusion algorithm that operates with the sensors' data and provides the absolute orientation, which facilitates the firmware development and releases the ESP32 from that computational load. Moreover, the fact that all three sensors and MCU are included in the same compact package increases the available space on the PCB. The main downside of this device is its price, which is significantly higher than the BMM150 sensor, but if it is compared with the Adafruit 10-DOF module it is cheaper. For these reasons, the BNO085 has been selected for next prototypes.

The BNO085 circuit that we propose is displayed in figure 5.5. It communicates with the MCU via I2C and its total cost can be observed in table 5.11, which is still much lower than the Adafruit 10-DOF module.

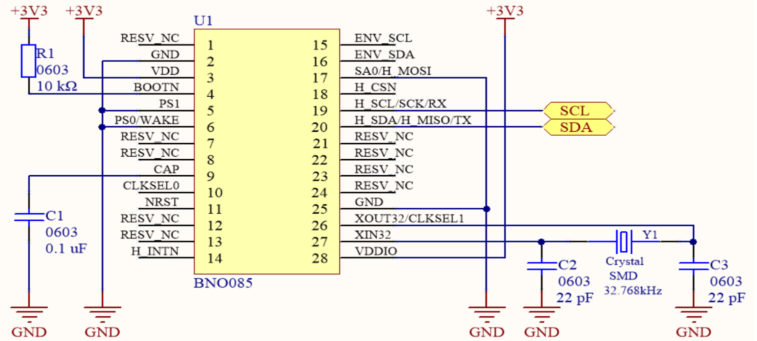


Figure 5.5 – BNO085 circuit

Component	Price per unit [€]
BNO085 x1	15.96 (DigiKey)
10 kΩ 0603 resistor x1	0.09 (Mouser)
0.1 μF x1	0.09 (Mouser)
22 pF x2	0.09 (Mouser)
32.768kHz Crystal x1	0.69 (Mouser)
Total cost [€]	17.01

Table 5.11 – BNO085 circuit cost

Regarding the previous year circuit that we are using for this prototype, based on the Adafruit 10-DOF. It is correct, however, the interrupt pins have been removed since they are not strictly necessary and we are short on ESP32 pins. The modified circuit is shown in sheet 6 of Appendix D.

5.2.5 Servo motor

As we mentioned in section 3.4, one of the servo motors is connected to the boat's rudder and will be used to control the direction. This motor has only three ports: one for the power supply, one for ground and one for the control voltage pulses. In the previous thesis, the servo motor was not tested experimentally so it was assumed that its supply voltage had to be 5 V obligatorily, so a 5 V XL1509 voltage regulator was implemented (the voltage regulators will be explained in detail in section 5.2.13). However, that is not correct, so this regulator will be removed from our design and the servo will be supplied at 6 V.

Moreover, a 3.3 V to 5 V bidirectional level shifter like the ones in figure 5 .3 was used to control the motor, but since we experimentally checked that the motor works correctly at a 3.3 V logic level, the level shifter will be removed. The money that will be saved with this modifications is displayed in table 5.12 and the servo motor circuit can be consulted in sheet 7 of Appendix D.

5

Component	Price per unit [€]
2N7002 x1	0.34 (Mouser)
10 kΩ 0603 resistor x2	0.09 (Mouser)
XL1509 module x1	1.32 (AliExpress)
Total saved money [€]	1.84

Table 5.12 – Servo motor circuit saved money

5.2.6 LEDs and buttons

In last year thesis' design, most of the LEDs were supplied from the 5 V voltage regulator. Nevertheless, this regulator has been removed since it is not necessary for the motor. Thus, all the resistor values in series with the LEDs will be recalculated.

5.2.6.1 5mm LEDs

As it was mentioned in the functional requirements (section 4.1), we need an all-round white light, a green light on the starboard side and a red light on the port side. This boat already contains 5mm LEDs that can be used as sidelights, but none of its LEDs can serve as an all-round light since they cannot provide a 360º visibility to other ships. For this reason, a hole should be made on the top part of the boat that allows the insertion of a 5 mm white LED. These LEDs were supplied from the 5 V regulator so the circuit will be completely redesigned.

One problem that could be noticed in last thesis' design is the fact that all the LED's in series resistors had the same arbitrary value. It is important to notice that depending on the color and the model, LEDs have a different forward voltage and luminosity index curve, thus, if the resistors are the same, the intensity of their brightness will probably differ between them and will not be the highest possible.

To avoid this, the forward current vs relative luminosity graph must be consulted in the LEDs' datasheet. Nevertheless, the exact LED models that are used in the boat are unknown. We have tried using other 5 mm LEDs' datasheets as reference but after looking at different models, we have concluded that the required current for the same relative intensity as well as the forward voltage widely varies between different models [29]. For this reason, the forward voltage of each LED will be experimentally measured and different resistors will be tested until achieving an uniform brightness.

In section 3.2 it was noticed that the green and red lights were broken so they have been replaced by new high intensity LEDs whose model is unknown since they were already in the laboratory. It is important to use high intensity lights since their purpose is to notify the presence of this boat to other vessels from the longest distance possible.

When calculating the LED resistors, our goal is to get the highest brightness using the smallest current possible. To achieve this experimentally, the forward voltage will be measured and then, the current on the LED will be increased until the maximum brightness is reached, this is the current that will be used to calculate the resistor value. Table 5.13 shows the results for each color:

LED color	Forward voltage [V]	Minimum current [mA] for the highest brightness	Resistor [Ω] to achieve the highest brightness	Closer available resistor [Ω] in our laboratory	Actual current [mA]
Red	2.04	12.70	311.81	300	13.20
Green	2.60	3.60	944.44	820	4.14
White	2.84	6.80	464.70	470	6.72

Table 5.13 – Optimal LED's resistors calculation

Figure 5.6 shows the experimental test with the new resistors' values, achieving their highest brightness possible and uniformity:

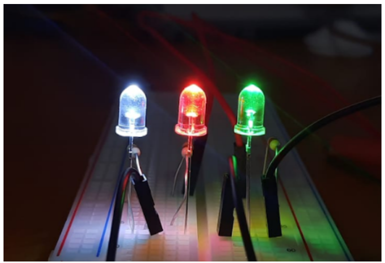


Figure 5.6 – Experimental test of the 5mm LEDs' brightness

The modified circuit is available in sheet 4 of Appendix D.

Component	**Price per unit [€]**
300 Ω 0603 resistor x1	0.14 (Mouser)
820 Ω 0603 resistor x1	0.09 (Mouser)
470 Ω 0603 resistor x1	0.10 (Mouser)
2.54 mm female 2 pin header x3	0.39 (Pololu)
Total saved money [€]	1.50

Table 5.14 – 5mm LEDs circuit cost

5.2.6.2 SMD LEDs

Apart from the navigation LEDs of the boat's case, the circuit will contain smaller SMD LEDs that will serve different functions: some of them will indicate whether some of the parts of the circuit are correctly supplied or not and some others will be used to make visual indications related to the program upload and flow.

Blue LEDs will be included in the most crucial power supply ports of the circuit, which are: "+Vin 6V", "+4V", "3V3" and the "ON/OFF" port (see Appendix D.)

Model	Price	Dimensions
VLMx1300 Datasheet	0.42 € (Mouser)	1.55 x 0.85 x 0.45 mm^3

Color	Current for a 0.5 luminous intensity [mA]	Forward voltage for a 0.5 luminous intensity [V]
Red	10	1.8
Orange	10	1.95
Blue/Green	10	3.25

Table 5.15 – VLMx1300 specifications and brightness characteristics

In view of table 5.15, we can redesign the circuit to optimize it:

- The forward voltage of the blue LED almost matches the 3.3 V power source, if we adjust the 3.3 V voltage regulator (see section 5.2.13) to obtain 3.25 V, then we can remove the in series resistor and the LED will not be over-driven. This saves two resistors, however, these resistor's pads will be left in the PCB in case the regulator could not be adjusted to perfectly match the LEDs' forward voltage.

- Concerning the 4 V power source, the resistor that must be placed in series with a blue LED to achieve 10 mA can be calculated as:

$$R_{blue\ LED} = \frac{4\ V - 3.25\ V}{10\ mA} = 75\ \Omega \tag{5.2.4}$$

- As for the 6 V power source, the resistor that must be placed in series with a blue LED to achieve 10 mA can be calculated as:

$$R_{blue\ LED} = \frac{6\ V - 3.25\ V}{10\ mA} = 275\ \Omega \tag{5.2.5}$$

In relation to the LEDs used to make visual indications related to the program, the circuit that was proposed last year is displayed in figure 5.7.

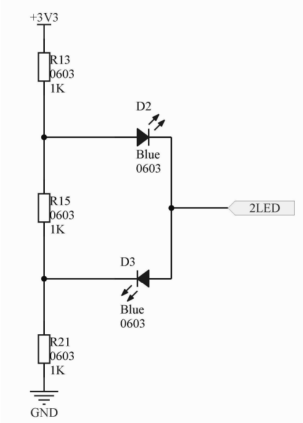


Figure 5.7 – Previous thesis' design of the program indication LEDs

In this circuit it was intended that when the “2LED” port was pulled-up, the bottom LED would turn on and the upper one would be off, if the “2LED” port was pulled-down, the bottom LED would be off and the upper one would be on and if the “2LED” pin was set to an intermediate value both LEDs would be off. In order to achieve an intermediate voltage value between 0 and 3.3 V with the ESP32, we must connect the "2LED" port to either GPIO25 or 26 because they are the only ones that feature DACs. However, since we are short on pins we cannot afford to use one of those just for the indication LEDs, for this reason they will be connected to GPIO6.

The advantage of connecting it to this GPIO is that, since it is connected to the internal flash memory where the programs are stored, it will make both LEDs automatically toggle while the program is being uploaded. We can deduce this from figure 5.1. Once the program has being uploaded, we have full control of that GPIO so we can toggle these LEDs to make other indications about the program state. The only downfall is that this pin does not have DAC so both LEDs cannot be turned off at the same time. Nevertheless, this is not a big problem because the consumption of a LED is very small, around 10 mA.

The last thesis' design (figure 5.7) has some problems, such as using resistor values that are too large and using a resistor in the middle part (R15) that it not necessary. This circuit can be optimized by removing resistor R15 and removing R13 and R21 because the forward voltage of the blue/green LEDs matches the ESP32's high state voltage. Moreover, instead of using two blue LEDs, we will use blue and green. The modified design is shown in sheet 8 of Appendix D.

With all these modifications we will save 5 resistors, in table 5.16 the saved money will be calculated.

Component	**Price per unit [€]**
1 kΩ 0603 resistor x3	0.09 (Mouser)
470 Ω 0603 resistor x2	0.09 (Mouser)
Total saved money [€]	0.45

Table 5.16 – SMD LEDs modifications saved money

The total cost of the circuit is calculated in table 5.17:

Component	**Price per unit [€]**
75 Ω 0603 resistor x1	0.09 (Mouser)
275 Ω 0603 resistor x1	0.09 (Mouser)
VLMx1300 LED x6	0.42 (Mouser)
Total cost [€]	2.70

Table 5.17 – SMDLEDs circuit cost

5.2.6.3 High power LED

It is placed in the bow and its characteristic curve was obtained in section 3.2, from it we know that we must design the circuit so the forward current is 35 mA and the forward voltage is 4.5 V. This LED was not used in the previous thesis so we have to design its circuit from scratch. Since we are supplying it from the 6 V source, the resistor value can be calculated as:

$$R_{high\ power\ LED} = \frac{6\ V - 4.5\ V}{35\ mA} = 42.9\ \Omega \qquad (5.2.6)$$

The closer value that is available in our laboratory is 50 Ω. In order to control this LED, a N-Channel MOSFET will be used. The model will be the 2N7002 since it is available in our laboratory and fulfills our requirements (table 5.18). This transistor's gate will be connected to GPIO9, which is connected to the ESP32's internal flash memory, so the LED will blink while the program is being uploaded, however, that is not a major problem. This circuit is shown in sheet 8 of Appendix D.

Name	2N7002 Datasheet
Type	N-Channel MOSFET
Maximum continuous drain current	115 mA
Maximum $\mathrm{R_{DS(on)}}$ at $V_{\mathrm{SG}} = 5$ V	7.5 Ω
$\mathrm{V_{GS(th)}}$	1 to 2.5 V
Package	SOT-23-3
Price	0.60 € (Mouser)

Table 5.18 – 2N7002 specifications

Component	Price per unit [€]
2N7002 x1	0.34 (Mouser)
50 Ω 0603 resistor x1	0.36 (Mouser)
2.54 mm female 2 pin header x1	0.39 (Pololu)
Total cost [€]	1.09

Table 5.19 – High power LED circuit cost

5.2.6.4 General purpose buttons

We will include two additional buttons that can be useful for the debugging. Their functionality is not defined yet, instead it will will be defined depending on our necessities when testing different parts. In order to save GPIOs, they were designed so they can be detected by one pin (figure 5.8).

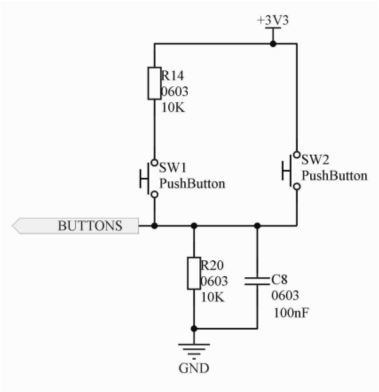


Figure 5.8 – Previous thesis' double button circuit

In this circuit, when none of the buttons are pressed, the "BUTTONS" port's voltage will be 0 V because of the R20 pull-down resistor. If the SW2 button is pressed, the "BUTTONS" port's voltage will be 3.3 V and when button SW1 is pressed, the voltage at the "BUTTONS" port can be calculated as the following voltage divider:

$$V_{buttons\ port} = \frac{3.3\ V \cdot R20}{R14 + R20} = 1.65\ V \tag{5.2.7}$$

Notice that if both buttons are pressed at the same time, the voltage at the "BUTTONS" port will be 3.3 V. In addition, just as the rest of the buttons of this project, a 100 nF capacitor has been placed in parallel with the pull-down resistor to avoid the bouncing effect. This design is completely fine and cannot be optimized so it will not be modified. Its cost is estimated in table 5.20.

Component	**Price per unit [€]**
10 kΩ 0603 resistor x2	0.09 (Mouser)
100 nF 0603 capacitor x1	0.09 (Mouser)
Push button x2	0.19 (Mouser)
Total cost [€]	0.65

Table 5.20 – Buttons circuit cost

5.2.7 Screen

The purpose of the screen is to display information during the debugging and configuration process of the boat. While the boat is being debugged this module is not essential as the board contains an USB port that allows it to display information in a computer. However, by integrating a display on the board it makes this process more comfortable and flexible since it would not require cables nor a computer to have access to the information. Moreover, it will be used as a menu where the user can configure different operation options.

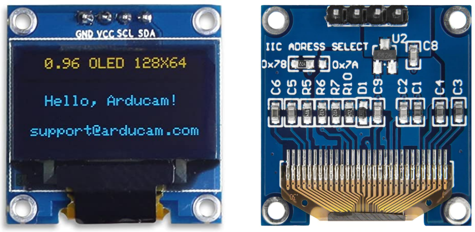
In the previous thesis, three different screen models were compared: the Nokia 5110 LCD, 16×2 character LCD and the OLED SSD1306. In table 5.21 the specifications of these modules will be shown and some new options will be added:

Name	Nokia 5110 LCD Datasheet	16x2 character LCD Datasheet	OLED SSD1306 Datasheet	OLED SSD1331 Datasheet	ILI9341 Datasheet
Supply voltage range	2.7 to 3.3 V	3.1 to 3.5 V	1.8 to 6 V	2.4 to 3.5 V	2.5 to 3.3 V
Maximum current	80 mA	2.5 mA	150 μA	500 μA	80 mA
Resolution	84 x 84 1.5”	128 x 10 2.6”	128 x 64 0.96”	96 x 64 0.95 ”	240 x 160 2.8”
Colors	Monochrome	Monochrome	Yellow and blue	65k colors	262k colors
Touch screen	No	No	No	No	Yes
Interface	SPI	I2C	I2C	SPI	8-bit/SPI
Dimensions	43.25 x 43.25 mm^2	80 x 36 x 13.5 mm^3	27 x 24.7 x 1.85 mm^3	30.7 x 27.3 x 11.3 mm^3	56 x 35 x 1.41 mm^3
Price	1.89 € (AliExpress)	0.81 € (AliExpress)	1.85 € (AliExpress)	6.56 € (AliExpress)	$ 29.95 (Adafruit)
Choice	✗	✗	✓	✗	✗

Table 5.21 – Screen modules trade-off table

The main factors when choosing the most appropriate module are the size, it must be as small as possible to fit into the PCB; the interface, ideally it should be I2C because it uses less pins; the price, it should be low; and the resolution, which should be as high as possible. The module that has the most appropriate and balanced qualities for our project is the OLED SSD1306.

This module is composed of the SSD1306 OLED/PLED driver chip, a four pin connector, a voltage regulator and some external circuitry required by the IC. There are numerous manufacturers that produce this module, for this reason, the circuitry, the components and the operation ranges can vary between them. It is not necessary to delve into the external circuitry connected to the IC, however it is important to look into the regulator to know the exact operating range of our device.



(a) Front view **(b)** Rear view

Figure 5.9 – SSD1306 module [47]

Our module uses a XC6206P332 adjustable regulator whose input operating voltage range is 1.8 to 6V. This regulator is configured to obtain a 3.3 V output voltage, required for the SSD1302 IC, whose input voltage ranges from 1.65 to 3.3 V. As for the panel, its operating voltage ranges from 7 to 15 V but another power supply is not needed since the IC contains a voltage regulator that boosts its input voltage to the 7.5 V required for the panel.

Regarding the MCU interface, the SSD1306 IC provides “8-bit 6800/8080-series parallel, 3/4 wire Serial Peripheral and I2C interfaces” [44] but the module is designed to only allow the I2C interface. By default, the I2C address of this module is fixed to 0x3C but it can be changed to 0x3D with a solder jumper, however this will not be needed since it does not coincide with the addresses of the other I2C devices of the board.

This may be confusing since in figure 5.9 (b) it can be read that the selectable addresses are 0x78 and 0x7A. The reason is that there are different types of I2C addresses, there are 7 bits addresses that do not include the Write/Read bit and 8 bits addresses that do include it. Both addresses are the same but 0x78 is interpreted as a 8 bits address that contains the Write/Read bit and 0x3C is interpreted as a 7 bits address that does not include the Write/Read bit [3]. To see it clearer 0x78 and 0x3C will be represented in binary:

0x78: 0111 1000

0x3C: 0011 1100

The bit sequence "011110" is the fixed part of the slave address, the next less significant bit is the selectable part of the slave address (in this case 0) and, in the case of 0x78, the less significant bit is a Write bit. In the Adafruit SSD1306 library, which is the one that will be used to program the board, the address is interpreted as 0x3C.

Regarding the circuit design, it is correct so nothing will be modified (see sheet 9 of Appendix D).

5.2.8 Rotary encoder

A rotary encoder is a position and rotational speed sensor that generates electrical pulses according to the rotational displacement of their shaft. They are typically classified as absolute, if they have the capability to provide the absolute position of the shaft, or incremental, if they only provide the amount of rotational displacement of the shaft and the rotational direction [32].

This component will allow the user to navigate through the aforementioned screen menu when debugging or when configuring the operation options of our device. In order to achieve this, an incremental encoder type is valid. Concerning the model selection, we will use the PEC11R-4020F-S0024 because it is available in our laboratory. It is an incremental encoder with a 20 mm shaft and a resolution of 24 pulses per 360º rotation. Moreover, the shaft can be pressed acting as a button.

The operating principle of an incremental encoder is the following: It has three terminals, one of them in the middle connected to ground (terminal C) and two pulled up to 3.3 V in this case (terminals A and B). As the shaft is rotated, it short-circuits terminals A and C and then B and C (or in the opposite order depending on the rotational direction), resulting in pulsed voltages with a 90º phase difference in terminals A and B. The rotational direction can be determined from the relative position of these two phases [42]. To achieve a better understanding of how this device works, it has been tested and measured and its operation can be observed in figure 5.10.

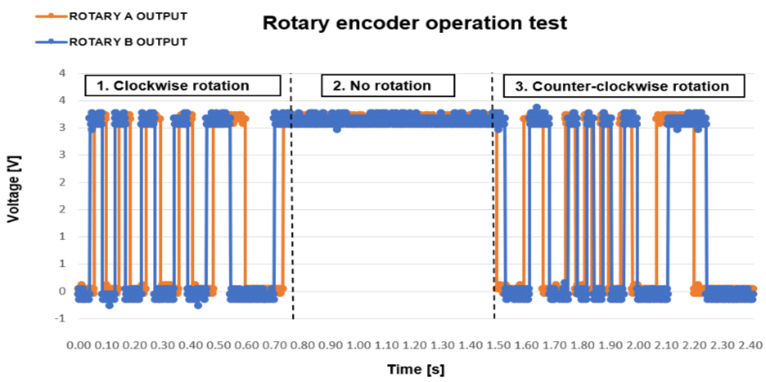


Figure 5.10 – Rotary encoder operation

The suggested design in the previous thesis is shown in figure 5.11 and it is correct, nonetheless, we can make some small changes that will make it cheaper. To correctly operate, the encoder requires pull-up resistors that drive the A and B rotary outputs and the rotary button to a logical high value. This can be done by using external 10 kΩ resistors as it is suggested in the encoder's datasheet, but in this case, we can take advantage of the internal 45 kΩ pull-up and pull-down resistors of the ESP32, saving three resistors. The value of the resistors is not important as long as they keep the current low, which they do:

$$I_{Rpullup} = \frac{3.3V}{45\ k\Omega} = 73.33\ \mu A \tag{5.2.8}$$

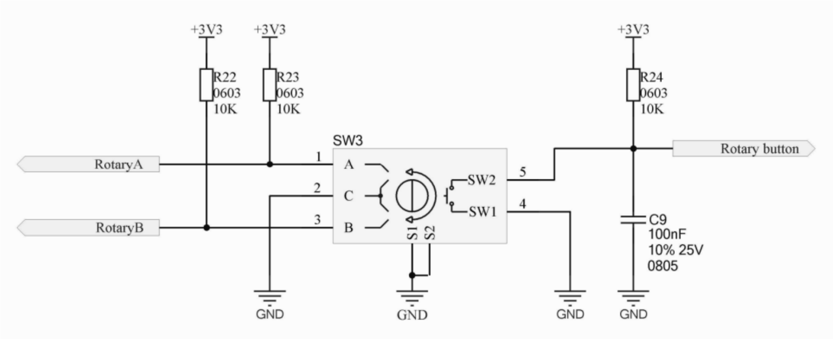


Figure 5.11 – Previous thesis' rotary encoder design

Every GPIO of the ESP32 except 34 to 39 contains both internal pull-up and pull-down resistors, in this case GPIOs 2, 27 and 5 have been connected to ports "Rotary A", "Rotary B" and "Rotary button" respectively. The final design is displayed in sheet 8 of Appendix D.

Component	**Price per unit [€]**
10 kΩ 0603 resistor x3	0.09 € (Mouser)
Total saved money [€]	0.27

Table 5.22 – Rotary encoder circuit saved money

Component	**Price per unit [€]**
PEC11R-4020F-S0024 x1	1.94 (Mouser)
100 nF 0603 capacitor x1	0.09 (Mouser)
Total cost [€]	2.03

Table 5.23 – Rotary encoder circuit cost

5.2.9 Real Time Clock (RTC)

It is an electronic device capable of keeping track of time through an oscillator and it is typically included to provide a precise date and time estimation that can be used by the MCU. It is not an essential part of our device, however, it allow us to implement additional features that would be useful:

- It could be used to set a schedule to automatically turn on and off the high power LED in the prow. This way, we will lower the power consumption because it will only be used when it is necessary. Moreover, we could adapt this schedule to the corresponding sunshine hours of each season.
- Once the trajectory calculation is finished and its trajectory length is determined, we could estimate the arrival time of the boat to the desired location and notify the user via the app.
- We could keep track of the time since different events such as the last charging of the battery, the last

firmware update, the last time the device was used and we could establish revision warnings at regular intervals. Notice that in order to keep this dates even after turning the device off it will be required to store them in the EEPROM.

Using an external RTC device is not strictly necessary since our microcontroller already contains two internal oscillators: One of them has a 150 kHz frequency and the lowest power consumption (10 μA), but it is less reliable as its stability is strongly affected by temperature fluctuations. The other one has a 8 MHz frequency (that can be divided by 256 obtaining 31.25 kHz) and it is much more stable, but its power consumption is higher (15 μA) [14]. Since our boat will work in variable temperature conditions, the 8 MHz oscillator would be chosen.

To achieve RTC functionality without an external module, the time and date that the sketch was compiled and uploaded is stored and a library such as TimeLib.h keeps track of time. However, this is not the best option for various reasons:

- Continuously keeping track of time supposes a higher computational complexity for the MCU, whose resources are already limited by the rest of the firmware. By using this module, the MCU will be released from these time tasks.
- We cannot turn off the ESP32 because it would lose track of time, instead we must put it into deep sleep mode, which still uses the RTC timer and the RC memory. However, this would prevent us for removing the battery or turning the device off completely.

When choosing the RTC, we encountered that the two most used and well documented modules for this kind of projects are the DS1302 and the DS3231. They will be compared in table 5.24.

Name	DS1302 Module IC's datasheet	DS3231 Module IC's datasheet
Main features	Year leap compensation	Interrupt pin, output clock pin, year leap compensation
Supply voltage	2 to 5.5 V	2.3 to 5.5 V
Primary source supply current	Typical: 0.425 mA Maximum: 1.28 mA	Typical: 200 μA Maximum: 300 μA
Backup battery supply current	Typical: 200 nA Maximum: 300 nA	Typical: 70 μA Maximum: 150 μA
Dimensions	31.2 x 15.5 mm^2	37 x 22 mm^2
Interface	3-Wire	I2C
Price	0.31 € (AliExpress)	1.52 € (AliExpress)
Choice	✓	✗

Table 5.24 – RTC tradeoff table

Both modules are very similar but DS3231 is more precise and has more functionalities. Nonetheless, the DS1302 was chosen because it was already available in our laboratory.

DS1302 is a RTC module capable of providing the time down to seconds and date with year-leap compensation. It contains an external 32.768 kHz crystal oscillator, which does not need any other components to operate and a battery holder for a Lithium button battery that allows it to keep track of time even when the ESP32 is off.

Regarding the circuit design of this module, it had a serious problem that would have made it unusable:

it was connected to GPIOs 6, 7 and 8, which are not recommended to use since they are connected to the integrated SPI flash. When tested this way, the operation was incorrect.

The tests displayed in 5.2.1 showed that they work fine as outputs but did not mention anything about their operation as input. This leads us to believe that they cannot operate correctly as inputs, which causes problems when using bidirectional communication. This should be studied with a digital analyzer to obtain a precise explanation of why this happens, but due to time constraints we will simply connect the RTC's pins to other GPIOs (CLK to 33, DAT to 12 and RST to 32).

5.2.10 Power supply

As we mentioned in Chapter 3 the boat's circuitry will be supplied by the 6 V lead acid battery. This could be achieved by simply connecting the battery directly to the different modules but this would not fulfill the functional and design requirements that were specified in 4.1 and 4.3. For this reason, the battery charger, the power switch button, the reverse polarity protection, an indication LED and the power monitor will be added to the circuit.

The first step is to revise the last thesis' circuit, which is in turn subdivided in two parts: the power switch button circuit and the rest, which are displayed in figures 5.12 and 5.13. In order to ease the analysis, each functionality will be revised separately. Note that all the components' and port's names in this section will refer to these figures:

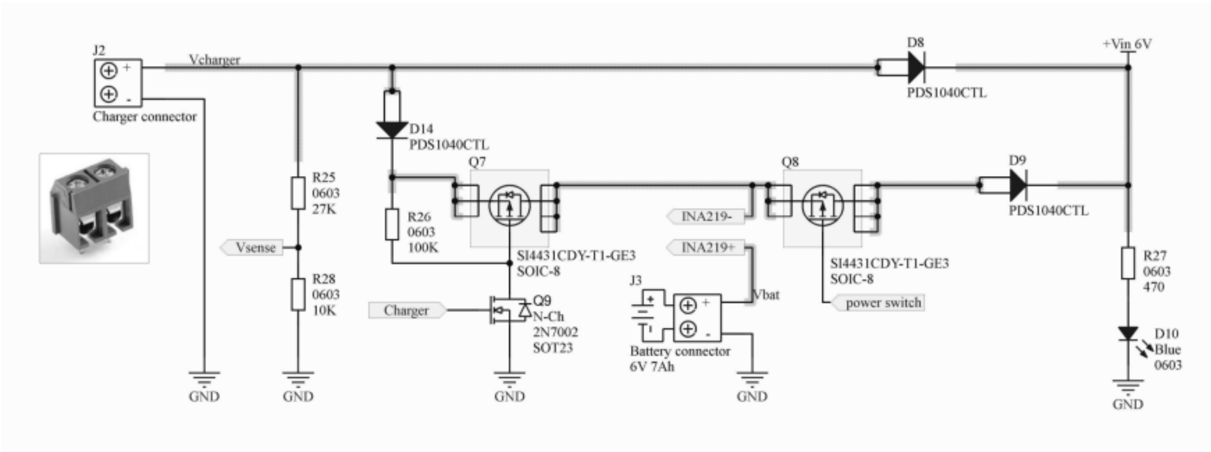


Figure 5.12 – Previous thesis' power supply circuit

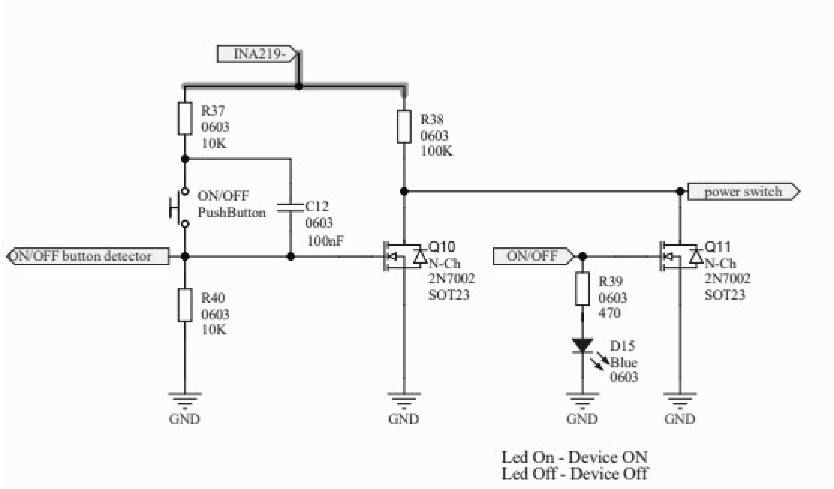


Figure 5.13 – Previous thesis' power switch circuit

5.2.10.1 Current and voltage monitor

A power sensor is required to track the current consumption and to estimate the state of charge of the battery, which prevents it from over discharges. In the previous thesis, the INA219 module was directly suggested to accomplish this task, however, we will research more alternatives that could be more optimal for our project. But, before that, we should estimate the current and voltage range that will be measured.

According to its datasheet, the battery voltage can range from 6 to 7.5 V when it is being charged and, regarding the current, we can estimate it by adding the consumption of each component, which will range from 9.93 to 13.14 A (table 5.25). Keeping this in mind, we can start comparing different current sensing options (table 5.26).

Component	Supply voltage [V]	Typical current [mA]	Maximum current [mA]	Typical power [mW]	Maximum power [mW]
ESP32	3.3	68	240	224.4	792
CH340C	3.3	12	30	39.6	99
DS1302	3.3	0.43	1.28	1.42	4.22
INA219	3.3	1	1	3.3	3.3
Adafruit 10-DOF	3.3	6.21	7.21	20.5	23.79
SSD1306	3.3	0.150	0.150	0.495	0.495
SIM800L	4	125	2000	500	8000
GY-NEO6MV2	4	48.1	71	192.4	284
SMD LEDs x3	3.3 to 7.5	30	30	133	148
5mm LEDs x3	6 to 7.5	24	24	144	180
High power LED	6 to 7.5	36	36	216	270
DC motor	6 to 7.5	9500	10000	60000	75000
Servo motor	6 to 7.5	75	700	450	562.5
Total	3.3 to 7.5	9925	13140	61925	85367.3

Table 5.25 – Global circuit's power consumption

Name	INA219 Module IC's datasheet	ACS712 Module IC's datasheet	MAX471 Module IC's datasheet	TMCS1100A2 Datasheet
Type	Current shunt and power sensor	Hall effect current sensor	Internal resistor power sensor	Hall effect current sensor
Supply voltage	3 to 5.5 V	4.5 to 5.5 V	3 to 36 V	3 to 5.5 V
Supply current	Typical: 1 mA Maximum: 1 mA	Typical: 7.5 mA Maximum: 12 mA	Typical: 50 μA Maximum: 113 μA	Typical: 6 mA Maximum: 6 mA
Current sense	±3.2 A (can be increased adding shunt resistors)	±10 A	±3 A (can be increased adding shunt resistors)	±14.5 A
Voltage sense	0 to 26 V	None	$5 \cdot V_{\text{supply}}$	None
Dimensions	26 x 22 x 6 mm^3	32 x 13 x 14 mm^3	20 x 19 x 8 mm^3	5 x 4 x 2 mm^3
Interface	I2C	Analog signal	Analog signal	Analog signal
Price	1.06 € (AliExpress)	1.79 € (AliExpress)	1.59 € (AliExpress)	4.04 € (Mouser)
Choice	✓	✗	✗	✗

Table 5.26 – Current/power monitors trade-off table

From these options, the best ones are the INA219 and the MAX741 since they are able to sense current and voltage simultaneously. However, INA129 is still a better option because it does not require any additional pins as the I2C pins had to be used anyway by other modules and the MAX741 production was discontinued.

The INA219 module is a measures current through a 0.1 Ω shunt power resistor with a ±320 mV shunt voltage, which results in a maximum current of:

$$I_{max} = \frac{320\ mV}{0.1\ \Omega} = 3.2\ A \tag{5.2.9}$$

Then, if we want to increase the maximum current we must use a smaller shunt resistance but, since it is difficult to find power resistors smaller than 0.1 Ω, we will place resistors in parallel. When calculating the resistor value we will consider the typical current consumption as the maximum value includes peak currents, assumes that the battery is charged to its peak and it is very unlikely to happen. Then, the shunt resistor value we need to place to measure ±9.92 A can be calculated as:

$$R_{shunt\ total} \leq \frac{320\ mV}{9.92\ A} \leq 0.0322\ \Omega \tag{5.2.10}$$

This value can be achieved by placing four 0.1 Ω resistors in parallel, so we would obtain a 0.025 Ω shunt resistance. Then, the maximum current that can be measured is:

$$I_{max} = \frac{320\ mV}{0.025\ \Omega} = 12.8\ A \tag{5.2.11}$$

Moreover, we must ensure that these resistors can handle such current. Through each of these resistors it would circulate a fourth of the current, 2.48 A, so the power that each of them had to dissipate would be:

$$P_{shunt} = R_{shunt} \cdot I^2 = 0.1 \cdot 2.48^2\ = 0.61\ W \tag{5.2.12}$$

In that case, we will use the the HoGT2512-2W-100mR resistor which can dissipate up to 2 W and has the same package as the resistor included in the INA219 module. The modified design can be observed in sheet 13 of Appendix D.

Component	**Price per unit [€]**
INA219 Module x1	1.06 (AliExpress)
HoGT2512-2W-100mR x3	0.06 (LCSC)
Total cost [€]	1.24

Table 5.27 – Power monitor circuit cost

5.2.10.2 Power switch

This circuit will allow the user to turn the boat's electronics on and off by pressing a button, instead of manually connecting and disconnecting the battery, which is tedious and does not measure up to the industry standard. This circuit achieves this by placing a P-Channel MOSFET transistor (Q8) in series with the battery, behaving as a switch. The behaviour of this transistor is determined by the voltage between its source and gate terminals (V_{SG8}), but in this case the source is connected to the battery (which can be considered as a fixed voltage) so the transistor's behaviour will be controlled by the gate's voltage (the "power switch" port). For the Q8 transistor, the SI4431CDY-T1-GE3, whose technical specifications are listed in table 5.29, was proposed.

The two main conditions when choosing this transistor are: having a sufficiently low $V_{\mathrm{SG(th)}}$ to ensure that we can surpass it and enter in saturation mode and being able to operate at the maximum current consumption of its following circuit. If we used the original design, the second condition would not be fulfilled because the DC motor, which has proven to consume around 9.5 A, was supplied by the "+Vin 6V" port of figure 5.12 and the maximum continuous current of Q8 is just 9 A in the best case scenario. This will be fixed by connecting the DC motor directly to the battery (this will be discussed in more detail in section 5.2.12).

By adding the components that are supplied from the "+Vin 6V" port (including the "+3V3" and the "+4V" ports) we can approximate the current that transistor Q8 must handle, the results are shown in table 5.28 (they are the results of table 5.25 minus the DC motor). From them, we can confirm that transistor Q8 is appropriate since it allows a higher current than that.

"+Vin 6 V" port typical current [mA]	"+Vin 6 V" port maximum current [mA]
425	3140

Table 5.28 – "+Vin 6V" port's current consumption

Name	SI4431CDY-T1-GE3 Datasheet
Type	P-Channel MOSFET
Maximum continuous drain current	-5.6 A ($T_A = 70$ºC) -9 A ($T_C = 25$ºC)
Maximum $\mathrm{R}_{\mathrm{DS(on)}}$	0.049 Ω ($V_{\mathrm{SG}} = 4.5$ V)
$V_{\mathrm{SG(th)}}$	1 to 2.5 V
Package	SOIC-8
Price	0.71 € (Mouser)

Table 5.29 – SI4431CDY-T1-GE3 specifications

In view of the SI4431CDY-T1-GE3 datasheet, we know that Q8 will conduct whenever $V_{\mathrm{SG8}} > \mathrm{V}_{\mathrm{SG(th)}}$

(2.5 V maximum). Then, if V_{S8} is the battery voltage (6 V), Q8 will definitely conduct when the "power switch" port's voltage is lower than 3.5 V. This port's voltage will be controlled by the ON/OFF button and the MCU, its operation is as follows:

- If the button is not pressed and the circuit is off, the "ON/OFF button detector" port's voltage will be 0 V because of the pull-down resistor R40 so the Q10 N-Channel transistor will not conduct, driving the "power switch" port's voltage to V_{bat} through R38. If this is the case Q8 will not conduct, so the battery would be disconnected from the circuit.

- If the button is pressed, the "ON/OFF button detector" port's voltage will be 3.3 V if V_{bat} is 6.6 V because of the voltage divider implemented with R37 and R40:

$$V_{ON/OFF\ button\ detector} = \frac{V_{\text{bat}} \cdot R40}{R37 + R40} \tag{5.2.13}$$

  If that is the case, Q10 will conduct so the "power switch" port voltage will be 0 V. This will result in Q8 conducting, which connects the battery to the circuit and turns on the MCU.

- Once the MCU is on, it will drive the "ON/OFF" port to 3.3 V almost instantly, thus Q11 conducts, keeping the "power switch" port at 0 V so the battery does not disconnect from the circuit. At this point, the power switch button can be released.

- If we want to turn off the circuit, we can program the MCU so when it detects that the ON/OFF button has been pressed while it was on, it drives the "ON/OFF" port to 0 V. This way Q11 would stop conducting and the voltage at the "power switch" port would be V_{bat} so the battery would be disconnected from the circuit.

Regarding the circuit design, it is correct for the most part, however there are a few details that could cause problems:

- When the MCU is off, the "ON/OFF" port's voltage would be floating. If this is the case the MOSFET static charge could built at the gate to source parasitic capacitance and turn on the transistor and the circuit randomly. This can be easily fixed by placing a pull-down resistor, 10 kΩ resistors are typically used when working with MOSFETs.

- The voltage divider composed of R37 and R40 would drive the "ON/OFF button detector" port's voltage to an acceptable level when the battery is not charging. Nonetheless, even though the boat is not though to operate while the battery is being charged, if the button was accidentally pressed in this situation, the ESP32 could be damaged because V_{bat} could reach 7.5 V and the "ON/OFF button detector" port's voltage would be 3.75 V (equation 5.2.13). This voltage exceeds the maximum supply voltage of the ESP32, which could lead to damage.

  This can be fixed by using a lower value for R40, as long as the minimum voltage of the battery (which will be 6 V) surpasses the ESP32 input logic level voltage, which is 2.5 V approximately. If we R40 is replaced by 8 kΩ, the "ON/OFF button detector" port's voltage will be 3.3 V when V_{bat} is 7.5 V and 2.6 V when V_{bat} is 6 V.

The modified power switch circuit can be seen in sheet 12 of Appendix D.

Component	Price per unit [€]
10 kΩ 0603 resistor x2	0.09 (Mouser)
8 kΩ 0603 resistor x2	0.40 (Mouser)
100 kΩ 0603 resistor x1	0.09 (Mouser)
100 nF 0603 capacitor x1	0.09 (Mouser)
2N7002 x2	0.34 (Mouser)
SI4431CDY-TE1-GE3 x1	0.71 (Mouser)
Push button x1	0.19 (Mouser)
Total cost [€]	2.83

Table 5.30 – Power switch circuit cost

The biggest problem of this circuit the ESP32 GPIOs choice. At first sight they look correct but when this circuit was tested with a simplified version of the circuit (figure 5.14) based on the NodeMCU-32s (an ESP32-based module), it did not work. The reason is that the "ON/OFF button detector" port has to be connected to a GPIO that is configured by default as an input after reset, if not, the "ON/OFF button detector" port's voltage when the button is pressed does not correspond to the voltage divider calculation. GPIO36 was the the pin that was originally assigned, but even though it is an only input pin, it is not set as input after reset. For this reason, it will be replaced by GPIO23, which is fulfills the condition and has proven to work fine experimentally.

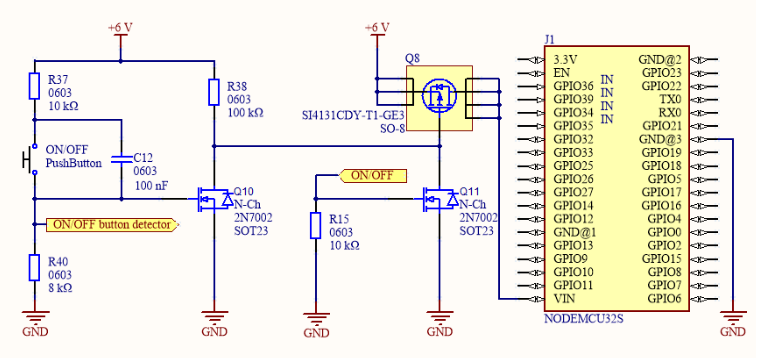


Figure 5.14 – Power switch simplified test circuit

5.2.10.3 Reverse polarity protection

An important feature that should be added to our circuit is a reverse battery polarity protection. Occasionally, batteries are placed with their polarity reversed due to human error which could damage partially or totally the circuit they are connected to. To avoid these risks, a protection circuit that disconnects the battery when it is incorrectly placed must be added. There are multiple options:

- Protection using a diode: this method consists in placing a diode in series with the battery. When the battery is placed correctly, the diode will conduct and the circuit will work. If the battery is placed with reverse polarity, the diode will not conduct so the circuit will not be damaged.

  This option is easy to implement as it is composed of just one diode. In this case, it is essential that the diode has a very low reverse current and that it can handle at least the 3.14 A calculated in table 5.28. Furthermore, as efficiency is an important concern in this application, a diode with a low forward voltage should be chosen to minimize the dissipated power.

- Protection using a N-Channel MOSFET: this option uses a N-Channel transistor to disconnect the battery, which usually dissipates less power than a diode. Nevertheless, the disadvantages of this circuit are that it is more complex and more expensive, as it requires more components. The circuit is as follows [10]:

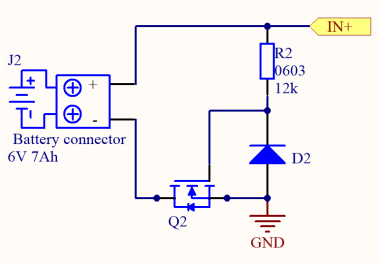


Figure 5.15 – Reverse polarity protection using a MOSFET

When the battery is placed correctly, the Zener diode will be reverse biased, fixing the transistor's V_{GS} to the Zener diode's reverse voltage. By choosing this voltage so it is higher than the transistor's $V_{\text{GS(th)}}$, the transistor will conduct so the battery be connected.

If the battery is placed incorrectly, the Zener diode will be forward biased, fixing the transistor's V_{GS} to the the Zener diode's forward voltage. By choosing this voltage so it is lower than the transistor's $V_{\text{GS(th)}}$, the transistor will not conduct so the battery will be disconnected, preventing damage.

The resistor is used to limit the current that will flow through the Zener diode, so its value should be high.

In the previous thesis, the first option was implemented with the D9 diode of figure 5.12. This design is correct and will be kept but it will be slightly modified. In the previous design, a PDS1040CTL diode was used. This has been replaced by the SSL56F diode, which has very similar characteristics (table 5.31), since the other one was not available in our distributor's site.

Name	SSL56F Datasheet
Type	Schottky diodes barrier
Forward voltage	0.5 V
Maximum rectified current	5 A
Package	SMAF
Price	$ 0.07 (LCSC)

Table 5.31 – SSL56F specifications

5.2.10.4 Battery charger

The charging process of a lead acid battery is complex and has different parts, which requires a specific charger. Typically, an external charger is used, but in this case, we will integrate the charger into the board so it will not be necessary to remove the battery from the boat in order to charge it.

The performance and life of a lead acid battery significantly depends on its charging procedure, therefore it is important to ensure that the charging method is the most appropriate one and that it is precisely executed. The charging method typically differs depending on the service type of our battery: float or cyclic.

- The float service is used to maintain the battery charged when it is not being used. To achieve this, it must be constantly charged to counteract the self discharge process. For the most part, our boat will work outdoors without access to the electrical grid so this mode will not be used.
- The cyclic service is typically used in portable equipment, such as ours. In this case, the battery is periodically charged and then discharged until reaching the end voltage (around 5.8 V for our battery)

[7]. If the battery is discharged below this point it will lose its ability to receive charge. In this mode, several methods could be used but our battery's datasheet suggests the curve in figure 5.16 which corresponds to a constant current constant voltage charge.

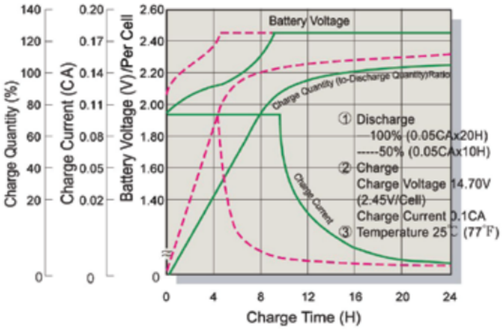


Figure 5.16 – Battery charge curve [17]

This is be the method that will be implemented and, as its name suggest, it is composed of two stages:

- Constant charge current: the current limited to approximately 0.1 CA (0.7 A for our battery) is applied to the battery making the voltage increase gradually until it reaches the set voltage limit, in this case about 2.45 V per cell (7.35 V for the whole battery).
- Constant voltage: at this point, the set voltage has been reached which leads the battery to a saturation state that gradually limits the charge current. Once the current reaches the lowest point of the graph (figure 5.16) we can conclude that the battery is fully charged.

From these two stages, it is obvious that we need a device capable of monitoring and limiting current and voltage. Usually, this is accomplished with an that abstracts the designer from the charge method's characteristics. However, in last year thesis this option was dismissed since some of these circuits were too small to manufacture in our laboratory and the rest were out of stock. For these reasons, the left half of the circuit in figure 5.12 was proposed.

This circuits detects and monitors a power source in the charger connector through a voltage divider (R25 and R28). This external source could range from 7.5 to 12 V, so this voltage divider has be able to adapt 12 V to 3.3 V maximum. This can be achieved with R25= 27 $k\Omega$ and R28= 10 $k\Omega$, proven as:

$$V_{sense} = \frac{12\ V \cdot 10k\Omega}{27\ k\Omega + 10\ k\Omega} = 3.24\ V \tag{5.2.14}$$

If this source is detected, the charging process will begin. The voltage and current that is supplied to the battery is controlled through a pass transistor (Q7), in turn, controlled through transistor Q9 and the "Charger" DAC pin of the MCU. For the pass transistor, the SI4431CDY-T1-GE3 (table 5.29) will be used too since it has a low $\mathrm{V_{SG(th)}}$ and can handle the maximum charge current (around 0.7 A).

The battery's charge voltage and current will be monitored by the aforementioned INA219 and according to that, the "Charger" pin voltage will adjust the pass transistor to achieve the desired charge curve. This circuit was simulated in LTSpice® in the previous thesis and seemed to work fine, so it will not be modified. Depending on its actual performance in this prototype some changes could be made.

Regarding the diodes, D14 works as a reverse polarity protection for the external power source and D8 allows power to the the "+Vin 6V" port when the battery is connected. However, this last one will only be used for the debugging process, so we can try the circuit without connecting it to the battery.

The full circuit can be consulted in sheet 11 of Appendix D.

5.2.11 USB interface

When using a MCU, it is required some kind of port and circuit that allows it to interface with the computer easily. This interface is required to write the firmware and to visualize data in the computer's serial terminal. This is typically achieved through an USB port. Specifically, we will use a mini type B connector since it is an industry standard, commonly found in the ESP32 modules, easy to solder and available in our laboratory.

In order to connect the communicate the MCU with the computer, the ESP32 UART needs to send and receive asynchronous serial data through that USB port. Some MCUs include built-in USB peripherals that support this type of communication through the implementation of USB CDC (for example, the ESP32-S2) [15], nevertheless this is not the case in our ESP32, which requires an external IC to translate the USB data to UART, and vice versa. The main advantages of using an external USB to UART bridge are that no firmware is required to implement a serial class, which complicates and slows down the firmware development process and also the MCU consumes less computational resources since the USB protocol is handled by the bridge chip [36].

When designing this circuit, a reverse engineering process was carried out from the NodeMCU-32s board, which uses the CH340C as an USB-to-UART bridge. This IC is available in our laboratory and its specifications are listed in table 5.32.

Name	CH340C Datasheet
Type	USB-to-UART/RS232/RS485 bridge
Supply voltage	3.3 to 3.8 V if the V3 pin is connected to the VCC pin, if not 4.5 to 5.3 V
Absolute maximum voltage	-0.3 to 6 V
Current	Typical: 12 mA Maximum: 30 mA
Package	SOP-16
Price	0.45 € (Utmel)

Table 5.32 – CH340C specifications

The only external component that this IC requires according to the datasheet is a 0.1 μF power decoupling capacitor in the VCC pin to suppress high frequency noise. Nevertheless, it is typical to include an ESD protection in the USB to UART circuits; this chip is especially sensitive to power surges, which can be produced by the electrostatic discharges that are common when plugging and unplugging the USB. When an ESD takes place, a voltage ranging from tens to thousands of volts can be generated [35] and, in this case, just 6 V is enough to permanently damage the CH340C. The USBLC6-2 was chosen last year as it was available in our laboratory stock, its characteristics are listed in table 5.33. The previous thesis design is displayed in figure 5.17.

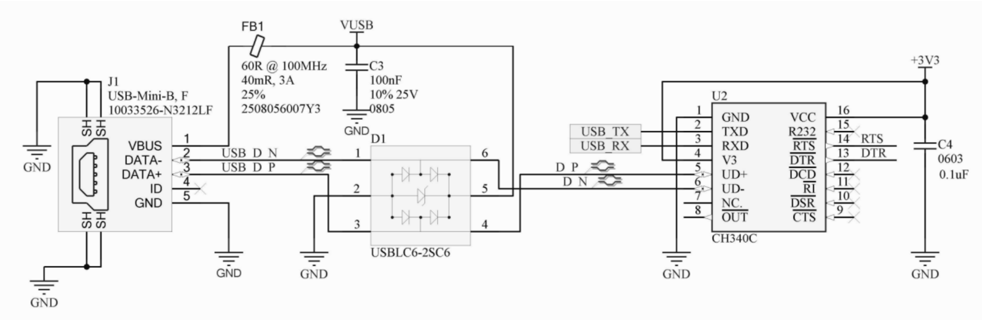


Figure 5.17 – Previous thesis' USB to UART circuit

Name	USBLC6-2 Datasheet
Type	ESD protection
Clamping voltage	12 to 17 V
Number of lines protected	2 data-line and VBUS
Maximum leakage current	150 nA
Package	SOT23-6L
Price	0.56 € (Mouser)

Table 5.33 – USBLC6-2 specifications

However, after carefully examining its characteristics it was noticed that its Clamping voltage is too high. The ESD suppressor must be chosen so that its Clamping voltage is lower than maximum voltage of the device it is protecting [35], therefore a new ESD suppressor must be chosen. In table 5.34 the different options' details are listed.

Name	PESD3V3X4UHCYL Datasheet	BZA408B Datasheet	ESDS304 Datasheet
Clamping voltage	4.5 V	-5 to 5 V	3 to 6 V
Lines protected	4	4	4
Maximum leakage current	100 nA	100 nA	50 nA
Package	DFN1308-6	SOT457	5-Pin SOT23
Price	0.38 € (Mouser)	0.46 € (Mouser)	0.52 € (Mouser)
Choice	✗	✗	✓

Table 5.34 – USBLC6-2 alternatives trade-off

Technically, the PESD3V3X4UHCYL would be the best option since it is the cheapest one, has the lowest and a small leakage current. However, its package is too small to be manufactured in our laboratory so it has to be discarded. The BZA408B would be a good option as it allows a 5 V Clamping voltage, nevertheless it also allows a negative voltage which could destroy the CH340C. Finally, the ESDS304 was chosen because, even though its maximum Clamping voltage is the same as the absolute maximum voltage of the CH340, it still protects that component. It also has a very low leakage current and its package can be easily manufactured in our laboratory.

Moreover, in the previous thesis design a ferrite bead and a 100 nF capacitor were added to the USB "VBUS" pin reduce its noise.

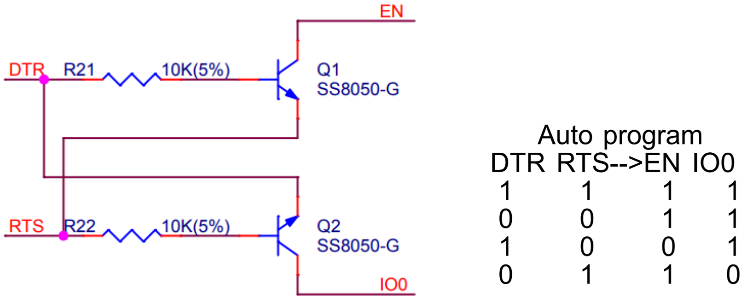
The total cost of this circuit is estimated in table 5.35:

Component	Price per unit [€]
10033526-N3212LF x1	0.67 (Mouser)
40 mΩ Ferrite Bead 0805 x2	0.42 (LCSC)
100 nF 0603 capacitor x2	0.09 (Mouser)
CH340C x1	0.45 (Utmel)
ESDS304 x1	0.52 (Mouser)
Total cost [€]	2.66

Table 5.35 – USB to UART circuit cost

5.2.11.1 Boot mode selection

A bootloader is a small piece of software that allows uploading of sketches onto the microcontroller [5]. Depending on the state of GPIO0, the ESP32 will enter different bootloader modes. If GPIO0 is pulled high, the program is executed normally and if is pulled high, it enters in programming mode. Moreover, the board can be reset by pulling low the EN pin. This can be automatically selected connecting the DTR and RTS pins of the USB to UART converter and implementing the following circuit commonly found in different ESP32 modules:



Auto program

DTR	RTS-->	EN	IO0
1	1	1	1
0	0	1	1
1	0	0	1
0	1	1	0

(a) Auto-reset circuit **(b)** Truth table

Figure 5.18 – ESP32 Devkitc V4 bootloader circuit

The only modification that was made to this circuit is to replace the obsolete BJT transistors with MOSFETs, which do not require gate resistors. Apart from that, two buttons will be included to access the different modes manually:

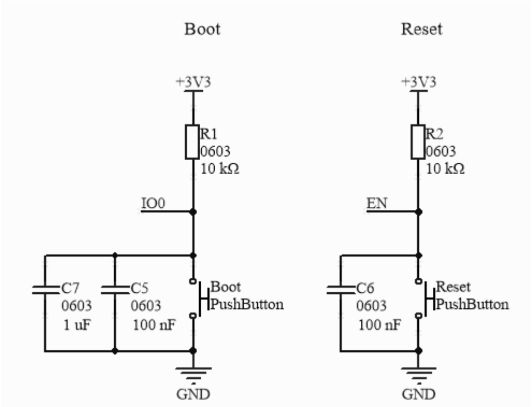


Figure 5.19 – Bootloader modes buttons

This design can be optimized by removing the pull-up resistors since those pins are pull-up high after reset through internal resistors. Moreover, in the ESP32 documentation it is recommended to place a capacitor ranging from 0.1 to 1 μF in the EN pin, if that is the case, no capacitors are required in the IO0 pin. The total cost of this circuit is estimated in table 5.36:

Component	Price per unit [€]
2N7002 x2	0.34 (Mouser)
100 nF 0603 capacitor x1	0.09 (Mouser)
Total cost [€]	0.77

Table 5.36 – Boot mode selection circuit cost

5.2.11.2 USB-powered debugging

A new addition is that, in order to facilitate the testing and debugging of the ESP32 code, the 5 V VBUS pin of the device's USB connector will be connected through a diode to the "Vin+" pin of the XL1509 regulator module (explained in section 5.2.13) that provides 3.3 V.

This allows the user to test the ESP32 and the 3.3 V supplied modules connected to it without the necessity to connect the circuit to the battery or another external 6V source, by simply connecting the it to a PC. This is translated into less usage of the battery, so it will deteriorate slower, and much more flexibility for the user.

When the battery is connected, the voltage of the "Vin+" pin of the regulator will be set to 6 V. In this case the cathode's voltage of the diode will be higher than the anode's so it will not conduct, then, the board will be supplied from the battery. If the battery is disconnected, the diode will conduct so the voltage of the "Vin+" pin of the regulator will be VBUS voltage (5 V) minus the forward voltage of the diode, supplying the 3.3 V components of the board from the PC.

We are using a 10033526-N3212LF female USB 2.0 connector which can supply a 0.5 A current, enough to supply the regulator. Then, it must be ensured that the diode can handle 0.5 A and have a low forward voltage as the minimum input voltage of the regulator is 4.5 V. The VS-30BQ015-M3/9AT diode fulfills these specifications (table 5.37).

Name	VS-30BQ015-M3/9AT Datasheet
Type	Diode
Maximum average forward current	3 A
Maximum leakage current	50 mA
Package	DO-214AB-2
Price	0.58 € (Mouser)

Table 5.37 – VS-30BQ015-M3/9AT specifications

Even though we have removed the pull-up resistors, their pads will remain in the PCB in case unexpected errors appeared in this design.

5.2.12 DC motor

As it was mentioned in Chapter 3, the boat contains a DC motor to propel it through the water. The boat's speed can be controlled by limiting the current that flows through this motor, typically achieved through PWM, and the direction of rotation can be reversed by inverting the polarity of its voltage, allowing the boat to move forward and backward. These capabilities can be achieved with an H-bridge.

An H-bridge is a circuit that can switch the polarity of the voltage applied to a load, in this case a DC motor. Its operation principle is simple: the circuit is composed of four switches (figure 5.20), when S1 and S4 are closed and the others are open, the current flows from the positive to the negative terminal of the motor and when S2 and S3 are closed and the rest are open, the current flows in the opposite direction, switching the direction of rotation. If all the switches are open, no current will flow but if all of them are closed, both branches will be short-circuited and the circuit could be damaged.

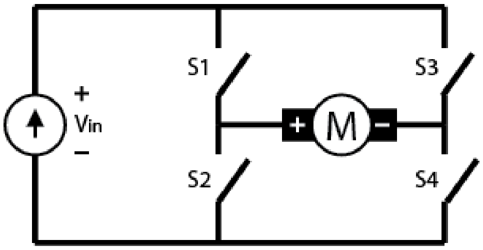


Figure 5.20 – H-bridge circuit [16]

In the previous thesis' design, the L298N Dual H-Bridge module was chosen, however, after testing the boat in the water and measuring the motor's current, this module has to be discarded. The reason is that this module's maximum current is 4 A and the motor consumes around 9.5 A. There are other H-bridge modules options, such as the BTS7960 module, that handles currents up to 43 A. However, they are based on out-dated ICs that are not available anymore so they are not a good option.

For that reason, we will design our own H-bridge circuit. Even though the basic operation of this circuit is simple, there are many factors that have to be taken into consideration when designing it. The first one is the switches, we need a device that can be controlled by the MCU and that can handle large currents. These conditions are fulfilled by MOSFETs.

Regarding the exact MOSFET model, it will be chosen through a reverse engineering process on the boat's original H-bridge circuit, displayed in figure 5.21. This circuit used six N-Channel IRL3803 MOSFETs: two of them in parallel for S1 and for S4 and one for S3 and S4 (referred to figure 5.20). The reason for using 2 transistors in parallel is to dissipate less power and heat through each transistor when the motor is operating forward, which is the most usual operation mode. The IRL3803 technical specifications are listed in table 5.38.

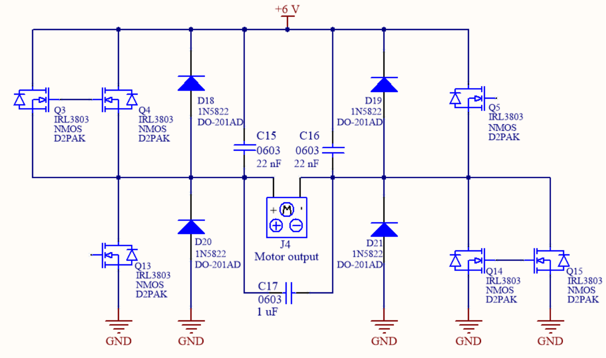


Figure 5.21 – Boat's original H-bridge circuit

Name	IRL3803STRLPBF Datasheet
Type	N-Channel MOSFET
Surface mount application power dissipation	2 W
Maximum $\mathrm{R_{DS(on)}}$	0.009 Ω
$\mathrm{V_{GS(th)}}$	1 V
Package	TO-252-3
Price	2.35 € (Mouser)

Table 5.38 – IRL3803 specifications

For the average current motor (9.5 A), the IRL3803's power dissipation can be calculated as:

$$P_{dissipated} = R_{\mathrm{DS(on)}} \cdot I^2 = 0.009 \cdot 9.5^2 = 0.81\ W \tag{5.2.15}$$

The power that will be dissipated is lower than the surface mount application power dissipation, thus, this transistor is suitable for our project. Nevertheless, using N-Channel MOSFETs has one notable downside: it requires a high side driver circuit to control the transistors above the load. The reason is that if we set the gates of the high side transistors to 6 V, they will enter saturation mode, setting their source gate to 6 V. When this happens, their $\mathrm{V_{GS}}$ will become 0 V and they will stop conducting, becoming useless.

This could easily be fixed by using P-Channel MOSFETs for the high side, nevertheless, that is not possible in our project because we are operating at high currents. P-Channel MOSFETs have larger $\mathrm{R_{DS(on)}}$ than the N-channel ones and can dissipate less power so the. If we take a look at the IRFR5305, which is a P-Channel power MOSFET, its maximum surface mount application power dissipation is 2 W and have a $\mathrm{R_{DS(on)}}$ of 0.065 Ω. Then, the power dissipation would be:

$$P_{dissipated} = R_{\mathrm{DS(on)}} \cdot I^2 = 0.065 \cdot 9.5^2 = 5.87\ W \tag{5.2.16}$$

The dissipated power in a P-channel MOSFETs is much larger than the N-channel's and cannot be dissipated by its package, which would burn the device. Therefore, a high side driver circuit is necessary.

Most high side drivers use a bootstrap circuit to control the transistor's operating point. These circuits use a capacitor that is charged to a certain voltage level while the high side transistor is off and then sets that transistor's $\mathrm{V_{GS}}$ to the capacitor voltage, turning it on until the capacitor discharges. For that reason, these drivers require PWM since they cannot keep the transistor on for a long time.

There are high and low side drivers ICs that would fulfill our requirements such as the L6389ED (specifications in table 5.39). However, since we have to prioritize the usage of the available components of our laboratory, we will design a basic high side driver based on the operation of a bootstrap capacitor with the available components. This circuit is displayed in figure 5.22.

Name	L6389ED Datasheet
Type	High and low-side driver
Supply voltage	-0.3 to 17 V
Maximum voltage rail	600 V
Logic inputs	3.3 V, 5 V and 15 V
Maximum switching frequency	400 kHz
Package	SO-8 tube
Price	1.53 € (Mouser)

Table 5.39 – L6389ED specifications

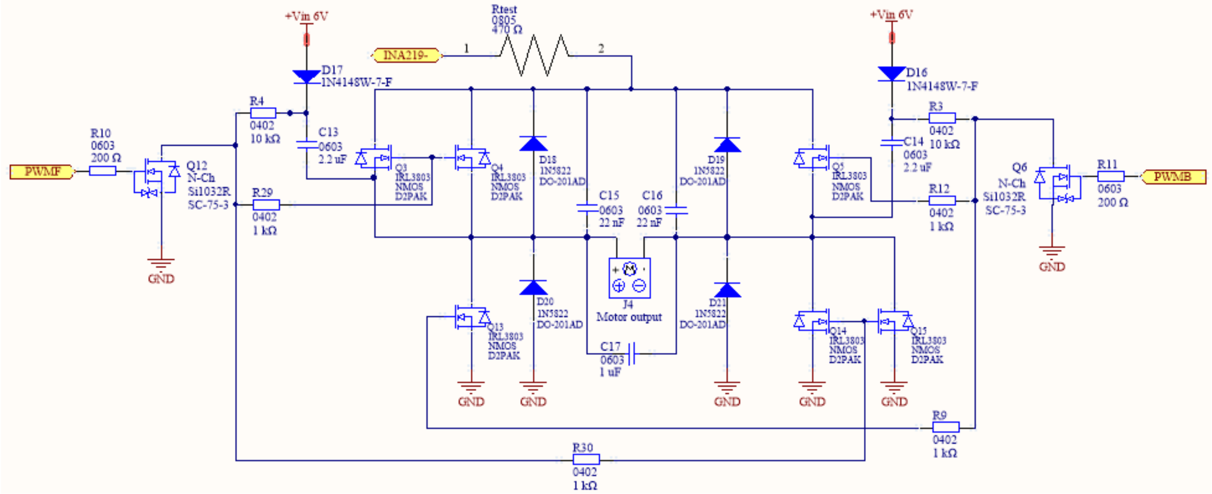


Figure 5.22 – Our H-bridge circuit

To explain its operation only one side will be analyzed. Firstly, the "PWMF" pin (forward PWM signal) is set to 3.3 V, thus transistor Q12 is on, which sets its drain voltage to 0 V. This results in having 0 V on the Q3 and Q4 gates so they will not conduct, that is to say, the motor will be stopped. Also, capacitor C13 will be charged to the battery's voltage through diode D17.

When the "PWMF" pin is set to 0 V, then Q12 will be off and we will have the capacitor voltage (equal to the battery's) between the gate and the source of Q3 and Q4, turning them on. If these transistors are on, they will have the battery's voltage in their source, thus, the gate voltage of Q14 and Q15 will be approximately 11 V and current will circulate through the motor.

One of the downsides of this bootstrap circuit is not being able to use a duty cycle of 100%. This is due to the bootstrap capacitor, which will discharge when the motor is on, eventually turning off the high side MOSFET. The capacitor will discharge through R4 and R29 and through the leakage current of the diode. Nevertheless, this should not be a problem since we can still use a high duty cycle in the order of 90%, which would be enough to drive the motor at a high rotational speed.

We can check that all these transistors are in saturation mode by measuring the voltage across the motor terminals, which should be almost equal to the battery's since $R_{DS(on)}$ is very close to 0 Ω.

The purpose of resistor R4 is to avoid a short circuit between Q12 drain and the cathode of D17 when the "PWMF" pin is set to 3.3 V. Its value is chosen so a small current circulates through it, if we want a current of 0.5 mA, then R4 can be calculated as shown in equation 5.2.17, where V_{in} is the battery's voltage and V_F is the diode's forward voltage.

$$R4 = \frac{V_{\text{in}} - V_F}{I} = \frac{6\ V - 1\ V}{0.5\ mA} = \ 10\ k\Omega \tag{5.2.17}$$

R29 and R30 are gate resistors, they mitigate the ringing effect of the MOSFETs and limit their parasitic gate current to a desired value. In this case, 1 $k\Omega$ resistors will be used, which limit the current to:

$$I = \frac{V_G - V_S}{R} = \frac{6\ V + 5\ V - 0\ V}{1\ k\Omega} = \ 11\ mA \tag{5.2.18}$$

R10 accomplish the same function as R29 and R30, it limits the parasitic gate current that can be supplied by the ESP32. The maximum current that can be supplied from an ESP32 pin is 20 mA, then, using a 200 Ω gate resistor the current will be limited to:

$$I = \frac{V_G - V_S}{R} = \frac{3.3\ V - 0\ V}{200\ \Omega} = \ 16.5\ mA \tag{5.2.19}$$

Regarding the capacitor's value, it is recommended to use one whose charge is much greater than the the charge required by the high side transistor's gate (in this case the IRL3803). The reason for this is to keep the voltage source drop due to charge sharing small [6].

$$Q_{\text{cap}} >> Q_g \tag{5.2.20}$$

In equation 5.2.20, Q_{cap} refers to the bootstrap capacitor's charge whereas Q_g refers to the high side transistor's total gate charge, which is 140 nC for the IRL3803. If the capacitor's charge is chosen to be 80 times greater than the gate charge, the following equation will be used to obtain the capacitor's value, where C_{cap} is the capacitance of the bootstrap capacitor and V_{cap} is the voltage that the capacitor will be charged to, approximately 5 V.

$$Q_{\text{cap}} = \ 80 \cdot Q_g\ ; C_{\text{cap}} \cdot V_{\text{cap}} = \ 80 \cdot Q_g\ ; C_{\text{cap}} = \frac{80 \cdot Q_g}{V_{\text{cap}}}\ ; C_{\text{cap}} = \frac{80 \cdot 140\ nC}{5\ V}\ ; C_{\text{cap}} = 2.24\ \mu F \tag{5.2.21}$$

Regarding the component selection, diodes 1N4148W were already available in our lab and suit our circuit due to their high switching frequency and Si1032R were chosen due to their fast switching speed and their low voltage operation. Their technical specifications are displayed in tables 5.40 and 5.41.

Name	1N4148W-7-F Datasheet
Type	Small signal diode
Forward voltage	1 V
Maximum forward continuous current	0.3 A
Reverse recovery time	4 ns
Package	SOD-123
Price	0.15 € (Mouser)

Table 5.40 – 1N4148W technical specifications

Name	Si1032R Datasheet
Type	N-Channel MOSFET
Maximum continuous drain current	200 mA
Maximum $R_{\text{DS(on)}}$	10 Ω
$\text{V}_{\text{GS(th)}}$	0.7 V
Switching time	35 ns
Package	SC-75-3
Price	0.39 € (Mouser)

Table 5.41 – Si1032R technical specifications

To summarize, the motor will rotate forward when "PWMF" is set to 0 V and "PWMB" (backwards PWM signal) is set to 3.3 V, it will rotate backward when "PWMF" is set to 3.3 V and "PWMB" is set to 0 V and will be off when both input pins are set 3.3 V. Just as it was mentioned previously, all the bridge's transistors must not be on at the same time to avoid short-circuiting both branches of the H-bridge, this would happen if both input pins were set to 0 V at the same time.

Moreover, if we analyze the boat's original H-bridge design (figure 5.21), we notice the presence of four diodes in parallel with the transistors and a few capacitors. These are necessary due to transient behavior of the motor when it starts and stops working. As we explained in section 3.5.1, the voltage equation of the motor is:

$$E_a = V - IR \tag{5.2.22}$$

When the motor is not moving, the EMF, which is proportional to the rotational speed of the motor, will be 0 V. Then, when the motor is started, its current will increase to:

$$I = \frac{V}{R} = \frac{6\ V}{0.51\ \Omega} = \ 11.76\ A \tag{5.2.23}$$

Once the motor starts moving, an EMF will appear, which regulates the speed and the current so their values are lower and constant. Sometimes, the starting current of the motor can be too high, leading to damages to the motor. For this reason, there are different types of starting methods that limit the current. In this case, as it was mentioned before, the method that has been used is placing capacitors between the terminals of the motor. They absorb current when charging, which slows the step response of the current and softens its peak, so the equation 5.2.23 is not valid since this was not taken into consideration.

When the motor is suddenly disconnected, the inductance of the motor will oppose the sudden change of current by generating an opposite sign EMF. This will make the current “slowly” decrease and eventually become negative until the motor stops moving, which ceases the EMF and the current. This negative current could damage the IRL3803 transistors, which is the reason the 1N5822 diodes are placed. These will provide the current a path to the battery, slightly charging it. This concept is called regenerative braking and it is one of the reasons why the motor is connected directly to the battery.

To obtain the exact transient behavior of the motor, it will be experimentally tested. Using the N6705A power analyzer, 6 V will be supplied to the motor and its starting current will be measured (figure 5.23). The time it takes for the current to stabilize is 58 ms approximately.

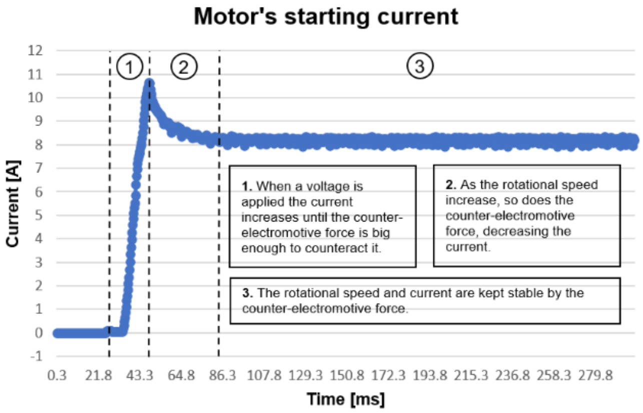


Figure 5.23 – DC motor starting current

Furthermore, the stopping current will be measured (figure 5.24), obtaining that it takes to reach 27 ms to reach 0 A. The minimum pulse width of the PWM input signal can be approximated by adding the motor's starting and the stopping time, resulting in 85 ms.

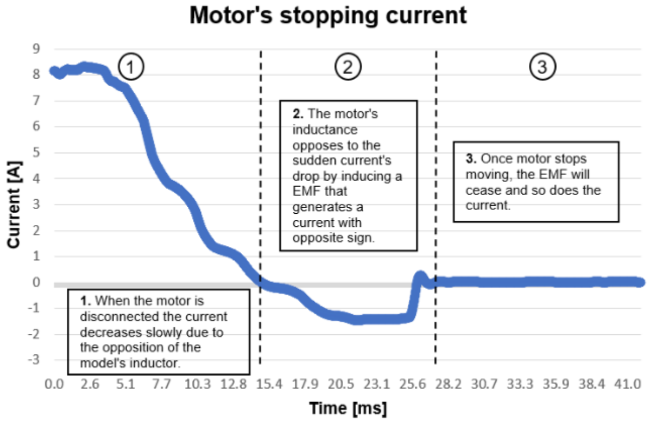


Figure 5.24 – DC motor stopping current

With this information we can now simulate the circuit in LTSpice®, obtaining the following results, where MT+ and MT- are the positive and negative terminals of the motor:

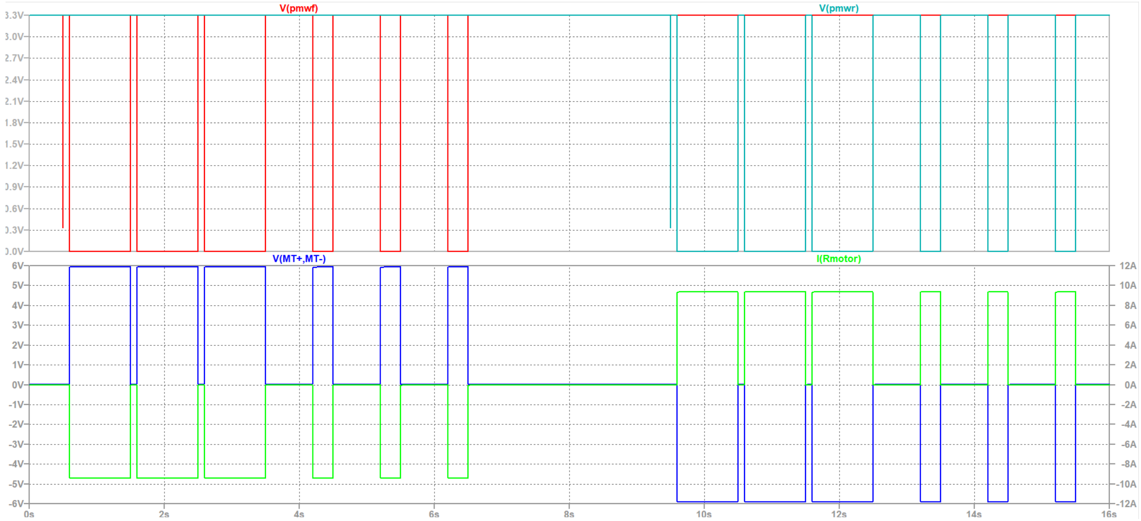


Figure 5.25 – H-bridge simulation of normal operation

For this simulation, firstly, the motor will rotate forward (the current sign is negative because of the sign convention of the circuit but it actually flows from MT+ to MT-), its operation has been tested for two different duty cycles and then the motor is turned off for three seconds. After that, the motor will rotate backwards with two different duty cycles and then it is turned off. The operation of the motor is correct as it reaches the desired current of 9.5 A (for the simulation, the motor has been modeled as a 0.63 Ω resistor to have the same current that was obtained in the tests) and the voltage across its terminals V(MT+,MT-) reaches 6 V.

This was the ideal case, when the PWM signals are perfectly programmed, which could not always be the case during the firmware development process. For that reason, it is important to check what would happen if both inputs are set to 0 V at the same time and the branches of the bridge are both short-circuited:

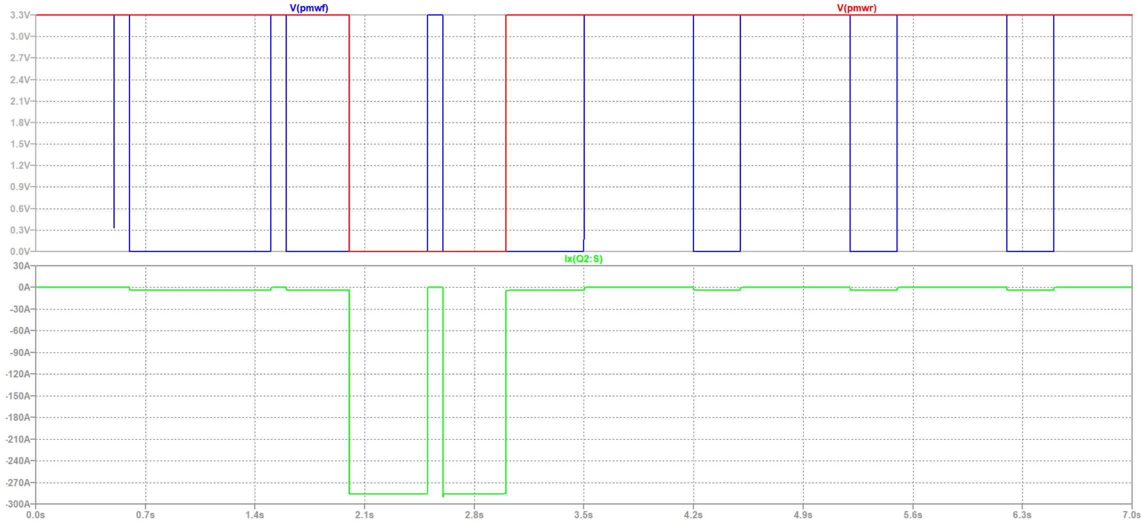


Figure 5.26 – H-bridge simulation of an incorrect operation

When both PWM inputs are 0 V both branches are short-circuited causing the current to increase greatly, specifically to 285 A. According to the data sheet, if this happens for a very small period of time, the transistor will resist this since its maximum pulsed drain current is 470 A. Nevertheless, if this current persists for a longer period of time, the transistor will burn as the maximum continuous drain current is 140 A.

To avoid damaging the circuit during the development and testing of the firmware, a resistor will be placed between the battery output and the drains of the high side transistors. This way, if there is an error, the short circuit current of the branches will be limited by the resistor to a certain value. This resistor will be removed once the firmware has been proved to work correctly. In addition, it has to be taken into consideration that, when doing the testing, the bridge load will not be the motor, instead a resistor will be used so the current that flows through the bridge is lower enough for the aforementioned resistor to dissipate the power.

If a 470 Ω resistor is placed between the battery and the drains of the high side transistors, then the short circuit current will be limited to:

$$I_{\text{load}} = \frac{V_{\text{bat}}}{R_{\text{test}}} = \frac{6\ V}{470\ \Omega} = 12.60\ mA \tag{5.2.24}$$

If another 470 Ω resistor is used as the H-bridge's load, then the load current during the correct operation of the bridge will be:

$$I_{\text{load}} = \frac{V_{\text{bat}}}{R_{\text{test}} + R_{\text{load}}} = \frac{6\ V}{470\ \Omega + 470\ \Omega} = \ 6.38\ mA \tag{5.2.25}$$

These currents are small enough to avoid using power resistors and allow the validation of the circuit and the firmware with an oscilloscope.

As it was mentioned before, the motor will be off when both PWM signals are 3.3 V. To avoid undesired states, the pins that will be used to control the motor have to be carefully selected so their they are internally pulled-up after reset. After analyzing the IO_MUX table of the ESP32 datasheet, GPIOs 14 and 15 have been selected as they fulfill this condition.

The total cost of the H-bridge with the high side drivers is calculated in table 5.42.

Component	**Price per unit [€]**
10 kΩ 0402 resistor x2	0.12 (Mouser)
1 kΩ 0402 resistor x 4	0.14 (Mouser)
200 Ω 0603 resistor x 2	0.14 (Mouser)
Si1032R x 2	0.39 (Mouser)
1N4148W-7-F x 2	0.15 (Mouser)
IRL3803STRLPBF x 6	2.35 (Mouser)
1N5822 x 4	0.41 (Mouser)
22 nF 0603 capacitor x 4	0.11 (Mouser)
1 μF 0603 capacitor x 2	0.09 (Mouser)
2.2 μF 0603 capacitor x 2	0.11 (Mouser)
Total cost [€]	18.74

Table 5.42 – H-bridge circuit cost

5.2.13 Voltage regulators

In order to adapt the battery's voltage to the operating voltage of the different modules we will implement voltage regulators. These circuits provide a constant output voltage regardless of the load conditions or the input voltage (within a certain range).

The supply voltages that are required are: 3.3 V for the ESP32, the CH340C, the DS1302, the INA219, the Adafruit 10-DOF and the SSD1306 screen and 4 V for the SIM800L and the GY-NEO6MV2. The rest of the circuit is supplied at 6 V. This differs from the previous thesis' considerations, where a 5 V voltage regulator was introduced to supply the servo motor and the navigation LEDs. This regulator will be removed because we have previously checked experimentally that the servo motor can be supplied at 6 V.

Regarding the regulators' choice, we must consider the different types of regulators and see which one fulfills our project's requirements best:

- Linear regulators: they are implemented with a voltage-controlled current source, an output voltage monitor and a control circuit. The latter regulates the current source to achieve a stable voltage at the output regardless of the load variations. The output current limit of this devices will be defined by the current source [43]. Depending on their dropout voltage they can be classified as: Standard linear regulators (1.5 to 2.2 V), Quasi-Low-Dropout Regulator (LDO) (1.5 V maximum) and LDO (0.8 V maximum).

  The advantages of this type are their insignificant output voltage ripple, their low cost and fast response time to load changes [31] while the disadvantages are a larger package size, which often requires some kind of heatsink, and a low efficiency.

- Switching regulators: they use a high frequency switching element to transform the incoming power supply into a pulsed voltage, which is then smoothed using capacitors, inductors, and other elements [40]. In general terms, this type can be classified based on whether the output voltage is higher or lower than the input: Boost (step-up), Buck (step-down), Buck/Boost (step-down/up).

  The advantages of this type are their high efficiency, smaller size, low heat generation and lower dropout voltage while the disadvantages are their higher price, intricate design, higher voltage ripple and noise caused by EMI.

Moreover, it is required to estimate the output current required for each regulator. This will be achieved by adding the consumption of each component (tables 5.43 and 5.44).

3.3 V Component	Typical current [mA]	Maximum current [mA]
ESP32	68	500
CH340C	12	30
DS1302	0.425	1.28
INA219	1	1
Adafruit 10-DOF	6.21	7.21
SSD1306	0.150	0.150
Total	87.76	539.64

Table 5.43 – 3.3 V supplied components' power consumption

4 V Component	Typical current [mA]	Maximum current [mA]
SIM800L	125	2000
GY-NEO6MV2	51.85	74.75
Total	176.85	2074.75

Table 5.44 – 4 V supplied components' power consumption

In projects such as ours, where the circuit is supplied from a battery, the most determining factor when choosing the voltage regulator is the efficiency. For this reason, the switching regulators were selected in last year thesis, specifically a Buck regulator. It was decided to use regulator modules to simplify the design process and some of the most common ones of this type are the XL1509 and LM2596.

Name	XL1509 Module IC's datasheet	LM2596 Module IC's datasheet
Type	Switching Buck Regulator	Switching Buck Regulator
Input voltage	4.5 to 40 V	4.5 to 36 V
Output voltage	1.23 to 30 V	1.27 to 37 V
Output current	2 A (max)	3 A (max)
Dropout voltage	1.5 V	1.3 A
Efficiency	74% (adjustable version)	73% (adjustable version)
Dimensions	22.5 x 16.5 x 4 mm^3	43 x 21 x 14 mm^3
Price	0.67 € (AliExpress)	2.99 € (Amazon)
Choice	✓	✗

Table 5.45 – Voltage regulators models trade off

Both models have very similar technical characteristics, however, XL1509 module was chosen due to its lower price and smaller size and it was already available in our laboratory.

Besides this, one more observation was made: the maximum peak current of the 4 V components is 2.075 A but the maximum output current of the XL1509 is 2 A. This should not be a problem because the difference is very small and it is likely that the XL1509 can reach slightly higher values than 2 A (especially if it is a peak current). However, the SIM800L module is very sensitive regarding its supply current so we will include another 4 V regulator just in case the XL1509 does not provide enough current.

This other regulator will be a linear one, which will allow us to compare the different type regulators' performance experimentally and maybe make changes in newer prototypes. The regulator that will be used is the LM317 because it is available in our laboratory, whose specifications are listed in table 5.46.

Name	LM317 Datasheet
Type	Linear Regulator
Input voltage	4.25 to 40 V
Output voltage	1.25 to 37 V
Output current	2.2 A
Dropout voltage	0.8 V
Package	TO-220-3
Price	0.68 € (Mouser)

Table 5.46 – LM317 specifications

The manufacturer's suggested circuits will be implemented, adapting the resistors' values according to equation 5.2.26 to achieve an 4 V output. Moreover, a 0.1 μF input bypass capacitor was added to improve its stability and a 1 μF output capacitor to improve the transient response.

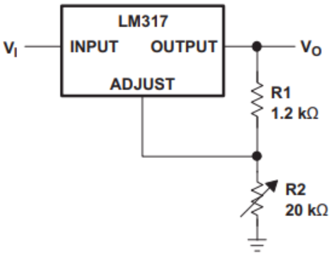


Figure 5.27 – LM317 suggested circuit [45]

$$V_{out\ LM317} = 1.25 \cdot (1 + \frac{R2}{R1}) = 1.25 \cdot (1 + \frac{2.2\ k\Omega}{1\ k\Omega}) = 4\ V \tag{5.2.26}$$

The final regulators circuit can be consulted in sheet 16 of Appendix D.

Component	**Price per unit [€]**
XL1509 module x2	0.67 (AliExpress)
LM317 x1	0.68 (Mouser)
1 kΩ 0603 resistor x 1	0.14 (Mouser)
2.2 kΩ 0603 resistor x 1	0.09 (Mouser)
1 μF 0603 capacitor x 1	0.09 (Mouser)
0.1 μF 0603 capacitor x1	0.09 (Mouser)
Total cost [€]	2.43

Table 5.47 – Regulators circuits costs

Now that the voltage regulator have been selected, it is necessary to estimate their input current, which in the case of switching regulators does not coincide with the output one.

In the case of the XL1509 module, to estimate the input current, the efficiency equation will be employed, where η refers to the efficiency.

$$\eta = \frac{P_{in}}{P_{out}}; I_{in} \cdot V_{in} = \eta \cdot I_{out} \cdot V_{out}; I_{in} = \frac{\eta \cdot I_{out} \cdot V_{out}}{V_{in}} \tag{5.2.27}$$

In the case of the 3.3 V and 4 V adjusted XL1509 modules, the output typical and maximum currents were calculated in tables 5.43 and 5.44. Then, using equation 5.2.28, the typical and maximum input currents of each regulator will be obtained and represented in table 5.48.The input voltage from the battery will be approximated to 6.5 and the efficiency will be 74% as shown in table 5.45.

I$_{in}$ [mA]	V$_{\text{in}}$ [V]	I$_{\text{out}}$ [mA]	V$_{\text{out}}$ [V]
32.97 (typical)	6.5	87.76 (typical)	3.3
202.74 (maximum)	6.5	539.64 (maximum)	3.3
80.53 (typical)	6.5	176.85 (typical)	4
944.81 (maximum)	6.5	2074.75 (maximum)	4

Table 5.48 – XL1509 input currents estimation

For the LM317, the input current will be almost the same as the output one since it is a linear regulator.

5.2.14 Battery duration

In view of the current consumption estimated throughout this chapter, we can now calculate an approximation of the battery duration. This can be calculated as:

$$time = \frac{C_{bat}}{I_{bat}} \tag{5.2.28}$$

The current consumption of the whole circuit can be calculated by adding the input current of the regulators and the rest of components. This is obtained in table 5.49:

3.3 V Component	Typical current [mA]	Maximum current [mA]
3.3 V regulator	32.97	202.74
4 V regulator	80.53	944.81
DC motor	9500	10000
Servo motor	75	220
LEDs	59	59
Total	9747.5	11426.55

Table 5.49 – Current consumption of the full circuit

When calculating the battery duration, the typical current will be used as it is more representative of the normal performance of the boat and the maximum current includes peak currents, which only happen for a very short time.

$$time = \frac{C_{bat}}{I_{bat}} = \frac{7\ Ah}{9.75\ A} = 0.72\ h \tag{5.2.29}$$

The approximated duration of the battery in this conditions is 43 minutes, which is quite short for our purpose. However, the most influential component in the battery consumption is the DC motor, whose average current can be reduced with the H-bridge. By reducing its current to a half (4.75 A), the battery duration can be extended to 1 hour and 24 minutes:

$$time = \frac{C_{bat}}{I_{bat}} = \frac{7\ Ah}{5\ A} = 1.4\ h \tag{5.2.30}$$

5.3 PCB design

In the previous thesis, a full PCB design was proposed. However, due to the amount of changes that have been made in the schematics, the PCB will be redesigned, just keeping the old one's shape. This will be a multi-board project composed of two different boards that fit together and can be attached to the battery's support through straps.

5.3.1 Shape and arrangement inside the boat

As the original boat's circuitry has been removed, there is enough free space to place one or various PCBs inside the boat, nevertheless, they must fulfill certain conditions:

- The circuit must be easily accessible to the user, especially the parts that contains components such as the rotary encoder or the buttons that are meant to be manipulated.
- It must allow an easy access and removal of the battery in case it breaks and has to be replaced.
- It has to be stable and fixed to the boat's structure so it does not move nor suffer damages by hitting the walls. This is especially important because the boat will constantly swing and is at risk of capsizing.
- It must have rounded corners to avoid cuts and hurts to the user while manipulating it.

The boat included a battery support, which will be used to keep the battery fixed in its place. This support is attached to the boat's structure with screws and the battery is attached to it with velcro straps.

This straps can be easily unfastened, allowing the user to remove the battery. Nevertheless, in order to have access to the straw, the PCB cannot be on top of the battery, which limits the available space notably.

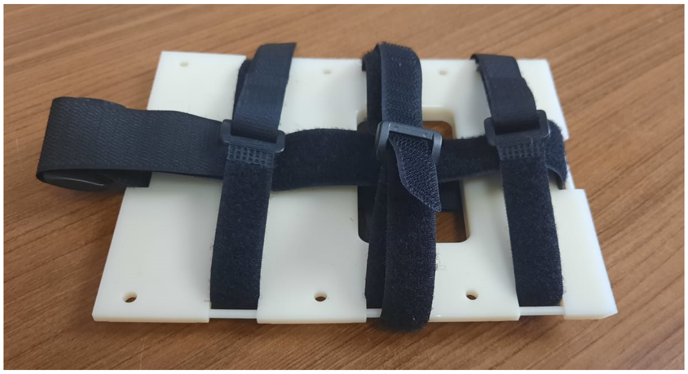


Figure 5.28 – Boat's battery support

Given these conditions, it was decided in the previous thesis that our board would be attached to the boat's battery support through the same screws that hold it to the boat's structure. This option is the most reliable and simple since it takes advantage of the original boat's structure, which has proven to work. However, using this support requires a vertically placed PCB with some kind of horizontal base that allows it to be attached to the battery support, as shown in figure 5.29. This would result in a very particular L-shape PCB.

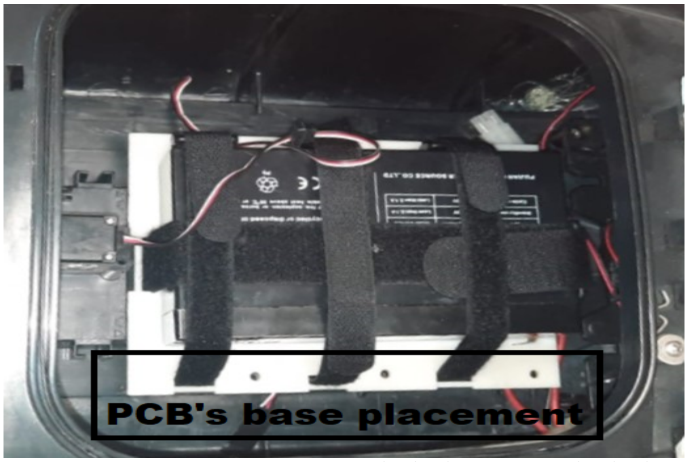


Figure 5.29 – Battery support with the battery inside the boat

This will be achieved by joining two boards: an a mount one that provides fixation and only contains the battery's and the charger's headers, and a main one that contains the whole circuitry. These boards will be joined by tabs that fit in the other board's holes. Moreover, pads will be added on the tabs and around the holes to connect both boards electrically and to provide an stronger fixation (figure 5.31). In addition, in case that fixation was too weak, we have included two slots that allows a permanent and highly reliable joint through a metallic piece (figure 5.30). This last part will only be included if it is strictly necessary.

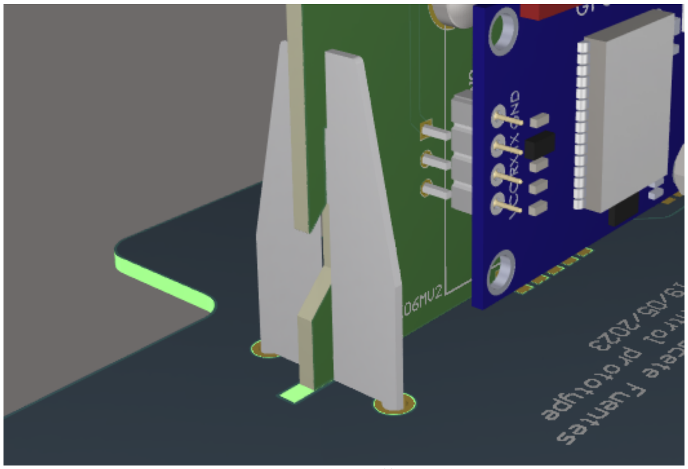


Figure 5.30 – Metallic joint piece

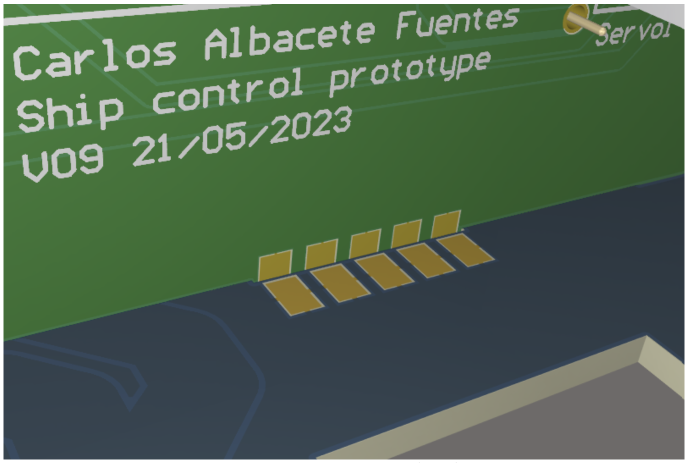


Figure 5.31 – Pad tabs

Both the mount and the main board shapes are displayed in figures 5.32 and 5.33 and its dimensions can be consulted in figures

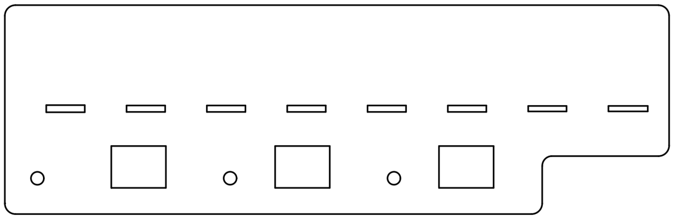


Figure 5.32 – Mount board shape

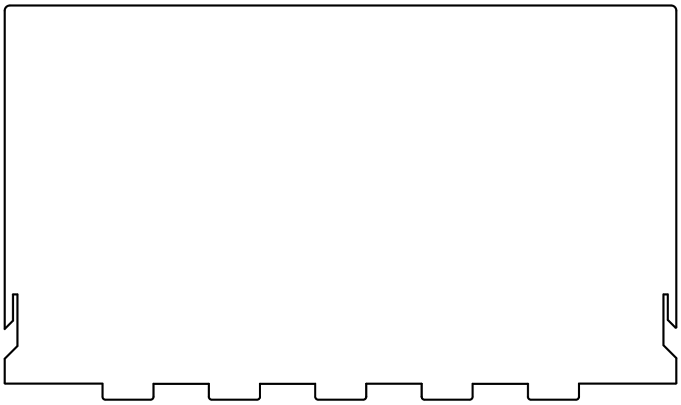


Figure 5.33 – Main board shape

5.3.2 Components' placement

When designing the PCB, it is important to take into consideration the position and orientation of the board as well as the boat's components that will be connected to it. These considerations are:

- All the headers that allow connection between the board and the boat's components must be placed in the side of the board that is facing the battery. The reason is that this side is the only one that can be accessed by the user, who would have to disconnect and connect the components in case the board was removed. Moreover, this side is closer to the servo motor, DC motor and battery connectors.

- The screw connectors must be placed so that they can be tighten or untighten without removing the battery nor the board from the inside of the boat. This leaves us two options: placing them on the mount board or placing them on the higher half of the main board, which would not be covered by the battery.

  Finally, the battery and charger headers were placed on the mount board because they are closer to the battery terminals, and the DC motor header was placed on the main board, on the DC motor side to avoid undesirable curving of the cables.

- The GPS must have access to a clear and full view of the sky and must be placed parallel to the geographical horizon [46], this will allow it to receive signals from as many satellites as possible.

  The ideal placement would be on top of the boat since the antenna would not be blocked by any material, however, this is not possible due to the small length of the antenna cable and the possibility of contact with water, that would damage it. For these reasons, it will be placed on the mount board facing upwards to have access to the sky. The fact that the antenna is covered by the boat's shell is not a problem as it is made of ABS plastic, which can be penetrated by the GPS signals [18].

  It is convenient to mention that patch antennas such as these, are not soldered into the PCB, instead, they must be attached to it with some kind of adhesive such as silicone or double-sided tape. Moreover, it should not be placed on top of any power plane as this could interfere its performance.

- The ESP32 contains an on-board Wi-Fi and Bluetooth antenna that will not be used in this project. Nevertheless, it must me operational in case that new functionalities are added. According to the ESP32 hardware design guidelines, the antenna should be outside the board, keeping the 15 mm area around it clean.

- The OLED screen must be visible from the boat's opening when the board is inside. Thus, the screen will be on top of the main board, facing the boat's opening, and connected through a curved header.

- The rotary encoder, buttons and USB connector must be easily manageable by the user when the board is inside the boat. For that reason, these components will be places on the top part of the PCB, as close as possible to the boat's opening.

The position of the rest of the components is irrelevant so their location will be determined by the tracks, aiming for the most direct connections. The PCB 3D model can be consulted in Appendix F.

5.3.3 Routing

The first decision that must be taken when starting the routing place is how many layers the board will have. Since this PCB is relatively large for the amount of components it has, a 2-layer board is the best option as there is enough free space for traces and using more layers would raise the fabrication price and time.

Then, all the components will be connected by traces. The main factor when working with traces is the width, which determines their maximum current. For most of the circuit the current will be rather low, however, the parts that are connected to the DC motor: the H-bridge and the battery, can reach average currents around 9.5 A at most, which is a considerably high current. In table 5.50, the maximum current is calculated for different widths using the typical copper thickness (0.035 mm):

Width [mm]	Maximum current [A]
0.2	0.5
0.5	2
1.5	4
2	5
4	10

Table 5.50 – Maximum current for different trace widths [28]

The minimum trace width of our PCB will be 0.75 mm, this is enough to handle the maximum current peak of the low power section of the board (2 A consumed by the SIM800L). The high power traces will be 4 mm, whose maximum current is 10 A, almost the same as the DC motor current, which would lead to overheating. Nevertheless, this would only be a problem if our DC motor operated at its maximum power, but with the H-bridge we can reduce the average current consumption to a half part, avoiding overheating.

The vias are another important factor to consider. They are holes that communicate different layers, and their dimensions must be calculated too. These calculations will be carried out in the Saturn freeware and depend on the via height, which is 1.6 mm in this case (an standardized value), the hole diameter and the plating thickness, which depends on the manufacturer. In Granasat, PCBs are manufactured by JLCPCB, that uses a via plating thickness of 18 μm. The results for a 20ºC maximum temperature raise are:

Hole diameter [mm]	Maximum current [A]
0.7	2.8
2	5.1
6.4	10.1

Table 5.51 – Maximum current for different via diameters

For the high power traces, using a 6.4 mm diameter over complicates the routing due to the lack of space. However, if two 2 mm vias are used in the same trace they will handle the same current and more space will be saved. For the rest of the circuit, 0.7 mm vias will be used as they can handle currents higher than 2 A.

In addition, when designing a PCB there are certain design rules established by the manufacturer. In our case, JLCPCB rules can be be found in its official website and those will be the ones that we will implement.

The unfulfillment of these rules could lead to errors in the PCB fabrication so it is important to carry out a design rule check to avoid problems.

5.3.4 Additional details

The following details are not crucial but they improve the overall appearance of the PCB, making it look more professional.

- Granasat's logo: it can be introduced in the PCB as a BMP image. From a marketing point of view, it is essential as it allows the users to identify the manufacturer.

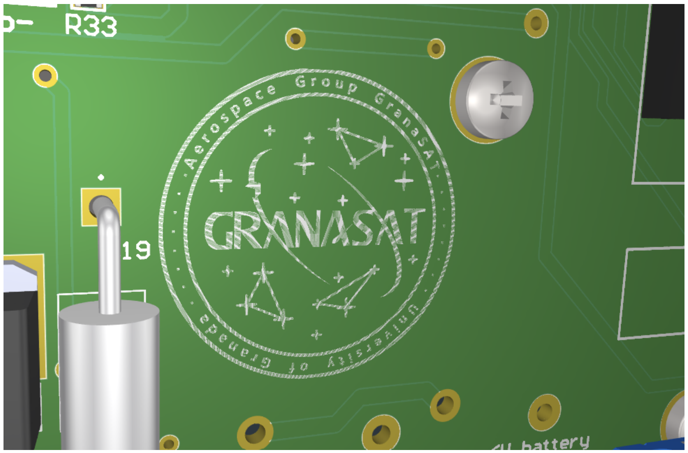


Figure 5.34 – Granasat logo in the PCB

- Prototype information: since this is not a finished product, it is recommended to indicate info such as the prototype version, the date and the author to differentiate it from further versions.

- Blank rectangle: it is usually added to prototypes to write names or notes that identify the different boards. This is especially useful during the assembly and validation stage.

- Order number: when ordering boards from JLCPCB, the order number is always added to differentiate them from other customers' boards unless an extra charge is paid. We can indicate the order number location by adding "JLCJLCJLCJLC" in the silk layer so it does not interfere with other texts of the PCBs.

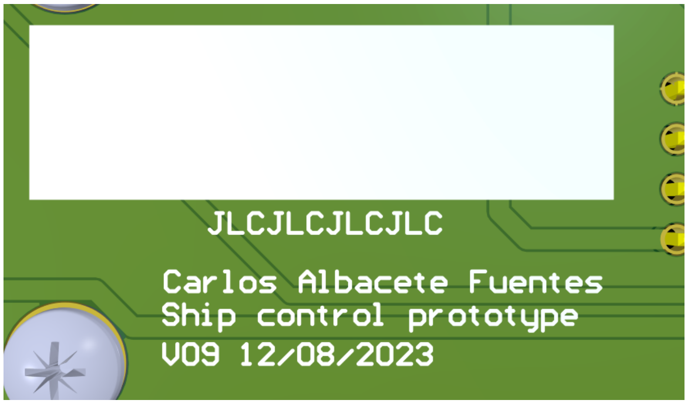


Figure 5.35 – PCB info, blank rectangle and order number indication

5.3.5 Assembly and validation

The minimum amount of boards that can be ordered from JLCPCB are five. This is convenient because it is common to make some mistakes in the assembly process, some of them leading to permanent damage. Normally, to place the components a stencil is used to apply solder paste to the PCB and then heated following a certain thermal curve. Nevertheless, ordering a stencil is much more expensive and not very practical, since this is just a prototype. For that reason, all the components will be placed manually using a soldering iron, which draws out the process and makes it more susceptible to mistakes and short-circuits.

To minimize damage risks when assembling and testing this circuit for the first time, the process will be divided in the different parts. Firstly, the ESP32, the USB to UART and the 3.3 V regulator circuits will be assembled because it is the most simplified version of the circuit that can be tested. Once that part has proven to work, the power switch circuit can be added with a 6 V power supply and then, all the different modules can be added one by one until the whole circuit works.

When testing the ESP32 programming at the beginning, an unknown error appeared. After searching for information and re-soldering various components we noticed that the error was caused by the LEDs connected to GPIO6. Apparently, the drawn current was higher than the expected 10 mA, which caused problems to the internal ESP32 flash, where the programs are stored. To fix this, 47 Ω resistors were added in series with the LEDs, which limit the current to:

$$I = \frac{3.3\ V - V_{forward}}{47\ \Omega} = \frac{3.3\ V - 3.25\ V}{47\ \Omega} = 1.06\ mA \tag{5.3.1}$$

The brightness is lower but it is enough for the user to see and for the ESP32 to be programmed correctly.

One problem that was noticed after the board was manufactured is that the Adafruit 10-DOF footprint that was used had the order of the pins reversed. Luckily, this could be resolved by turning around the module.

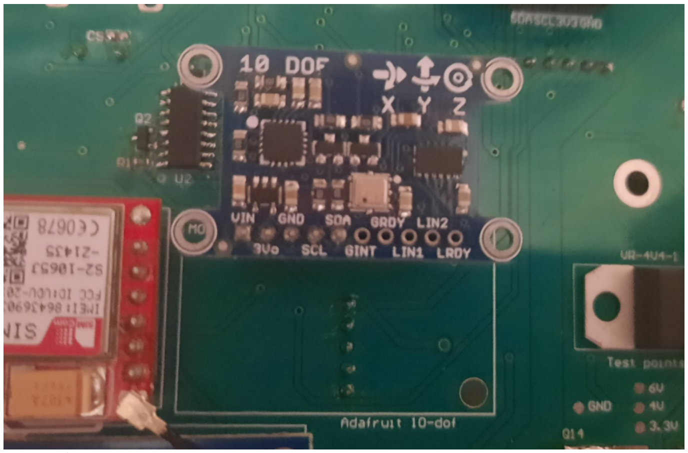


Figure 5.36 – Adafruit 10-DOF module placement, it does not match the footprint

Moreover, the charger circuit operation was not as expected. It will require changes for the following prototypes but it is not a deal-breaker as the battery can be removed and charged with an external charger. This will be explained in more detail in section 5.4.8.

In addition, when testing the SIM800L module when supplying it with the XL1509 module, its operation was not correct. This happens because this voltage regulator cannot supply enough current to reach the required current peaks, for this reason it will be discarded for further prototypes. Nevertheless, with the LM317 the operation was correct so it will be kept on the board.

Apart from that, the rest of the circuit worked as expected. Only minor changes have been made, mostly replacing some resistor and capacitor values with similar values because the exact ones were not available in our laboratory.

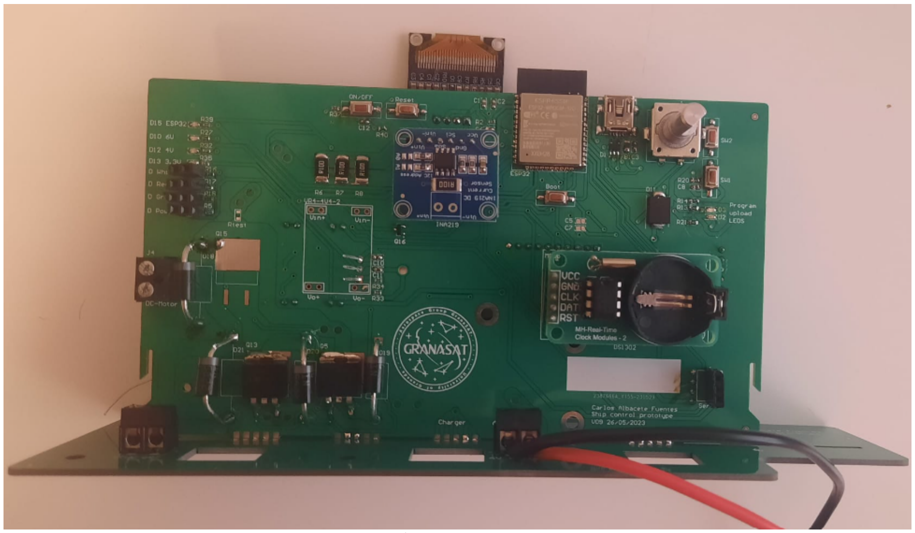


Figure 5.37 – Assembled PCB front view

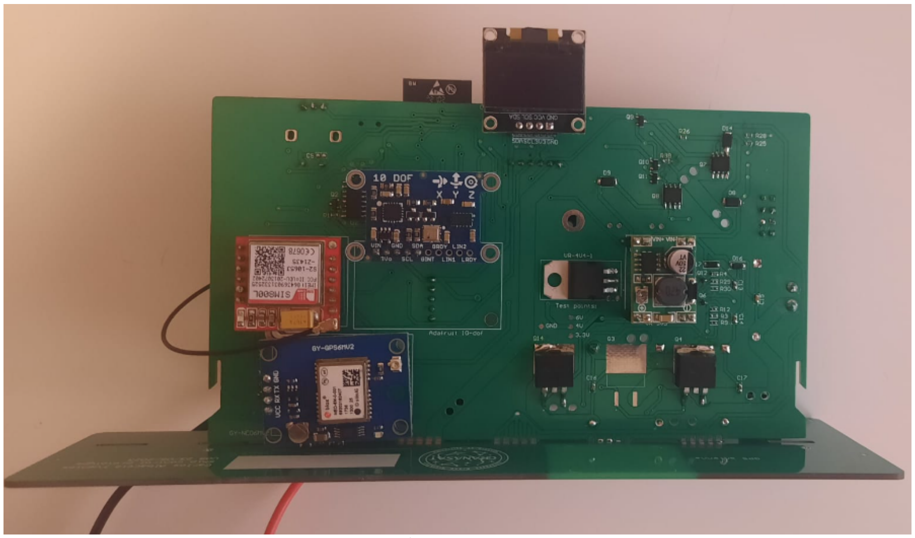


Figure 5.38 – Assembled PCB rear view

5.4 Firmware design

Once the PCB is complete, we can test and program the full circuit. Regarding the previous thesis' firmware, the development made is almost negligible, thus we will implement a design from scratch. For each part of the circuit, a firmware sketch will be developed to distribute and facilitate the verification process and then, all the different sketches will be grouped into a global sketch with full functionality.

The firmware will be developed in the Arduino framework due to its compatibility with the ESP32, our familiarization with it and the C++ language acquired in our degree and the amount of available libraries that facilitates the programming of the different modules.

5.4.1 Power switch

This part of the code is the one that has to be executed first because without it the board could not be turned on unless it was connected to a PC though an USB. This program is simple, it just has to set the "ON/OFF" pin to 3.3 V as soon as the program starts and keep it that way until the ON/OFF button is pressed. When this happens, an interruption will stop the program execution and set the "ON/OFF" pin to 0 V, turning the board off. This interruption activates when a rising edge is detected in the "ON/OFF button detector" pin.

It is necessary to add a small delay (200 ms approximately) to the first push of the button is not detected and the boat does not turn off as soon as it is turned on. Moreover, it is worth mentioning that every GPIO of the ESP32 supports external interruptions. This code is available and commented in detail in Appendix C.1.

5.4.2 Rotary encoder

A generic rotary encoder code will be developed so it can be modified and implemented according to our needs. Firstly, the rotary GPIOs must be configured as input pull-ups since we have removed the external pull-up resistors.

Then, this code constantly checks the state of the encoder and compares it with the previous state through a loop. It detects that the encoder is moving when the previous state of one of the pins is different that the current one, when this happens, it compares the state of one of the pins with the other, if the state of the B pin is different than the A pin's then the direction is clockwise and the other way around. This can be seen more clearly in figure 5.10.

Regarding the rotary button, it works as a normal button, so we can detect if it is pressed by checking if the "Rotary Button" pin is at 0 V. The full code is available and commented in detail in Appendix C.2.

5.4.3 INA219

To facilitate the programming of this module, the Adafruit INA219 library will be used, which, in turn, requires the Arduino's Wire.h library to communicate through I2C. This code creates an instance of the INA219 object, without specifying its I2C address because we are using the default one (0x40). Then, the I2C communication will be initialized and, if this step is successful, we can use the sensor reading functions. The variables that can be read with this functions are:

- The bus voltage: defined as the voltage between the "INA219-" port and ground.
- The shunt voltage: defined as the voltage between the "INA219-" and "INA219+" ports.
- The current across the default 0.1 Ω shunt resistor, in our case it has to be multiplied by four as we are using four 0.1 Ω resistors in parallel.

The battery voltage can be obtained as the bus voltage minus the shunt voltage, The full basic

INA219 code, commented in detail, can be consulted in Appendix C.3.

5.4.4 SSD1306 OLED

The Adafruit SSD1306 and the Adafruit GFX library are provided by the manufacturer to facilitate the programming of the screen. They provide a set of graphic functions for multiple screen models and abstract the user from the I2C set of commands.

These libraries require to create an instance of the OLED display indicating the screen dimensions and initialize the screen, in this case with the screen's default I2C address 0x3C. To print a simple string, the text size, the color and the starting pixel must be indicated. The full code is commented and available in Appendix C.4.

5.4.5 GY-NEO6MV2

In order to test the operation of the GPS module, a code that obtains the latitude and longitude coordinates and displays them on the OLED screen will be implemented. When the module starts receiving signals from satellites, it send GPS data to the ESP32 through the UART1 port. This data is NMEA-formatted, so a library is required to decode it and obtain the coordinates in decimal degrees. For this purpose, the TinyGPSPlus library by Mikal Hart has been chosen as it is simple and effective.

In this code we have to create the screen and the TinyGPSPlus instances and initialize them. Then, the raw NMEA data will be read from UART1 and decoded using one of the GPS library's methods. If new data is received, which is indicated through a flag, then we will check if this data is valid using one of the library's methods and display it in the OLED screen with a six decimal digits precision. The full code is available and commented in detail in Appendix C.5.

When testing this code, it is important to mention that it may take some time (from a few seconds to minutes) to work since the GPS module has to get a satellite lock.

5.4.6 Wireless communication

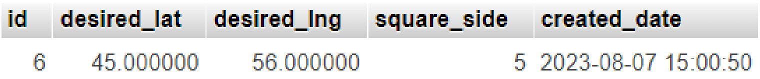
Establishing a remote connection between the SIM800L and a mobile app is one of the most crucial requirements for our project. There are different options to achieve this, however, using a direct communication between two devices is not recommended since it is likely that the boat's IP changes while it navigates, so we would not be able to send data to the boat while it moves.

For this reason, a server will be created. It will act as a broker that handles the messages that are sent by the user and the board. The user will introduce the desired coordinates and the side of the square that the boat will describe around that location in a mobile app, these values will be sent to the server and stored in a database, which can be read by the module. The module will send messages to the user the other way around. This way only the server's address is required to communicate.

Then, we need to develop three parts to achieve communication: the SIM800L module, the mobile app and the server. All of these parts are related and the server will contain the database, that contains two tables: one for the desired coordinates and the square side, sent by the app and read by the boat, and other for the current coordinates of the boat and warnings, sent from the boat and read by the app. Each part will be explain in more detail below.

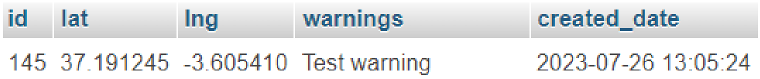
5.4.6.1 Server files and databases

To create the server where the database will be allocated, we will use 000WebHost, which is a free hosting platform that allows a website and a MySQL database. As it was mentioned, two different tables will be created in the database, the reason is to simplify the server program.



id	desired_lat	desired_lng	square_side	created_date
6	45.000000	56.000000	5	2023-08-07 15:00:50

Figure 5.39 – Database table to store the desired location and the square side (desired_location)



id	lat	lng	warnings	created_date
145	37.191245	-3.605410	Test warning	2023-07-26 13:05:24

Figure 5.40 – Database table to store the current location and the warnings (tbl_gps)

The website will be developed through various PHP files and each of them accomplishes a different task:

- config.php: defines the database parameters, sets the default timezone and connects to the database.
- add_current_loc.php: obtains the current GPS coordinates of the boat and the warnings through the $_GET method, inserts them into the figure 5.40 table and shows them in the website.
- current_loc_map.php: extracts the current GPS coordinates from their table, displays them on the website and opens the an inline frame that shows those coordinates in Google Maps.
- display_warnings.php: extracts the warning and the time it was sent from the figure 5.40 table and displays them in the website. This website will be accessed by the app.
- add_desired_loc.php: allows the app to send the desired GPS coordinates and the square side through the $_GET method. Those values are inserted into the figure 5.39 table.
- display_desired_loc.php: extracts the desired GPS coordinates and the square side from their table and displays them on the website in three lines.

Since it has been mentioned a few times, we will introduce the $_GET method: it is super global variable which is used to collect form data after submitting an HTML form with method="get" [51]. Then, in order to send data from the SIM800L and the app to the databases we just need to fill the corresponding HTML form. All the PHP codes are available and commented in detail in Appendix C.10.

5.4.6.2 SIM800L

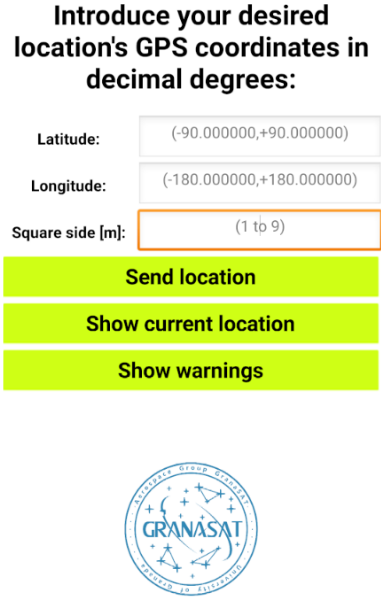
Regarding the ESP32 and SIM800L interface, this module is controlled through GSM AT commands. However, the list of these commands is very extensive, not always very intuitive and requires additional functions that read the serial data. For that reason we will use the TinyGsm library by Volodymyr Shymanskyy to simplify some processes, such as the initialization and the GPRS connection verification. In this code, we will explore the processes of sending data to the server and reading data from it through GPRS.

Firstly, the initialization process will begin: the UART2 port communication is started, the module is restarted and establishes the GPRS connection. Once this is completed, the loop execution will begin, which consists in a verification of the connection and an HTTP reading and sending test every ten seconds.

This test begins by initializing the HTTP service and sending the test coordinates through the add_current_loc.php url. Then, it will read the desired location coordinates from the display_desired_loc.php url by using the AT+HTTPREAD command. This command reads everything that is displayed in a certain url, so it requires a complex function that, knowing the website format, extracts the desired data and stores it in variables. This is achieved through the sendHTTPREAD() function, which detects the number of digits of the coordinates and extracts the desired location and the square side to store it in variables. If the reading is not successful, the program will warn the user and not display the reading. After this, the HTTP service and GPRS connection are shut. All this code is available explained in detail in Appendix C.12.

5.4.6.3 Mobile app

Since creating an app can be a challenging task and not much time is available, the MIT App Inventor website will be used. This allows the development of mobile apps online in an intuitive way and, even though it is aimed for learning purposes, it is powerful enough for our application. This tool is a block-structured language mixed with a graphical interface that emulates a phone screen where the characteristics of the different graphical elements can be edited.



Introduce your desired location's GPS coordinates in decimal degrees:

Latitude: (-90.000000,+90.000000)

Longitude: (-180.000000,+180.000000)

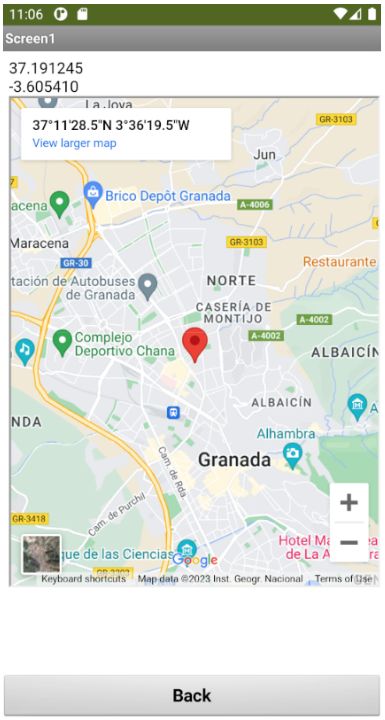
Square side [m]: (1 to 9)

Figure 5.41 – App's interface

Our app's interface is composed of a title box, three text boxes, three buttons and the Granasat's logo. In the text boxes, the latitude and longitude coordinates will be introduced taking into consideration that the latitude ranges from -90 to 90º and longitude from -180 to 180º and the square side has to be between 1 to 9 m. if these conditions are not fulfilled the app will display a warning to the user.

The "Send location" button stores the introduced coordinates and square side in their corresponding database table. These text boxes only allow to introduce numbers and they must be all filled or else the values will not be sent and a warning will be displayed.

The "show current location" button opens the aforementioned "current_loc_map.php", which opens an inline frame that shows the boat's position in Google Maps.


11:06

Screen1

37.191245
-3.605410

37°11'28.5"N 3°36'19.5"W
View larger map

Figure 5.42 – App showing the current GPS location

The "Show warnings" button opens the aforementioned "display_warning.php" website, where the latest warning and its date will be shown. This is not the most efficient way to display warnings as it requires the user to constantly check and update the website, however, due to time constraints this is the method that will be used for this prototype, which must be improved in further versions.

11:09

Screen1

Test warning
2023-07-26 13:05:24

Figure 5.43 – App showing the most recent warning

Finally, the Granasat's logo is placed at the bottom of the screen and works as a button that opens the Granasat website when pressed.

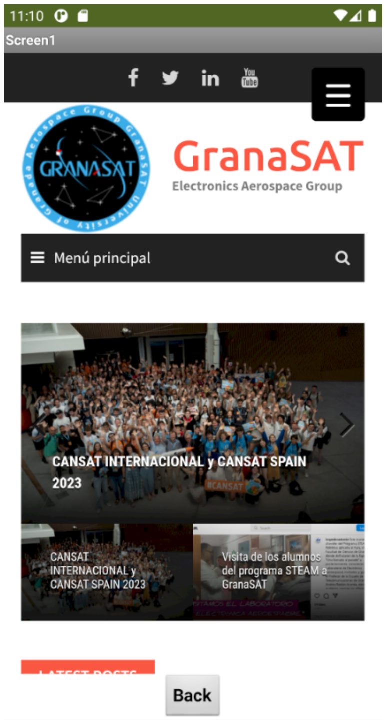

11:10
Screen1
GRANASAT
GranaSAT
Electronics Aerospace Group
Menú principal
CANSAT INTERNACIONAL y CANSAT SPAIN 2023
CANSAT INTERNACIONAL y CANSAT SPAIN 2023
Visita de los alumnos del programa STEAM a GranaSAT
LATEST POSTS
Back

Figure 5.44 – App showing the Granasat's website

The block code is available in Appendix C.11.

5.4.7 DS1302

To facilitate the programming of this module, the Michael Miller's RTC library will be used. This library works for multiple RTC modules, but in our case we must include the RtcDS1302.h and the ThreeWire.h files. This last one allows the MCU to interface with the DS1302 through the three wire synchronous serial protocol.

Firstly, it is required to create an instance of the ThreeWire method and construct a RTCDS1302 object using the ThreeWire instance. Once this is done, we will create a RtcDateTime instance from the compilation date and time using the standard predefined macros "___DATE___" and "___TIME___" and we will check the validity of the time and date and the correct operation of the module by using some of the library's methods. Then, another RtcDateTime instance will be created to store the current time and date, this instance will be updated with the compilation time in case the RTC time was older than the compilation time.

Now, the DS1302 is configured and ready to be used. In order to test it, the time and date will be displayed through the serial port every five seconds. The full DS1302 test code is displayed and commented in detail in Appendix C.6.

In the final code, depending on the month and the hour the front LED will be turned off or on to light up its path. It will be turned on when it gets dark, this can be approximated by looking the average sunset and sunrise hours of Spain each month. The light will be turned on from 9h to 18h from October to March and from 8h to 20h in the rest of the year.

5.4.8 Battery charger

Since the lead acid battery is delicate and could be easily damaged by an improper charge process, the charger circuit will be tested without the battery first. We need to ensure that we can regulate the voltage that is supplied to the battery correctly with the pass transistor, to facilitate this task we will use the rotary encoder, that will allow us to control the voltage at the "Charger" pin.

The "Charger" pin corresponds to GPIO26, which features an 8-bit DAC. This means that we can specify the analog voltage within a digital range of 0 to 255. Its resolution can be calculated as shown in equation 5.4.1, where n is the DAC's number of bits:

$$Resolution = \frac{V_{full\text{-}scale}}{2^n - 1} = \frac{3.3\ V}{2^8 - 1} = 12.94\ mV \tag{5.4.1}$$

Moreover, we need to obtain the measures of the external power source that will be used as a charger. This will be accomplished through a voltage divider connected to the "Vsense" pin, that corresponds to GPIO36. This pin features a 12-bit ADC, so it will translate an analog voltage between 0 to 3.3 V to a digital value between 0 to 4095. Knowing this, the charger voltage can be obtained as:

$$Vcharger = \frac{V_{sense\ ADC} \cdot V_{full\text{-}scale} \cdot (R_{25} + R_{28})}{(2^n - 1) \cdot R_{28}} \tag{5.4.2}$$

The voltage that is supplied to the battery will be monitored with the INA219 and displayed in the SSD1306 screen. This code, commented in detail, is shown in Appendix C.8.

5.4.9 State of charge estimation

An important feature of the boat is the capability to estimate the battery's charge so it can return back when it is close to the discharge and turn itself off before the end voltage is reached. One of the simplest methods to make this estimation is the integration of the instantaneous current, resulting in:

$$q(t) = \int_0^t i(t)dt + q_0 \tag{5.4.3}$$

This equation must be to the Arduino language, which is not designed to carry out complex operations such as integrals. However, we can approximate it by dividing the function in rectangular sections, calculating the area of each of them and adding them all, this is known as the Riemann Sum. The resulting equations, where T is the sampling rate and k is the iteration number, is as follows:

$$q(kT) = \sum_{k=1}^{n} i(kT) \cdot T + q_0 \tag{5.4.4}$$

Nevertheless, one of the problems of this method is the estimation of the initial charge q_0. In order to fix this problem, the only solution is to introduce the battery fully charged in the first boat's usage so we can assume that q_0 is 7Ah. After that, the current state of charge can be calculated and if the boat is turned off, we can store the last value in the Flash memory and use it as q_0 in the next usage.

The problem in our current prototype is that the charger does not work, so when we removed the battery to charge it we would lose track of its state of charge. To temporarily solve this, the board should only be turned on when the battery is charged to its maximum capacity so we can always assume that q_0 is 7Ah.

Once the charger circuit is corrected this will no longer be a problem.

The battery current will be measured with the INA219 and the sampling will rate will be fixed to 1 ms, which has proven to be enough to have a precise estimation. The full state of charge code is available and commented in Appendix C.13.

In the final code, if the battery capacity reaches 40% (2800 mAh) the boat will return to the location where it was released, if it reaches 25% (1750 mAh) it will send a warning to the user indicating that the boat will be turned off soon and if it reaches 20% (1400 mAh) the board will finally turn itself off to avoid damages.

5.4.10 Adafruit 10-DOF

As it was mentioned in section 5.2.4, this module is composed of 3 different ICs, however, only one of them is necessary for our project: the LSM303, which contains a three-axis magnetometer and accelerometer, enough to provide us with the pitch, roll and heading of the boat. To facilitate its programming, we will use the libraries that are suggested by the manufacturer, these are: Adafruit Unified Sensor, which works as a driver for multiple Adafruit sensors, LSM303DLHC, that contains the algorithms to transform the raw data into useful information that can be interpreted by the user and Adafruit10DOF, which contains helper functions that complement the other two libraries.

Firstly, an instance of each library will be created and an unique ID has to be assigned to each sensor, then the I2C communication is established between the sensors. Through the accelerometer, we will obtain the roll (rotation about the x-axis of the module) and the pitch (rotation about the z-axis of the module) and through the magnetometer, we will obtain the boat's heading around the y-axis of the module. It has been tested experimentally that this heading value, that ranges from 0 to 360º, minus 90º is equivalent to the compass' angle. This, combined with the roll, is what enough to detect possible capsizes and the boat's orientation. It is important to mention that the board has been positioned so the boat's box points at the same direction as the compass.

The roll angle ranges from -90 to 90º, and by rotating the board we have determined that the boat could definitely be considered capsized if this angle was smaller than 15º. If this is the case, a warning should be send the user's mobile app through the SIM800L. The Adafruit 10-DOF code is displayed in Appendix C.9.

5.4.11 Servo motor code

As we mentioned in section 3.4, the Arduino's Servo library will be used to program the servo motor. This library allows us to control its position by specifying the angle in a 0 to 180º range, however, this angle does not correspond to our rudder so the angle correspondence had to be determined experimentally, obtaining the results in table 3.5.

This means that when the servo angle is set to 0º in the code, the boat will move to starboard, when it is 145º, the boat will move to port and if it is 70º it will move straightforward. In the code in Appendix C.14.

5.4.12 DC motor code

Firstly, a very simple code will be developed to test if the H-bridge circuit works correctly. In this code, both of the PWM inputs will be set to 3.3 V for 100 ms and then one of them will be set to 0 V for 100 ms repeatedly to simulate its typical operation with a 0.5 duty cycle. Using a resistor instead of the motor and measuring across its terminals, the following voltage is obtained:

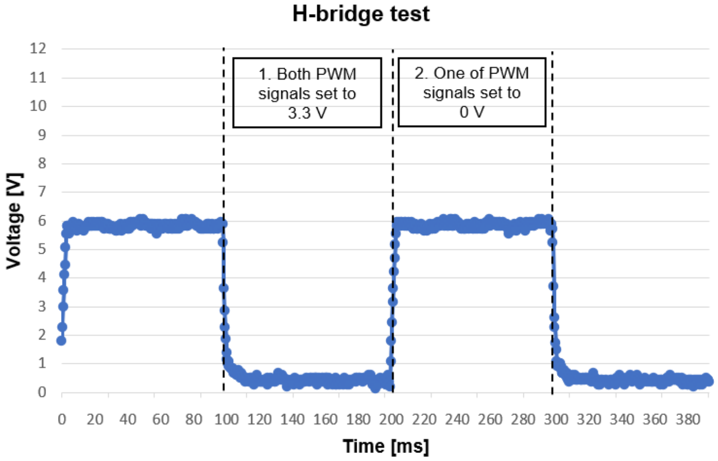


Figure 5.45 – H-bridge test with a resistor instead of the DC motor

It can be observed that the voltage switches from 0 to 6 V approximately, this means that the operation is correct. If one of the transistors did not enter saturation mode, the voltage across the terminals would not reach 6 V and the transistors would not be able to dissipate the power if the real motor was used.

Now that the H-bridge has been tested, a more sophisticated code can be developed. It will consist of a function where the direction of rotation and the duty cycle can be specified. Since the H-bridge's bootstrap capacitor discharges whenever the motor is on, the on pulse width will always be the the lowest possible (85 ms) and the off pulse width will adjust to the duty cycle.

This code is available in Appendix C.15.

5.4.13 Navigation algorithm

Once all the sensors are operational and the human-machine interface is established, the boat has access to all the data required to autonomously navigate. Following our client's indications, the navigation process must be as follows:

1. The user sends through the app the desired coordinates that the boat has to reach and the side of the square that the boat will describe around that location. This square will indicate the divers the area of interest. The user must be able to change these values at any time.

2. Once the boat receives the data, it starts moving to the desired location and as soon as it is reached, the boat will notify the user through the app and trace a square shape around it. If a new location is received, the boat will immediately recalculate the trajectory.

3. If the boat capsizes, the boat must stop moving and show a warning in the app so a diver can pick it up.

4. If the battery is below a certain point, the user will be notified and the boat will return to the location from where it was released.

From a high level standpoint, navigation algorithms are usually composed of different processes or states, whose execution is determined by the system inputs: either data provided by the sensors and the user or data calculated during the program execution. This high level control can be described and implemented through a Finite State Machine (FSM).

This first approximation of the boat's navigation algorithm consists in a thirteen states FSM. Each state will be explained in detail:

- State 1: it is the initial navigation state. In this state the boat does not move and waits for a new desired location defined by the user. If the new desired location is different from the previous desired location (which is 0º, 0º at the beginning of the program), it will switch to state 2.
- State 2: this state checks if the boat is already in the desired location by comparing it with the GPS coordinates. If the boat is in the desired location, it will calculate the time the boat requires to move from the center of the square to the center of one of the square sides and switch to state 3. If the boat is not in the desired location, it will switch to state 9.

  To calculate the time to cover the aforementioned distance, it will use half the square side provided by the user and the speed value obtained in the DC motor testing (section 3.5). Since we are going to use a 50% duty cycle, that speed will be divided by two, resulting in 0.31 m/s.

$$time = \frac{distance}{speed} \tag{5.4.5}$$

- State 3: the boat will move forward until the calculated time is reached. Then, the compass orientation that the boat has to reach to turn in a 90º angle will be calculated by subtracting 90º to the current orientation of the boat and it will switch to state 4. If the boat deviates from the desired location more than 10 m, it will switch to state 9 to correct its location, this will be done in all states from 3 to 8.
- State 4: the boat will rotate by turning the DC motor on with a low duty cycle and fully turning the rudder to one side. If the previously calculated orientation is reached, it will calculate the time to reach one of the corners of the square and will switch to state 5.
- State 5: the boat will move forward until the calculated time is reached. Then, the compass orientation that the boat has to reach to turn in a 90º angle will be calculated and it will switch to state 6.
- State 6: the boat will rotate by turning the DC motor on with a low duty cycle and fully turning the rudder to one side. If the previously calculated orientation is reached, it will calculate the time to reach the next corner of the square using the square side value and then switch to state 7.
- State 7: the boat will move forward until the previously calculated time is reached. Then, the compass orientation that the boat has to reach to turn in a 90º angle will be calculated and it will switch to state 8.
- State 8: the boat will rotate by turning the DC motor on with a low duty cycle and fully turning the rudder to one side. If the previously calculated orientation is reached, it will calculate the time to reach the next corner of the square using the square side value and then switch back to state 7. This way, the boat will describe a square until a certain event, such as: a new specified location, major deviations in the location or a low battery.
- State 9: this state is executed when the boat is not in the desired position and it will calculate the compass orientation that the boat has to reach to be able to access the desired location in a straight line and the time it will need to reach it.

  For this calculation, we will treat the GPS coordinates as points in a Cartesian plane, where latitude is the x-axis and longitude is the y-axis. We will calculate the distance between the boat and the desired location in both axis by subtracting the boat's current latitude and longitude to the desired latitude and longitude respectively. With these values, treating the boat's location as the Cartesian plane's origin, the angle between the x-axis and the desired location can be calculated as:

$$\alpha = acrtg\ (\frac{longitude_{desired} - longitude_{boat}}{latitude_{desired} - latitude_{boat}}) = acrtg\ (\frac{longitude_{distance}}{latitude_{distance}}) \tag{5.4.6}$$

To implement the *arctan* function, the C++ Math.h library must be included.

Then, that angle must be adapted to be equivalent to a compass angle. This will depend on the quadrant where the desired location is. The relationship between the latitude and longitude distances' signs and the quadrants is displayed in the following table:

Latitude distance sign	Longitude distance sign	Quadrant
Positive	Positive	First
Negative	Positive	Second
Negative	Negative	Third
Positive	Negative	Fourth

Table 5.52 – Relationship between the latitude and longitude distances' signs and the quadrants

The required compensation to make the required angle equivalent to a compass angle can be observed in table 5.53.

Quadrant	Angle compensation
First	90º -α
Second	270º-α
Third	270º-α
Fourth	90º -α

Table 5.53 – Compensation to adapt the required angle to the compass angles

This compensation allows to compare the calculated angle with the compass angle.

To calculate the time, that the boat will take to reach the desired location in a straight line. Firstly, the distance can be calculated as:

$$distance = \sqrt{longitude_{distance}{}^2 + latitude_{distance}{}^2} \tag{5.4.7}$$

Once we know the distance that the boat has to cover and the boat's speed on the water with a 50% duty cycle, we can calculate the time that the DC motor has to be on to reach that position as in equation 5.4.5. After that, it will switch to state 10.

- State 10: the boat will rotate by turning the DC motor on with a low duty cycle and fully turning the rudder to one side. If the previously calculated orientation is reached, it will switch to state 11.
- State 11: the boat will move forward until the time to reach the desired location passes. At this point, the boat will be in the desired location or close to it, so it will switch to state 2.
- State 12: this state will be executed if the boat's roll angle is lower than 15º, which means that the boat has capsized. In this state, the boat warns the user about the situation and does not move, awaiting for the boat to be righted. If that is the case, it will switch to state 2 to repeat the navigation process from the beginning.
- State 13: this state will be executed if the current consumption is to high (superior to 11 A), this can happen if the boat is stuck in an obstacle. In this case, a warning will be sent and the DC motor will be stopped to reduced the consumption. In this case, a diver should pick up the boat and restart it.
- The most recent desired location sent by the user will be constantly compared to the previous desired location. If they differ, the FSM will switch to state 2.

This algorithm may not always be precise since it relies on the DC motor test's speed, that was obtained under ideal conditions (without waves nor wind). This could be improved by obtaining the speed from the GPS tracker the moment the boat starts to move and use it to calculate the time to cover the distance.

Nevertheless, this was not implemented because it was not realized until the algorithm was finished and it was too late to change it.

To improve the algorithm's reliability, the boat's location is constantly checked and the trajectory is recalculated if the boat deviates too much from the desired location. Despite the inconveniences, this algorithm is a first practical approximation that fulfills all the client's requirements and serves as a starting point for future corrections and improvements. The algorithm will be compacted in a function called "navigation" which can be accessed in Appendix C.16.

Due to time constraints, this algorithm has not been put into practice yet. This task will be accomplished in further theses.

5.4.14 Full firmware

After developing the basic codes for each component, all of them will be combined into a single one with full functionality. Some of parts such as the rotary encoder or the buttons will not be included as they were mainly included for debugging purposes.

Firstly, all of the components will be initialized in the setup function. Then, the following processes will be carried out in the loop:

- Current check: the INA219 sensor measures the battery's current and, in case it surpasses 11 A, the navigation state machine will switch to state 13.
- Time check: the DS1302 RTC checks the time and date and, if it is night-time, turns on the front LED.
- State of charge estimation: estimates the battery percentage and acts consequently as mentioned in section 5.4.9.
- GPS read: the GY-NEO6MV2 raw NMEA data is read and converted into GPS values each second.
- App communication: checks if a GPRS connection is established, if that is the case, reads the values introduced in the app and sends the most recent current location to the app.
- Navigation algorithm: depending on the data acquired on the previous tasks, the navigation state will change from the initial one and execute the rest as explained in section 5.4.13.

The full code is explained through comments in more detail in Appendix C.16.

Chapter 6

Conclusions, future work and lessons learned

6.1 Conclusions

Throughout the development of the project, we have carried out most of the parts of the design process of a commercial electronic product: requirements, objectives and constraints definitions, reverse engineering process on an already-existing product, review and correction of a previous prototype, development, fabrication and verification of a PCB and development of the first firmware version.

There are still some tasks that could not be carried out, mainly due time constraints and lack of means, such as a complete maneuverability test and an operation test of the finished boat prototype. Other tasks were actually carried out, but their operation was not correct such as the battery charger circuit and in other cases, its operation has not been fully tested and needs further improvements, such as the navigation algorithm.

However, despite the difficulties and the tasks that were not completed, the author is highly satisfied with this project due to the great progress achieved compared to the previous thesis and the accomplishment of the most crucial and complex tasks: a correct characterization of the original boat's elements, the modelling of the DC motor, the previous schematics correction, the development of a functional high power H-bridge, the the fabrication and verification of a functional multi-board PCB, the development of a mobile app that can communicate remotely with the boat through a database and the development of a first firmware version.

As it was stated in Chapter 1, our motivation was not to finish the whole project as it would be unrealistic, but to make as much progress as we could while minimizing the design errors, providing a reliable starting point for future advances. In view of the results, we can firmly state that this objective was achieved.

Moreover, during the development of this project, the author has refined his set of engineering skills that were acquired in the degree, specially the circuit design, C++ coding, laboratory instrumentation manipulation and has acquired new ones, such as PCB development, mobile apps development, PHP and SQL coding.

In summary, even though the author is highly satisfied with the work results presented, it is obvious that this product requires further progress and improvement in order to consider its commercialization. In this thesis, all the required elements to continue the project have been presented in a detailed and rigorous way, appropriately cited when needed. Nevertheless, in case of lack of understanding, the referenced bibliography may be a good point to start.

6.2 Future work

As we mentioned in Chapter 1, this project was conceived as a long-term work that would not be completely finished during this year. While many important milestones were reached during this year, there are still other tasks that may required enough work to carry out a following thesis. These tasks are:

- Electronics:
    - To design, implement and verify a new charger circuit.
    - To remove the 4 V adjusted XL1509, leaving just the LM317.
    - To replace the SIM800L by the SIM7600E LTE CAT1 Module.
    - To replace the Adafruit 10-DOF by the BNO055.
- Software:
    - To perfect the navigation algorithm from the navigation and maneuverability tests. This algorithm can be improved by:
        - To develop a more efficient way of turning the boat an specif angle that does not require the boat to be almost stopped.
        - To calculate the time that the DC motor has to be turn on to cover a certain distance using the boat's speed provided by GPS module.
        - To develop an obstacle recognition system that detects when the boat's position does not change regardless of the efforts and recalculates an alternative path.
    - To implement the complete state of charge code taking into consideration the state of charge estimations obtained during the battery charging. This way the boat could be started without having the battery fully charged at the beginning.
    - To optimize the remote communication system, especially the reading of the desired location and square side, which is not always effective.
    - To optimize the app's warning display method and so the warnings automatically appear at the screen the moment they are received.
- Testing and qualification:
    - To carry out a maneuverability tests to achieve a better understanding of the boat's direction system.
    - To test the full system operation in a controlled aquatic environment and verify that the first version of the navigation algorithm works correctly.
    - To measure the real battery duration during typical operation.

6.3 Lessons learned

One of the biggest difficulties of this project was the lack of experience, knowledge and understanding of many of its aspects. Some of the tasks that were carried out had to be learnt along the way, overcoming countless obstacles and correcting mistakes until being satisfied with the obtained results.

A list of all the lessons and abilities that were acquired would be infinite, however, a summary of the most determining ones is presented here:

- Familiarization with vessels' legal regulations, which can be decisive when defining the system requirements. For that reason, they should be checked during the very first design phase.

- Familiarization the industry standards and its implementation. Some details are not always completely necessary but should be included to increase the product's appeal and accessibility.
- Precise estimations of the time required for each tasks are very challenging to make. When making these estimations, unexpected difficulties should be anticipated and tasks may be rearranged accordingly.
- During the realization of the project, new requirements and ideas can appear and we must be flexible enough to evaluate their addition when possible.
- Debugging elements must be always included in a prototype. This can complicate the design process but will save valuable time and effort during the testing and verification phase.
- Familiarization with the experimental modeling process of a real system and the difficulty and strictness required to obtain some of the measurements.
- Better understanding of the components selection criteria and practical experience comparing a width variety of them. Moreover, it must always be ensured that the selected component is recommended for new designs.
- Increased experience with basic laboratory instrumentation such as the oscilloscope and familiarization with more specific equipment such as the tachometer.
- PCB design and the conditions and details that must be ensured depending on its application. Moreover, familiarization with the Altium Designer® 19 software, the most complete and professionally used software of this kind.
- It is very easy to make small mistakes during the PCB fabrication and verification process, which can cause irreversible damage to the board or cause operation failures that are very challenging to detect.
- Introduction to PHP and SQL, as well as familiarization with web hosting providers, such as 000webhost.
- Algorithms implementation through FSMs, which are commonly used in the MCUs' firmware field.

Bibliography

[1] Geometría de masas: Definiciones y expresiones de cálculo (s.d). Available at https://www.fnb.upc.edu/mecanica/s01-gm/Geometr%C3%ADa%20de%20Masas%20-%20Definiciones.pdf Accessed May 2023.

[2] *MG996R High Torque Metal Gear Dual Ball Bearing Servo (s.d).* Available at https://www.electronicoscaldas.com/datasheet/MG996R_Tower-Pro.pdf Accessed May 2023.

[3] Oled i2c silkscreens are wrong, or are they? (s.d), 2021. Available at https://community.element14.com/members-area/personalblogs/b/blog/posts/oled-i2c-silkscreens-are-wrong-or-are-they Accessed April 2023.

[4] Agilent Technologies. *DC Power Analyzer Model N6705A User's Guide*, 2008. Available at https://www.atecorp.com/atecorp/media/pdfs/data-sheets/keysight-n6705a-dc-power-analyzer-manual.pdf?ext=.pdf Accessed May 2023.

[5] Arduino. *Bootloader*, 2023. Available at https://docs.arduino.cc/hacking/software/Bootloader#whats-a-bootloader Accessed July 2023.

[6] Baptiste, I., and Wood, A. *Bootstrap supply circuit and guidelines.* Allegro Microsystems, 2021. Available at https://www.allegromicro.com/-/media/files/application-notes/an296220-bootstrap-supply.pdf?sc_lang=en Accessed May 2023.

[7] Beale, A. Lead acid battery voltage charts (6v, 12v 24v), 2022. Available at https://footprinthero.com/lead-acid-battery-voltage-charts#:~:text=6V%20sealed%20lead%20acid%20batteries%20are%20fully%20charged%20at%20around,%25%20max%20depth%20of%20discharge Accessed May 2023.

[8] B.L.Theraja, and A.K.Theraja. *Electrical Technology.* Chand (S.) Co Ltd ,India, July 2008.

[9] Castaño, S. A. Modelo de motor dc, 2019. Available at https://controlautomaticoeducacion.com/analisis-de-sistemas/modelo-de-motor-dc/ Accessed May 2023.

[10] Circuits DK. *Reverse current attery protection circuits*, 2010. Available at https://www.circuits.dk/reverse-current-battery-protection-circuits/ Accessed May 2023.

[11] COLREGs. *International Regulations for Preventing Collisions at Sea*, 1972. Available at http://www.mar.ist.utl.pt/mventura/Projecto-Navios-I/IMO-Conventions%20(copies)/COLREG-1972.pdf Accessed May 2023.

[12] Concordia University. *Lead acid batteries*, 2016. Available at https://www.concordia.ca/content/dam/concordia/services/safety/docs/EHS-DOC-146_LeadAcidBatteries.pdf.

[13] Embedded. Pushing performance limitations in microcontrollers, 2010. Available at https://www.embedded.com/pushing-performance-limitations-in-microcontrollers/ Accessed June 2023.

[14] Espressif. *System Time*, 2023. Available at https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/system_time.html Accessed April 2023.

[15] Espressif. *USB console*, 2023. Available at https://docs.espressif.com/projects/esp-idf/en/v4.3/esp32s2/api-guides/usb-console.html Accessed May 2023.

[16] Franz, K. What is an h-bridge?, April 2023. Available at https://digilent.com/blog/what-is-an-h-bridge/ Accessed July 2023.

[17] Fujian. *MS Series SLA Battery*, 2012. Available at http://www.baterije.org/specifikacijebaterija/MS7-6.pdf Accessed May 2023.

[18] Hensen, M. What material will block gps signal, June 2023. Available at https://www.trackingsystemdirect.com/what-material-will-block-gps-signal/ Accessed June 2023.

[19] Hernández, C. G., and Quijano, M. G. Obtención experimental de los parámetros del motor que se utilizará en el sistema de locomoción de una esfera rodante. Master's thesis, Escuela de Ingeniería y Administración, Universidad Pontificia Bolivariana, 2009. Available at https://repository.upb.edu.co/handle/20.500.11912/504.

[20] HIOKI. *Lead-acid Battery Handbook, Facilitating Accurate Measurement of Lead-acid Batteries*, 2020. Available at https://idm-instrumentos.es/wp-content/uploads/2020/04/Bater%C3%ADas-de-plomo-%C3%A1cido-lead-acid-batteries.pdf Accesed May 2023.

[21] Ipanaqué, E. E. *MÓDULO GSM SIM800L*, 2017. Available at http://electropro.pe/image/data/imgProductos/140.%20M%C3%B3dulo%20GSM%20SIM800/SIM800L.pdf Accessed May 2023.

[22] Keysight. *Keysight E4980A/AL Precision LCR Meter User's Guide*, 2021. Available at https://www.keysight.com/us/en/assets/9018-05655/user-manuals/9018-05655.pdf Accessed May 2023.

[23] Kristjan, J. What you need to know about 2g/3g network shutdowns in europe, 2023. Available at https://1ot.com/resources/blog/2g-3g-network-shutdowns-europe#:~:text=Cullen%20International's%20research%20shows%20that,off%202G%20in%20early%202021. Accessed June 2023.

[24] M, S., 2023. Available at https://www.theengineerspost.com/dc-motors-types/?utm_content=cmp-true Accessed May 2023.

[25] Martín, I. G. Mechanical, electronic design and implementation of a cansat. Master's thesis, University of Granada, 2021.

[26] Mastin, L. Riemann sum – two rules, approximations, and examples, 2020. Available at https://www.storyofmathematics.com/riemann-sum/ Accessed July 2023.

[27] Maxon academy. *Maxon Motor Data and Operating Ranges*, 2010. Available at https://www.maxongroup.com/medias/sys_master/8798985748510.pdf Accessed May 2023.

[28] M.J.Bellido. Normas básicas y recomendaciones en el diseño de pcbs, 2015. Available at https://docplayer.es/26804954-Normas-basicas-y-recomendaciones-en-el-diseno-de-pcbs-manuel-j-bellido-diaz-oct
html Accessed Febrary 2023.

[29] Multicomp. *LED Yellow/Green 5mm*, 2012. Available at https://www.farnell.com/datasheets/1671521.pdf Accessed May 2023.

[30] Nichibo. *K3SFN*, 2018. Available at http://www.nichibo-motor.us/product/content/31 Accessed May 2023.

[31] of Engineering, R. C., and Technology. Voltage regulators with working principle, 2017. Available at https://www.rcet.org.in/uploads/academics/rohini_91175795696.pdf Accessed May 2023.

[32] OMRON. *Technical Explanation for Rotary Encoders (s.d)*. Available at https://www.ia.omron.com/data_pdf/guide/34/rotary_tg_e_7_2_rotary_connect_cg_e_2_1.pdf Accessed May 2023.

[33] OMRON. *Technical Explanation for Servomotors and Servo Drives (s.d)*. Available at https://www.ia.omron.com/data_pdf/guide/14/servo_tg_e_1_1.pdf Accesed May 2023.

[34] Pernia, M. A. Conceptos básicos de máquinas de corriente continua, 2014. Available at https://www.researchgate.net/profile/Marino-Pernia/publication/235752021_Conceptos_Basicos_de_Maquinas_de_corriente_continua/links/0912f5131e8e23bfa1000000/Conceptos-Basicos-de-Maquinas-de-corriente-continua.pdf.

[35] Peterson, Z. Pcb design guidelines for using tvs diode for transient protection, 2022. Available at https://resources.altium.com/p/pcb-design-guidelines-using-tvs-diode-transient-protection Accessed May 2023.

[36] Piskor, M. Ftdi.ft-x series usb to serial bridges can be still reasonable option even for today, 2017. Available at https://www.soselectronic.com/en/articles/ftdi/ftdi-microcontrollers-with-usb-interface-are-common-but-1783 Accessed May 2023.

[37] PowerSonic. *Sealed lead-acid batteries*, 2011. Available at https://www.power-sonic.com/wp-content/uploads/2018/12/Technical-Manual.pdf Accessed May 2023.

[38] PowerSonic. *How to charge lead acid batteries*, 2020. Available at https://www.power-sonic.com/wp-content/uploads/2020/12/How-to-charge-Lead-Acid-batteries.pdf Accessed May 2023.

[39] Rembor, K., Miller, D., and DiCola, T. Adafruit ina219 current sensor breakout library reference, 2021. Available at https://learn.adafruit.com/adafruit-ina219-current-sensor-breakout/library-reference Accessed July 2023.

[40] Rohm Semiconductor. *What is the Difference Between Linear and Switching Regulators? (s.d)*. Available at https://www.rohm.com/electronics-basics/dc-dc-converters/linear-vs-switching-regulators Accessed May 2023.

[41] RS components. *RS PRO Tachometer*. Available at https://docs.rs-online.com/bb02/A700000008880739.pdf Accessed May 2023.

[42] Sharp. *Technical Explanation for Rotary Encoders*, 1986. Available at http://www.bitsavers.org/components/sharp/Sharp_Rotary_Encoders_Mar1986.pdf Accessed May 2023.

[43] Simpson, C. *Linear and Switching Voltage Regulator Fundamentals Part 1*, 2011. Available at https://www.ti.com/lit/an/snva558/snva558.pdf?ts=1687712225530&ref_url=https%253A%252F%252Ffocus.ti.com%252Fdocs%252Fprod%252Ffolders%252Fprint%252Fua7812.html Accessed May 2023.

[44] SolomonSystech. *SSD1306*, 2008. Available at https://cdn-shop.adafruit.com/datasheets/SSD1306.pdf Accessed May 2023.

[45] Texas Instruments. *LM317 3-Terminal Adjustable Regulator*, 2020. Available at https://www.ti.com/lit/ds/symlink/lm317.pdf?ts=1690139122257&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FLM317%253Futm_source%253Dgoogle%2526utm_medium%253Dcpc%2526utm_campaign%253Dapp-null-null-GPN_EN-cpc-pf-google-wwe%2526utm_content%253DLM317%2526ds_k%253DLM317%2526DCM%253Dyes%2526gclid%253DCj0KCQjwn_OlBhDhARIsAG2y6zMLuuuvIxUmGNI6t2tWP_WEedI57di8dRSkxt2QBf_0HL_hYxYfiagaAtIbEALw_wcB%2526gclsrc%253Daw.ds Accessed May 2023.

[46] u-blox. *GNSS antennas*, 2019. Available at https://content.u-blox.com/sites/default/files/products/documents/GNSS-Antennas_AppNote_%28UBX-15030289%29.pdf Accessed May 2023.

[47] UCTRONICS. Uctronics módulo oled de 0.96 pulgadas 12864 128x64 amarillo azul ssd1306 driver i2c serial placa de exhibición autoluminosa para arduino raspberry pi pico, 2017. Available at https://www.amazon.com/UCTRONICS-SSD1306-Self-Luminous-Display-Raspberry/dp/B072Q2X2LL Accessed April 2023.

[48] Virgala, I., Frankovský, P., and Kenderová, M. Friction effect analysis of a dc motor. *American Journal of Mechanical Engineering 1* (2013), 1–5. Available at http://pubs.sciepub.com/ajme/1/1/1/.

[49] Virgala, I., and Kelemen, M. Experimental friction identification of a dc motor. *International Journal of Mechanics and Applications*, 3 (January 2013), 26–30. Available at https://www.researchgate.net/publication/253241458_Experimental_Friction_Identification_of_a_DC_Motor.

[50] Vovyo. *Brushless Motor VS Brushed Motor*, 2021. Available at https://www.vovyopump.com/brushless-motor-vs-brushed-motor/ Accessed Febrary 2023.

[51] W3Schools. $https://www.w3schools.com/php/php_superglobals_get.asp, 2023. Availableat$
Accessed July 2023.

Appendix A

Project budget

This appendix will analyse the investment made in terms of costs of material and manufacturing that was required to produce the boat's circuitry without including the manpower. It should provide an estimation of the product's price for commercialization purposes.

Due the product's nature, the estimation has been divided in:

- Components cost: a big variety of components and modules are required for our circuit, which represent the larger part of the overall costs. The cost of each sub-circuit has been estimated during this thesis, including providers and attempting to make the most affordable choices. The total cost of all the components that are required for the circuit is calculated in table A.1:

Components	
Circuit	**Cost [€]**
ESP32	4.06
GY-NEO6MV2	2.26
SIM800L	1.93
Adafruit 10-DOF	29.95
Navigation LEDs	1.03
SMD LEDs	2.70
Buttons	0.65
SSD130	1.85
Rotary encoder	2.03
DS1302	0.31
Power supply circuits	9.05
H-bridge	18.74
Voltage regulators	2.43
Total components cost [€]	76.99

Table A.1 – *Total cost of the circuit's components*

- PCB manufacturing: as we mentioned in Chapter 1, the PCB fabrication will be carried out by JLCPCB. Since this is just a prototype, no stencil will be ordered to solder the components because it is more expensive. The manufacturing price of each of the project's board is estimated in table A.2:

PCB manufacturing	
Board	**Cost [€]**
Main board	9.09
Mount board	8.62
Total PCB cost [€]	**17.71**

Table A.2 – *PCB manufacturing cost*

In total, adding the costs of tables A.1 and A.2, the total expenses for the materials and the manufacturing of the project's circuit ascends to 94.70 €.

Appendix B

Electronics BOM

In this appendix, all the components that are required for the manufacturing of this product are listed.

Component	**Designator**	**Quantity**
INA219 module	MOD1	1
Adafruit 10-DOF	MOD2	1
SIM800L module	MOD3	1
GY-NEO6MV2 module	MOD4	1
DS1302 module	MOD5	1
SSD1306 OLED	MOD6	1
ESP32-WROOM-32D	U1	1
CH340C	U2	1
10033526-N3212LF	J1	1
10mm 2 pin screw connector	J2, J3, J4	3
ESDS304	D1	1
VLMx1300 LED	D2, D3, D10, D12, D13 D15	6
VS-30BQ015-M3/9AT	D11	1
SSL56F	D8, D9, D14	3
2.54 mm female 2 pin header	D power, D white, D green, D red	4
1N4148W-7-F	D16, D17	2
1N5822	D18, D19, D20, D21	4
SMD push button	SW1, SW2 , boot, reset, ON/OFF	5
PEC11R-4020F-S0024	SW3	1
2N7002	Q1, Q2, Q9, Q10, Q11, Q16	6
IRL3803STRLPBF	Q3, Q4, Q5, Q13, Q14, Q15	6
SI1032R-T1-GE3	Q6, Q12	2
SI4131CDY-T1-GE3	Q7, Q8	2
SI1032R-T1-GE3	Q6, Q12	2

Table B.1 – *Electronics BOM*

Component	**Designator**	**Quantity**
Ferrite Bead 40 mΩ, 60 Ω @ 100 MHz	FB1	1
100 nF 0603 capacitor	C2, C3, C4, C6, C8, C9, C10, C12	8
10 μF 0603 capacitor	C1	1
1 μF 0603 capacitor	C11, C17	2
2.2 μF 0603 capacitor	C13, C14	2
22 nF 0603 capacitor	C15, C16	2
2.54 mm female 3 pin header	Servo1	1
10 kΩ 0603 resistor	R14, R15, R20, R28, R37	5
470 Ω 0603 resistor	R16, R39	2
300 Ω 0603 resistor	R17	1
820 Ω 0603 resistor	R18	1
50 Ω 0603 resistor	R15	1
27 kΩ 0603 resistor	R25	1
100 kΩ 0603 resistor	R26	1
275 Ω 0603 resistor	R27	1
200 Ω 0603 resistor	R10, R11	2
75 Ω 0603 resistor	R32	1
470 Ω 0805 resistor	Rtest	1
1 kΩ 0603 resistor	R33	1
2.2 kΩ 0603 resistor	R34	1
1 kΩ 0402 resistor	R9, R12, R29 ,R30	4
10 kΩ 0402 resistor	R3, R4	2

Table B.2 – *Electronics BOM*

Appendix C

Firmware

C.1 Power switch code

```
#include <Arduino.h>

//GPIO that detects if the push button is pressed or not:
#define BUTTON_DETECTOR_INPUT_PUSH_BUTTON   23
//GPIO that sets the board state (on/off):
#define ONOFF_BAT_OUTPUT                    13

//Interrupt rutine, it turns off the board:
void turn_off (void){
  digitalWrite(ONOFF_BAT_OUTPUT,LOW);
}

void setup() {
  //Pin modes configuration:
  pinMode(ONOFF_BAT_OUTPUT,OUTPUT);
  pinMode(BUTTON_DETECTOR_INPUT_PUSH_BUTTON,INPUT);

  //The ON/OFF pin must be set to HIGH as soon as the program starts:
  digitalWrite(ONOFF_BAT_OUTPUT,HIGH);
  //Delay so the interrupt doesn't detect the first push of the button:
  delay(200);

  //The interruption will be executed when a rising egde is detected on the
  BUTTON_DETECTOR_INPUT_PUSH_BUTTON pin:
  attachInterrupt(BUTTON_DETECTOR_INPUT_PUSH_BUTTON, turn_off, RISING);
}

void loop() {}
```

Listado C.1 – *Power switch code*

C.2 Rotary encoder code

```
#include <Arduino.h>

//GPIO connected to the A pin of the rotary encoder:
#define ROTARY_A_INPUT_PIN        2
//GPIO connected to the B pin of the rotary encoder:
#define ROTARY_B_INPUT_PIN        27
//GPIO connected to the button pin of the rotary encoder:
#define ROTARY_BUTTON_INPUT_PIN   5
```

```
//Current logic state of the A pin:
bool rotary_A_input_current_state;
//State of the A pin in the previous iteration of the the void loop() function:
bool rotary_A_input_last_state;
//Logic state of the button pin:
bool rotary_button_input_state;
/*Counter that is increased or decreased depending on the roation direction
 whenever a pulse is generated:*/
int rotary_pulses_counter=0;
//Auxiliar variable for the timer:
unsigned long time_aux=0;

void setup() {
  //Configures the input pins of the rotary as internal pull-ups:
  pinMode(ROTARY_A_INPUT_PIN,INPUT_PULLUP);
  pinMode(ROTARY_B_INPUT_PIN,INPUT_PULLUP);
  pinMode(ROTARY_BUTTON_INPUT_PIN,INPUT_PULLUP);

  Serial.begin (9600); //Serial initialization
  time_aux=millis();   //Timer initialization
  //Reads the initial state of the input A:
  rotary_A_input_last_state = digitalRead(ROTARY_A_INPUT_PIN);
}

void loop() {
  //Shows the button state every second:
  if(millis()-time_aux>=1000){
    //Reads the button state:
    rotary_button_input_state=digitalRead(ROTARY_BUTTON_INPUT_PIN);
    Serial.print("Rotary button state: ");
      //If the state is low the button is pressed:
      if(!rotary_button_input_state){
        Serial.println("Pressed");
      }else{
        //If the state is high the button is not pressed:
        Serial.println("Not pressed");
      }
    time_aux=millis(); //Updates the timer variable
  }

  //Reads the current state of the A pin:
  rotary_A_input_current_state = digitalRead(ROTARY_A_INPUT_PIN);
  //If the current state is different than the previous, a pulse has occurred:
  if(rotary_A_input_current_state!= rotary_A_input_last_state){
     //If the state of the B pin is different than the A pin's then the
     //direction was clockwise:
     if(digitalRead(ROTARY_B_INPUT_PIN) != rotary_A_input_current_state) {
       //The counter increases every time there is a clockwise pulse:
       rotary_pulses_counter++;
     } else {
       /*If the states of the A and B pins are the same, the rotation was
       counter-clockwise:*/
       //The counter decreases every time there is a counter-clockwise pulse:
       rotary_pulses_counter--;
     }
     //Displays the counter:
     Serial.print("Pulse count: ");
     Serial.println(rotary_pulses_counter);
   }
  //Stores the current A pin state in the previous state variable for the next
  //iteration of the loop:
  rotary_A_input_last_state = rotary_A_input_current_state;
}
```

Listado C.2 – *Generic code for a basic operation of the rotary encoder*

C.3 INA219 measurements reading code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_INA219.h>

Adafruit_INA219 ina219;  //Declaration of the INA219 object

//INA219 variables:
float shuntvoltage; //Shunt voltage
float busvoltage;   //Bus voltage
float current_mA;   //Current
float power_mW;     //Power

void setup() {
  Serial.begin(9600);    //Initializes the serial communication
  if (!ina219.begin())   //Checks if the INA219 sensor is detected:
    {
      /*If it's not detected, print the following message and stop the
      execution of the program:*/
      Serial.println("INA219 was not found");
      while (1);
    }
  //If it's detected display this message:
  Serial.println("INA219 initialization was correct");
}

void loop() {

  //Measurements reading:
  shuntvoltage = ina219.getShuntVoltage_mV();
  busvoltage = ina219.getBusVoltage_V();
  current_mA = (ina219.getCurrent_mA())*4;

  //Prints every variable:
  Serial.print("Bus Voltage:");
  Serial.print(busvoltage);
  Serial.println(" V");
  Serial.print("Shunt Voltage:");
  Serial.print(shuntvoltage);
  Serial.println(" mV");
  Serial.print("Current:");
  Serial.print(current_mA);
  Serial.println(" mA");

  delay(500); //Waits 500 ms
}
```

Listado C.3 – *Code to read measurements from the INA219*

C.4 SSD1306 code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

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

//Create an OLED display object connected to I2C:
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

void setup(){
  Serial.begin(9600); //Serial initialization:
  //Initialize OLED display with I2C address 0x3C:
  if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    //If the initialization is not correct stops the program:
    Serial.println("Failed to start SSD1306 OLED");
    while (1);
  }
  delay(200);
  oled.clearDisplay(); //Clears display
}

void loop() {
  oled.clearDisplay(); //Clears display
  oled.setTextSize(1); //Sets the text size
  oled.setTextColor(WHITE); //Sets the text color
  oled.setCursor(22,6); //Sets the cursor in a certain pixel
  oled.print("This is a test"); //Text to print
  oled.display(); //Displays the text
}
```

Listado C.4 – *Basic code for the SSD1306 display*

C.5 GY-NEO6MV2 test code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>

//Power switch definitions:
//GPIO that detects if the push button is pressed or not:
#define BUTTON_DETECTOR_INPUT_PUSH_BUTTON   23
//GPIO that sets the board state (on/off):
#define ONOFF_BAT_OUTPUT                    13

//Oled display sizes:
#define SCREEN_WIDTH 128 // OLED display width in pixels
#define SCREEN_HEIGHT 64 // OLED display height in pixels

//ESP32'S UART1 GPIOs:
#define ESP32_RX1_PIN 18
#define ESP32_TX1_PIN 19

//Creation of an OLED display object connected to I2C:
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire,-1);
HardwareSerial neogps(1); //Creates a HardwareSerial instance for UART1
TinyGPSPlus gps; //Creates a TinyGPSPlus instance

//Flag to indicate if a new gps data was read:
bool new_gps_data_flag = false;

//Power switch interrupt rutine, it turns off the board:
void turn_off (void){
  digitalWrite(ONOFF_BAT_OUTPUT,LOW);
}

//Function that prints the GPS latitude and longitude:
void print_gps_data(){
  oled.clearDisplay();
  oled.setTextColor(SSD1306_WHITE);
  oled.setTextSize(1);

  //If the gps location is valid, prints the latitude and longitude:
```

```
  if(gps.location.isValid()){
    oled.setCursor(2, 5);
    oled.print("Lat: ");
    oled.print(gps.location.lat(),6);

    oled.setCursor(2, 20);
    oled.print("Lng: ");
    oled.print(gps.location.lng(),6);
    oled.display();
  }
  //If the GPS location is not valid, warn it:
  else{
    oled.setCursor(2, 5);
    oled.setTextSize(1);
    oled.print("Invalid location");
    oled.display();
  }
}

void setup() {
  //Power switch circuit:
  pinMode(ONOFF_BAT_OUTPUT,OUTPUT);
  pinMode(BUTTON_DETECTOR_INPUT_PUSH_BUTTON,INPUT);
  //The ON/OFF pin must be set to HIGH as soon as the program starts:
  digitalWrite(ONOFF_BAT_OUTPUT,HIGH);
  //Delay so the interrupt doesn't detect the first push of the button:
  delay(200);
  //The interruption will be executed when a rising egde is detected on the
  //BUTTON_DETECTOR_INPUT_PUSH_BUTTON pin:
  attachInterrupt(BUTTON_DETECTOR_INPUT_PUSH_BUTTON, turn_off, RISING);
  //Begins UART1 communication for the  GY-NEO6MV2:
  neogps.begin(9600, SERIAL_8N1, ESP32_RX1_PIN, ESP32_TX1_PIN);
  //Initialize OLED display with I2C address 0x3C:
  if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    //If it's not detected, print the following message and stop the execution
    //of the program:
    Serial.println(F("Failed to start SSD1306 OLED"));
    while (1);
  }
  //Clears the OLED screen:
  oled.clearDisplay();
  oled.display();
  delay(1000); //1 second delay
}

void loop() {
  //During one second, if the UART1 is available it will read the GPS data:
  for (unsigned long gps_timer_aux = millis(); millis()-gps_timer_aux < 1000;){
    while (neogps.available()){
      //Reads and decodes the raw NMEA data to GPS coordinates:
      if(gps.encode(neogps.read())){
        new_gps_data_flag = true; //Updates de gps flag
      }
    }
  }
  //If a new GPS data has been received, calls the print_gps_data function:
  if(new_gps_data_flag){
    new_gps_data_flag = false; //Updates gps flag
    print_gps_data();
  }
  //If no data has been received, prints a warning:
  else{
    oled.clearDisplay();
    oled.setTextColor(SSD1306_WHITE);
    oled.setCursor(2, 5);
    oled.setTextSize(1);
    oled.print("Warning: no GPS data");
    oled.display();
  }
}
```

Listado C.5 – *Code to test the GY-NEO6MV2*

C.6 DS1302 code

```

#include <Arduino.h>
#include <ThreeWire.h>
#include <RtcDS1302.h>

#define DS1302_DAT 12 //GPIO connected to the data pin of the DS1302
#define DS1302_CLK 33 //GPIO connected to the clock pin of the DS1302
#define DS1302_RST 32 //GPIO connected to the data reset of the DS1302

//Defines an instance of the three wire method:
ThreeWire myWire(DS1302_DAT,DS1302_CLK,DS1302_RST);

//Constructs a RTCDS1302 object using the ThreeWire instance:
RtcDS1302<ThreeWire> Rtc(myWire);

//Variables to store the date and time
uint8_t current_hour, current_minute, current_second, month, day,
 day_of_the_week;
//Variable that stores the year:
uint16_t year;
//Auxiliar variable for the timer:
unsigned long time_aux;

void setup ()
{
    Serial.begin(9600); //Serial initialization

    //Displays the date and time of compilation:
    Serial.print("Compilation date and time: ");
    Serial.print(__DATE__);
    Serial.print(" ");
    Serial.println(__TIME__);

    Rtc.Begin(); //RTC library initialization
    //Creates a RtcDateTime instance from the compilation date and time:
    RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
    Serial.println(); //Prints an empty line

    //Checks if the date and time values of the RTC are valid:
    if (!Rtc.IsDateTimeValid()){
        /*If not, prints a warning and sets the RTC time to the time and date
        of compilation:*/
        Serial.println("RTC lost confidence in the DateTime!");
        Rtc.SetDateTime(compiled);
    }

    if (Rtc.GetIsWriteProtected()) //Checks if the RTC was write protected
    {
        //If it was, prints a warning en disables the write protection:
        Serial.println("RTC was write protected, enabling writing now");
        Rtc.SetIsWriteProtected(false);
    }

    if (!Rtc.GetIsRunning()) //Checks if the clock is running on the RTC
    {
        //If it wasn't, prints a warning a starts running it:
```

```
        Serial.println("RTC was not actively running, starting now");
        Rtc.SetIsRunning(true);
    }

    //Creates a RTCDateTime instance and stores the the RTC's date and time:
    RtcDateTime now = Rtc.GetDateTime();

    if (now < compiled) //Cheks if the RTC time is older than the compiled time
    {
        /*If that's the case, prints a warning a sets the RTC time to the time
        and date of compilation:*/
        Serial.println("RTC is older than compile time!");
        Rtc.SetDateTime(compiled);
    }

    //Checks if the RTC time is newer than the compiled time:
    else if (now > compiled)
    {
        //If that's the case, prints a notice:
        Serial.println("RTC is newer than compile time.");
    }

    //Checks if the RTC time is the same as the compiled time:
    else if (now == compiled)
    {
        /*If that's the case, prints a notice (it is unlikely but it's not a
        problem):*/
        Serial.println("RTC is the same as compile time!");
    }

    time_aux=millis(); //Timer initialization

}

void loop ()
{
    //Every 5 seconds do:
    if(millis()-time_aux>=5000){

        //Updates the time and date of the now instance:
        RtcDateTime now = Rtc.GetDateTime();

        /*Stores the date and time in their corresponding date and time
        variables:*/
        month=now.Month();
        day=now.Day();
        year=now.Year();
        current_hour=now.Hour();
        current_minute=now.Minute();
        current_second=now.Second();
        day_of_the_week=now.DayOfWeek();

        Serial.println(); //Prints an empty line

        //Prints the month with 2 digits:
        if(month<10){
            Serial.print("0");
            Serial.print(month);
        }else{
            Serial.print(month);
        }
        Serial.print("/");

        //Prints the day with 2 digits:
        if(day<10){
            Serial.print("0");
            Serial.print(day);
        }else{
            Serial.print(day);
        }
        Serial.print("/");

        //Prints the year:
        Serial.print(year);
        Serial.print(" ");

        //Prints the hour with 2 digits:
        if(current_hour<10){
            Serial.print("0");
            Serial.print(current_hour);
        }else{
            Serial.print(current_hour);
        }
        Serial.print(":");

        //Prints the minute with 2 digits:
        if(current_minute<10){
            Serial.print("0");
            Serial.print(current_minute);
        }else{
            Serial.print(current_minute);
        }
        Serial.print(":");

        //Prints the second with 2 digits:
        if(current_second<10){
            Serial.print("0");
            Serial.print(current_second);
        }else{
            Serial.print(current_second);
        }

    time_aux=millis(); //Updates the auxiliar timer variable
    }
}
```

Listado C.6 – *Code for a basic operation of the DS1302*

C.7 "BUTTONS" pin code

```
#include <Arduino.h>

#define BUTTONS_INPUT_PIN 39 // GPIO connected to the "BUTTONS" pin

int buttons_input_pin_value; // "Analog" value of the buttons pin
unsigned int time_aux;       // Auxiliar variable for the timer

void setup() {
  Serial.begin(9600); //Serial initialization
  //Configures the BUTTON_INPUT_PIN as an input:
  pinMode(BUTTONS_INPUT_PIN,INPUT);

}

void loop() {
    //Every 500 ms it prints the buttons_input_pin_value lecture and indicates
    //which button was pressed:
    if(millis()-time_aux>=500){
    //Analog reads the initial voltage at the BUTTON_INPUT_PIN:
    buttons_input_pin_value=analogRead(BUTTONS_INPUT_PIN);
    Serial.print("Button analog read: ");
    Serial.print(buttons_input_pin_value);
    //A lecture higher than 3900 means that the voltage is higher than 3.14 V:
    if(buttons_input_pin_value>3900){
      Serial.println("  Button 2 pressed");
    }
    //If the lecture is between 1600 and 2300 means that the voltage is between
    //1.28 V and 1.85 V:
    else if(buttons_input_pin_value>=1600 && buttons_input_pin_value<=2300){
      Serial.println("  Button 1 pressed");
    }
    //A lecture lower than 100 means that the voltage is almost 0 V
    else if(buttons_input_pin_value<=100){
      Serial.println("  No buttons pressed");
    }
    time_aux=millis(); //Updates the timer variable
  }
}
```

Listado C.7 – *Code to read the state of the "BUTTONS" pin*

C.8 Charger circuit test code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_INA219.h>

//Rotary encoder definitions:
//GPIO connected to the A pin of the rotary encoder
#define ROTARY_A_INPUT_PIN        2
//GPIO connected to the B pin of the rotary encoder
#define ROTARY_B_INPUT_PIN        27
//GPIO connected to the button pin of the rotary encoder
#define ROTARY_BUTTON_INPUT_PIN   5

//Charger definitions:
#define CHARGER_OUTPUT_PIN          26  //GPIO connected to the charger port
#define CHARGER_VOLTAGE_SENSOR_PIN  36  //GPIO connected to the Vsense port

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

Adafruit_INA219 ina219;  //Declaration of the INA219 object

//Creates an OLED display object connected to I2C:
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

bool rotary_A_input_current_state;  //Current state of the A pin (high or low)
//State of the A pin in the previous iteration of the the void loop() function:
bool rotary_A_input_last_state;
/* Counter that is increased or decreased depending on the roation direction
 whenever apulse is generated:*/
int rotary_pulses_counter=0;
unsigned long charger_timer_aux=0;  //Auxiliar variable for the timer
float charger_vsense_voltage;       //Voltage of the charger

void setup() {
  //Configures the input pins of the rotary as internal pull-ups:
  pinMode(ROTARY_A_INPUT_PIN,INPUT_PULLUP);
  pinMode(ROTARY_B_INPUT_PIN,INPUT_PULLUP);
  pinMode(ROTARY_BUTTON_INPUT_PIN,INPUT_PULLUP);
  //Configures the chager circuit pins:
  pinMode(CHARGER_VOLTAGE_SENSOR_PIN,INPUT);
  pinMode(CHARGER_OUTPUT_PIN,OUTPUT);

  Serial.begin(9600); //Serial initialization

  //Initialize OLED display with I2C address 0x3C:
  if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    //If it's not detected, print the following message and stop the execution
    //of the program:
    Serial.println(F("Failed to start SSD1306 OLED"));
    while (1);
  }

  if (! ina219.begin()) //Checks if the INA219 sensor is detected:
  {
    //If it's not detected, print the following message and stop the execution
    //of the program:
    Serial.println("Failed to find INA219 chip");
    while (1);
  }
  oled.clearDisplay(); //Clears display
  charger_timer_aux=millis();  //Timer initialization
  //Reads the initial state of the input A:
  rotary_A_input_last_state = digitalRead(ROTARY_A_INPUT_PIN);
}

void loop() {
  //Obtains the charger voltage in [V]
  charger_vsense_voltage=((analogRead(CHARGER_VOLTAGE_SENSOR_PIN))*3.3/4095)
 *37300/10000;
  //Voltage that is supplied to the battery
  float battery_voltage = ina219.getBusVoltage_V();
  //Reads the current state of the A pin:
  rotary_A_input_current_state = digitalRead(ROTARY_A_INPUT_PIN);
  //If the current state is different than the previous, a pulse has occurred:
  if (rotary_A_input_current_state!= rotary_A_input_last_state){
     //If the state of the B pin is different than the A pin's then the
     //direction was clockwise:
     if (digitalRead(ROTARY_B_INPUT_PIN) != rotary_A_input_current_state) {
       //The rotary pulses counter will be limited between 0 to 255:
       if(rotary_pulses_counter<255){
       //The counter increases every time there is a clockwise pulse:
        rotary_pulses_counter++;
       }
     } else
     {
       //If the states of the A and B pins are the same, the rotation was
       //counter-clockwise:
       if(rotary_pulses_counter>0){
        //The counter decreases every time there is a counter-clockwise pulse:
        rotary_pulses_counter--;
       }
     }
    //The number of pulses of the rotary corresponds to the analog voltage at
    //the charger pin:
    dacWrite(CHARGER_OUTPUT_PIN,rotary_pulses_counter);
  }
  if(millis()-charger_timer_aux>=200){
    //Displays the pulses counter, the charger voltage and the battery voltage:
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print(rotary_pulses_counter);
    oled.setCursor(22,26);
    oled.print(charger_vsense_voltage);
    oled.setCursor(22,48);
    oled.print(battery_voltage);
    oled.display();
    charger_timer_aux=millis(); //Updates the timer
  }
  //Stores the current A pin state in the previous state variable for the next
  //iteration of the loop:
  rotary_A_input_last_state = rotary_A_input_current_state;
}
```

Listado C.8 – *Code to test the charger circuit*

C.9 Adafruit 10-DOF test code

```
#include <Arduino.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303.h>
#include <Adafruit_10DOF.h>

//Assign a unique ID to the sensors:
Adafruit_10DOF                imu_sensor=Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified acelerometer=Adafruit_LSM303_Accel_Unified(30301);
Adafruit_LSM303_Mag_Unified   magnetometer= Adafruit_LSM303_Mag_Unified(30302);

//Structures to provide a single sensor event in a common format:
sensors_event_t acelerometer_event;
sensors_event_t magnetometer_event;
sensors_vec_t   orientation;

//Variable to store the ship's compass:
int ship_compass;
//Variable to store the ship's roll:
int ship_roll;

//Function to initialize the magnetometer and accelerometer:
void IMU_initialization()
{
  //Initializes the accelerometer:
  if(!acelerometer.begin())
  {
    //If it is not detected, shows a warning and stops the program:
    Serial.println(F("LSM303 acceleromter not detected"));
    while(1);
  }
  //Initializes the magnetometer:
  if(!magnetometer.begin())
  {
    //If it is not detected, shows a warning and stops the program:
    Serial.println("LSM303 magnetometer not detected");
    while(1);
  }
}

void setup() {
  Serial.begin(9600);   //Initializes the serial port
  Serial.println("Adafruit 10-DOF Test");
  Serial.println("");
  IMU_initialization(); //Initializes the sensors
}

void loop() {
  //Gets a new accelerometer event:
  acelerometer.getEvent(&acelerometer_event);
  //Gets the roll and the pitch:
  if(imu_sensor.accelGetOrientation(&acelerometer_event, &orientation))
  {
    //If the operation is correct, prints the roll:
    Serial.print("Roll: ");
    ship_roll=orientation.roll;
    Serial.print(orientation.roll);
    //If the roll is less than 16  shows a capsize warning:
    if(ship_roll<=15){
      Serial.print(" CAPSIZE WARNING");
    }
    Serial.print("\t");
  }
  //Gets a new magnetometer event:
  magnetometer.getEvent(&magnetometer_event);
  //Gets the boat's heading:
  if (imu_sensor.magGetOrientation(SENSOR_AXIS_Y, &magnetometer_event, &
 orientation))
  {
    //Subtracts 90 to the heading so it is equal to the compass' angle:
    ship_compass=orientation.heading-90;
    //Adjusts the compass value so it ranges from 0 to 360  :
    if (ship_compass < 0) {
      ship_compass = 360 + ship_compass;
    }
    //Displays the ship's compass:
    Serial.print("Ship's compass: ");
    Serial.println(ship_compass);
  }
  delay(500);//A new measurement is obtained every 500 ms
}
```

Listado C.9 – *Code to test the Adafruit 10-DOF*

C.10 Server codes

```php
<?php

//Defines the database's host, username, password and name:
define('DB_HOST', 'localhost');
define('DB_USERNAME', 'YourUser');
define('DB_PASSWORD', 'YourName');
define('DB_NAME', 'YourName');

//Sets the default timezone used by all date/time functions:
date_default_timezone_set('Europe/Madrid');

//Connects to the database:
$db = new mysqli(DB_HOST, DB_USERNAME, DB_PASSWORD, DB_NAME);

//Displays error if it failed to connect:
if ($db->connect_errno) {
    echo "Connection to database is failed: ".$db->connect_error;
    exit();
}
```

Listado C.10 – *config.php*

```php
<?php

//Includes the configuration file:
require 'config.php';

//Obtains the current latitude and longitude through the GET method:
$lat = $_GET['lat'];
$lng = $_GET['lng'];
$warnings = $_GET['warnings'];

//Displays those values:
```

```
echo $lat;
echo "<br>";
echo $lng;
echo "<br>";
echo $warnings ;
echo "<br>";

//Inserts the values into the tbl_gps table:
$sql = "INSERT INTO tbl_gps(lat,lng,warnings,created_date)
  VALUES('".$lat."','".$lng."','".$warnings."','".date("Y-m-d H:i:s")."')";

//Checks if there was an error in the query:
if($db->query($sql) === FALSE)
  { echo "Error: " . $sql . "<br>" . $db->error; }
```

Listado C.11 – *add_current_loc.php*

```
<?php

//Includes the configuration file:
require 'config.php';

//Extracts the current coordinates from the tbl_gps:
$query = "SELECT `lat`,`lng` FROM `tbl_gps` WHERE id=(SELECT max(id) FROM `
 tbl_gps`)";
$res = mysqli_query($db,$query); //Performs the query
//Fetches one row of data from the result set and returns it as an array:
$row = mysqli_fetch_array($res);

//Displays the values:
$lati=$row[0];
$long=$row [1];
echo $lati;
echo "<br>";
echo $long;

//Detects if there was an error:
if(mysqli_query($db, $query)){}
else{
echo "ERROR: Could not able to execute". $query." ". mysqli_error($db);
}

//Displays the google maps site with our desired coordinates:
?>

<iframe width="100%" height="500" src="https://maps.google.com/maps?q=<?php echo
  $lati; ?>,<?php echo $long; ?>&output=embed"></iframe>

<?php
```

Listado C.12 – *current_loc_map.php*

```
<?php

//Includes the configuration file:
require 'config.php';

//Takes the last warning and its date from the tbl_gps table:
$query = "SELECT `warnings`,`created_date` FROM `tbl_gps`
         WHERE id=(SELECT max(id) FROM `tbl_gps`)";
$res = mysqli_query($db,$query); //Performs the query
$row = mysqli_fetch_array($res); //Fetches one row of data from the result set
 and returns it as an array
//Displays the query result:
echo $row [0];
echo "<br>";
echo $row [1];
```

```
//Detects if there was a query error:
if(mysqli_query($db, $query)){}
else{
echo "ERROR: Could not able to execute". $query." ". mysqli_error($db);
}
```

Listado C.13 – *add_desired_loc.php*

```
<?php

//Includes the configuration file :
require 'config.php';

//Check if the GET request of the desired_lat variable is not null:
if( isset($_GET['desired_lat'])) {
    //Copies the GET requests into the POST array:
    $_POST['desired_lat']=$_GET['desired_lat'];
    $_POST['desired_lng']=$_GET['desired_lng'];
    $_POST['square_side']=$_GET['square_side'];
}

//These are in case setting headers, forcing it to always expire:
header('Cache-Control: no-cache, must-revalidate');
//Logs the POST string into the error log:
error_log(print_r($_POST,TRUE));

//Checks if the tag post is there and if it's been a proper form post:
if( isset($_POST['desired_lat']) && isset($_POST['desired_lng']) && isset($_POST
 ['square_side']) ){
  //Checks if the SQL key is correct:
  //if($_POST['key']==$SQLKEY){
    //Inserts the desired coordinates into the desired_location table:
    $query="INSERT INTO desired_location (desired_lat, desired_lng, square_side,
 created_date) VALUES ('".$_POST['desired_lat']."','".$_POST['desired_lng']."','
 ".$_POST['square_side']."','".date("Y-m-d H:i:s")."')";
    //Checks if the magic quotes are enabled and remove them if that's the case:
    if(get_magic_quotes_gpc()){
      $query=stripslashes($query);
    }
    //Checks if there was a connection error:
    if($db->connect_error){
      //Reports a connection error:
      header("HTTP/1.0 400 Bad Request");
      echo "ERROR Database Connection Failed: " . $db->connect_error,
 E_USER_ERROR;
    }
    else{
      //Performs the query:
      $result=$db->query($query);
      if($result === false){
        //If there was a error, notifies it:
        header("HTTP/1.0 400 Bad Request");
        echo "Wrong SQL: " . $query . " Error: " . $db->error, E_USER_ERROR;
      }
      //If the connection was correct, notifies it:
      else{
        echo "OK";
      }
      $db->close(); //Closes the database connection
    }
  }
//Notifies that the tag post were not correct:
else{
  header("HTTP/1.0 400 Bad Request");
  echo "Bad Request2";
}
?>
```

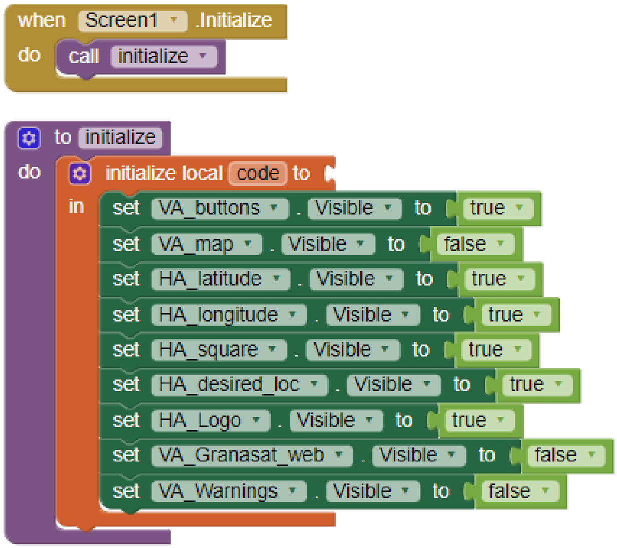
Listado C.14 – *add_desired_loc.php*

```php
<?php

//Includes the configuration file:
require 'config.php';

//Takes the last desired coordinates and the square side from the desired_location table:
$query = "SELECT `desired_lat`,`desired_lng`,`square_side` FROM `desired_location`
        WHERE id=(SELECT max(id) FROM `desired_location`)";
$res = mysqli_query($db,$query); //Performs the query
$row = mysqli_fetch_array($res); //Fetches one row of data from the result set and returns it as an array
//Displays the query result:
echo $row [0];
echo "<br>";
echo $row [1];
echo "<br>";
echo $row [2];

//Detects if there was a query error:
if(mysqli_query($db, $query)){}
else{
echo "ERROR: Could not able to execute". $query." ". mysqli_error($db);
}
```

Listado C.15 – *display_desired_loc.php*

C.11 App code

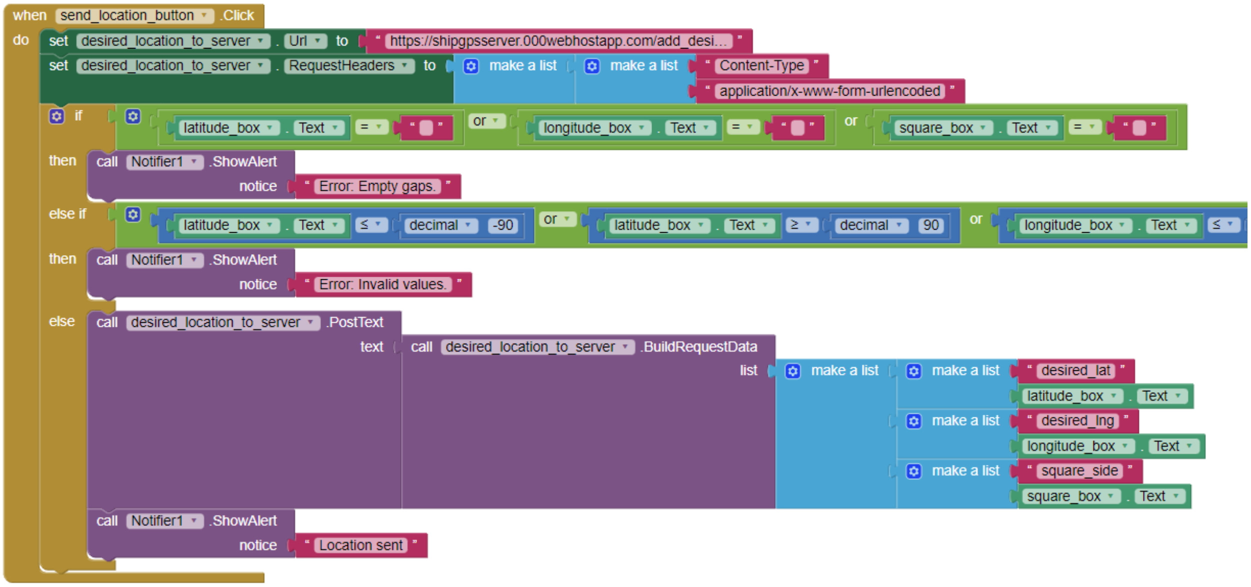
First, the app will be initialized to make visible or invisible the different arrangements that form the interface (figure C.1).



```
when Screen1 .Initialize
do  call initialize

to initialize
do  initialize local code to
    in  set VA_buttons . Visible to true
        set VA_map . Visible to false
        set HA_latitude . Visible to true
        set HA_longitude . Visible to true
        set HA_square . Visible to true
        set HA_desired_loc . Visible to true
        set HA_Logo . Visible to true
        set VA_Granasat_web . Visible to false
        set VA_Warnings . Visible to false
```

Figure C.1 – *App initialization block*

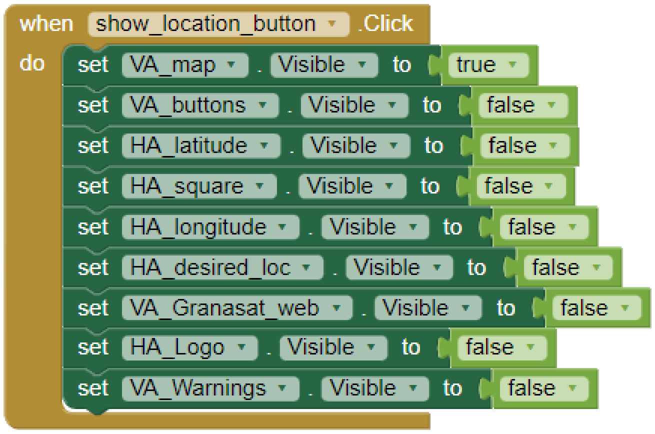
If the "send location button" is pressed, a HTML form is created from the data introduced in the desired latitude, longitude and square side text boxes. If any of the boxes are empty or the values are not within the correct range, errors will be displayed and the data will not be sent. If every text box is correct, the data will be sent to the server using the POST method. This can be observed in figure C.2, however one of the conditions is too large to fit the page. This condition is met if the latitude is larger than 90 or smaller than -90, if the longitude is larger than 180 or smaller than -180 or if the square side is larger than 9 or smaller than 1.



```
when send_location_button .Click
do  set desired_location_to_server . Url to " https://shipgpsserver.000webhostapp.com/add_desi... "
    set desired_location_to_server . RequestHeaders to make a list  make a list  " Content-Type "
                                                                                  " application/x-www-form-urlencoded "
    if  latitude_box . Text = " " or longitude_box . Text = " " or square_box . Text = " "
    then  call Notifier1 .ShowAlert
                           notice " Error: Empty gaps. "
    else if  latitude_box . Text ≤ decimal -90 or latitude_box . Text ≥ decimal 90 or longitude_box . Text ≤
    then  call Notifier1 .ShowAlert
                           notice " Error: Invalid values. "
    else  call desired_location_to_server .PostText
                                     text  call desired_location_to_server .BuildRequestData
                                                                       list  make a list  make a list  " desired_lat "
                                                                                                       latitude_box . Text
                                                                                          make a list  " desired_lng "
                                                                                                       longitude_box . Text
                                                                                          make a list  " square_side "
                                                                                                       square_box . Text
          call Notifier1 .ShowAlert
                           notice " Location sent "
```

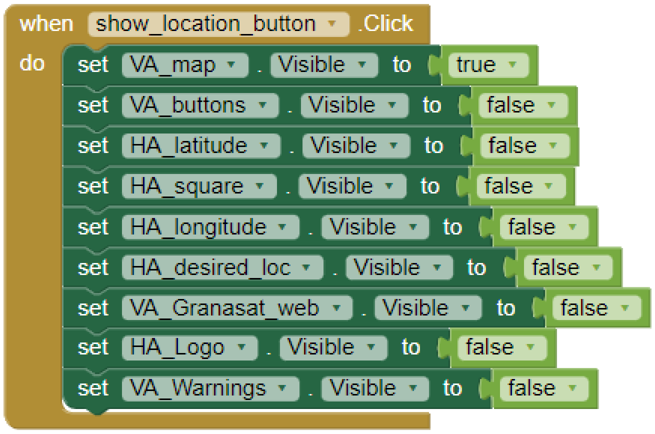
Figure C.2 – *App "send button" block*

If the "show location" button is pressed, the map arrangement will be visible and the rest invisible (figure C.3). Once this arrangement is visible, if the user presses the "back" button, the interface will return to the initial state (figure C.4).



```
when show_location_button .Click
do  set VA_map . Visible to true
    set VA_buttons . Visible to false
    set HA_latitude . Visible to false
    set HA_square . Visible to false
    set HA_longitude . Visible to false
    set HA_desired_loc . Visible to false
    set VA_Granasat_web . Visible to false
    set HA_Logo . Visible to false
    set VA_Warnings . Visible to false
```

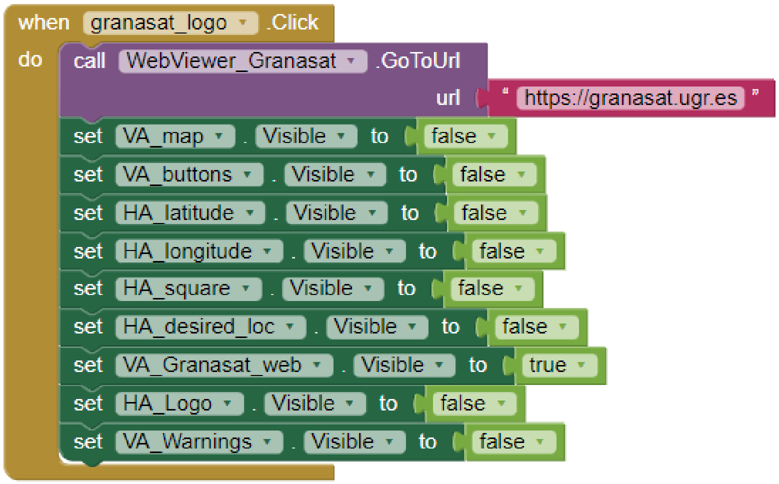
Figure C.3 – *App "show location" block*



```
when show_location_button .Click
do  set VA_map . Visible to true
    set VA_buttons . Visible to false
    set HA_latitude . Visible to false
    set HA_square . Visible to false
    set HA_longitude . Visible to false
    set HA_desired_loc . Visible to false
    set VA_Granasat_web . Visible to false
    set HA_Logo . Visible to false
    set VA_Warnings . Visible to false
```

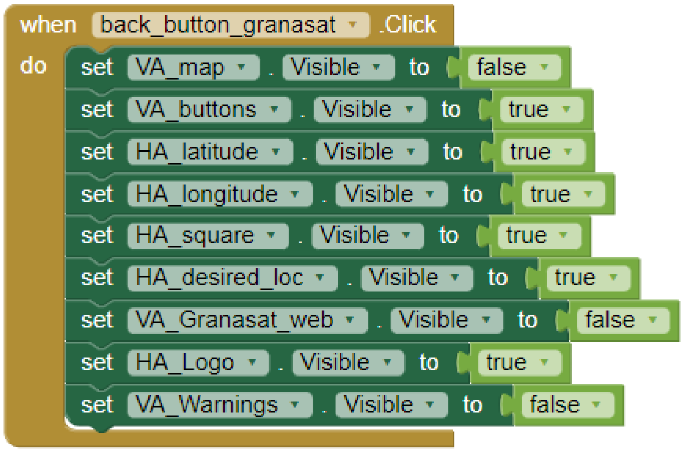
Figure C.4 – *App "back" block*

If the "Granasat" logo is pressed, the app will display the Granasat website (figure C.5). Once the web is shown, if the user presses the "back" button, the interface will return to the initial state (figure C.6).



```
when granasat_logo .Click
do  call WebViewer_Granasat .GoToUrl
                          url " https://granasat.ugr.es "
    set VA_map . Visible to false
    set VA_buttons . Visible to false
    set HA_latitude . Visible to false
    set HA_longitude . Visible to false
    set HA_square . Visible to false
    set HA_desired_loc . Visible to false
    set VA_Granasat_web . Visible to true
    set HA_Logo . Visible to false
    set VA_Warnings . Visible to false
```

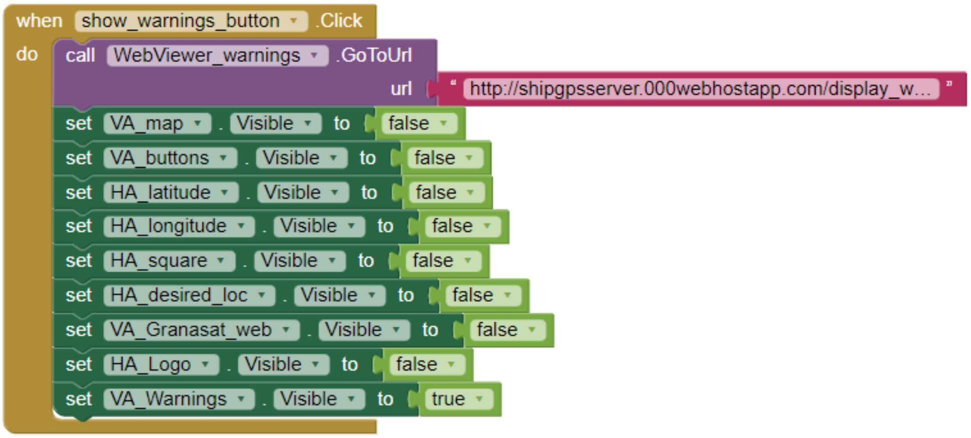
Figure C.5 – *App's Granasat logo block*



```
when back_button_granasat .Click
do  set VA_map . Visible to false
    set VA_buttons . Visible to true
    set HA_latitude . Visible to true
    set HA_longitude . Visible to true
    set HA_square . Visible to true
    set HA_desired_loc . Visible to true
    set VA_Granasat_web . Visible to false
    set HA_Logo . Visible to true
    set VA_Warnings . Visible to false
```

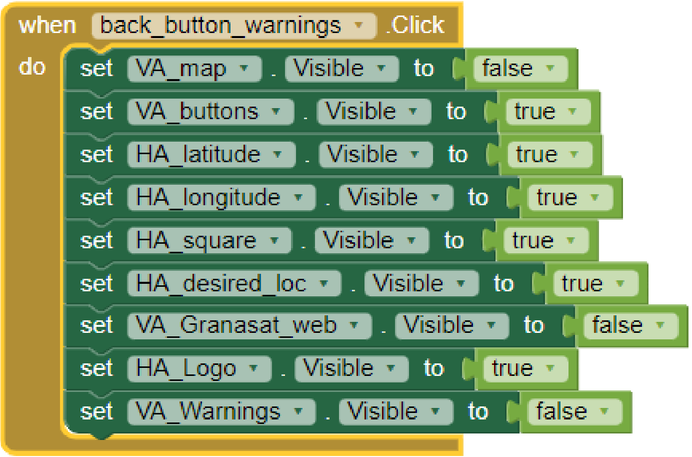
Figure C.6 – *App "back" block*

If the "Show warnings button" is pressed, the app will display our server's warning website (figure C.7). Once the web is shown, if the user presses the "back" button, the interface will return to the initial state (figure C.8).



```
when show_warnings_button .Click
do  call WebViewer_warnings .GoToUrl
                            url " http://shipgpsserver.000webhostapp.com/display_w... "
    set VA_map . Visible to false
    set VA_buttons . Visible to false
    set HA_latitude . Visible to false
    set HA_longitude . Visible to false
    set HA_square . Visible to false
    set HA_desired_loc . Visible to false
    set VA_Granasat_web . Visible to false
    set HA_Logo . Visible to false
    set VA_Warnings . Visible to true
```

Figure C.7 – *App "Show warning button" block*



```
when back_button_warnings .Click
do
  set VA_map . Visible to false
  set VA_buttons . Visible to true
  set HA_latitude . Visible to true
  set HA_longitude . Visible to true
  set HA_square . Visible to true
  set HA_desired_loc . Visible to true
  set VA_Granasat_web . Visible to false
  set HA_Logo . Visible to true
  set VA_Warnings . Visible to false
```

Figure C.8 – *App "back" block*

C.12 ESP32 SIM800L and app communication code

```
#include <Arduino.h>
#include <HardwareSerial.h>

#define TINY_GSM_MODEM_SIM800 //Definition of the module we are using, it must be defined before including the library
#include <TinyGsmClient.h>

//GPIO that detects if the push button is pressed or not:
#define BUTTON_DETECTOR_INPUT_PUSH_BUTTON   23
//GPIO that sets the board state (on/off):
#define ONOFF_BAT_OUTPUT                    13

//ESP32'S UART2 GPIOs:
#define ESP32_RX2_PIN   16
#define ESP32_TX2_PIN   17

//Define your SIM card code if necessary so it can be unlocked:
#define GSM_PIN ""

//Creates a HardwareSerial instance for the UART 2:
HardwareSerial SerialAT(2);
//Creates a TinyGsm instance with the name of the UART 2 serial:
TinyGsm modem(SerialAT);

unsigned long time_aux;     //Auxiliar variable for the timer
//Desired latitude established by the user in decimal degrees:
float desired_lat_float;
//Desired longitude established by the user in decimal degrees:
float desired_lng_float;

//Declaration of the GPRS credentials:
const char apn[]  = "airtelwap.es"; //Vodafone's APN
const char gprsUser[] = "";         //Only declare it if necessary
const char gprsPass[] = "";         //Only declare it if necessary

//Function that return the position of certain character in a char
//array:
int search_char_position(char response[],char searched_character){

  int length = strlen(response); //Obtains the number of characters of the array
   for(int i=0; i<length; i++) { //For each position of the array checks if the
 character is equal to the one searched
      if(response[i] == searched_character) {
         return i; //If the character is contained in the array it returns its
 position
      }
   }
  return -1; //If the character isn't contained in the array it returns -1
}

//Interrupt rutine, it turns off the board:
void turn_off (void){
  digitalWrite(ONOFF_BAT_OUTPUT,LOW);
}

//Function that sends the specified AT command with a timeout and //checks if
 the response is correct:
bool sendAT(const char* ATcommand, const char* expected_answer, unsigned int
 timeout){

    uint8_t i=0;            //Auxiliar variable for the do while loop
    //Flag variable to indicate if the command was answered:
    bool answerFlag=0;
    //Char array that stores the answer to the command:
    char answer[100];
    //Auxiliar timer variable that stores the time before the do while
    //loop started:
    unsigned long previous;

    //Initializes the answer array to null characters:
    memset(answer, '\0', 100);
    //Cleans the input buffer:
    while( SerialAT.available() > 0) SerialAT.read();
    delay(100);

    //Checks if the AT command's first character is not null has
    //content and sends it:
    if (ATcommand[0] != '\0'){
      SerialAT.println(ATcommand); //Send the AT command
    }

    previous=millis(); //Stores the time at which the do loop started

    //Reads serial data if the serial is available until an answer is
    //received or the timeout is reached:
    do{
       //Checks if there is data in the UART input buffer, if there is
       //it reads it and compare it with the expected answer:
        if(SerialAT.available() != 0){
            //Each character is stored in each of the array's position:
            answer[i] = SerialAT.read();
            //The position in the array is increased each iteration of
            //the loop:
            i++;
            //If the desired answer is received it is indicated
            //through the answer flag
            if(strstr(answer, expected_answer) != NULL){
                answerFlag = 1;
            }
        }
    }
    //Checks if the expected answer is received or if the timeout has
    //been reached:
    while((answerFlag == 0) && ((millis() - previous) < timeout));

  Serial.println(answer); //Prints the answer
  return answerFlag; //Returns the answer flag
}

//Specific function to send the HTTPREAD comment and extract the data:
void sendHHTPREAD(const char* ATcommand, unsigned int timeout){

    uint8_t i=0;             //Auxiliar variable for the do while loop
    //Char array that stores the response to the command:
    char response[100];
    //Part of the response array that will be used to check the
    //response to the command:
    char response_check[12];
    char lat_read[11];       //Desired latitude lecture
    char lng_read[12];       //Desired longitude lecture
    char square_side_read[2];//Desired square side lecture
    float desired_lat_float; //Desired latitude lecture in float type
    float desired_lng_float; //Desired longitude lecture in float type
    //Desired square side in float type:
    float desired_square_side_float;
    //Auxiliar timer variable that stores the time before the do while
    //loop started:
    unsigned long previous;
    //Position of a specific character within an char array:
    int position;


    //Initializes the char arrays to null characters
    memset(response, '\0', 100);
    memset(response_check,'\0',12);
    memset(lat_read,'\0',11);
    memset(lng_read,'\0',12);
    memset(square_side_read,'\0',2);

    //Cleans the input buffer:
    while( SerialAT.available() > 0) SerialAT.read();
    delay(100);

    //Checks if the AT command's first character is not null has
    //content and sends it:
    if (ATcommand[0] != '\0'){
      SerialAT.println(ATcommand); //Sends the AT command
    }

    previous = millis(); //Stores the time at which the do loop started

    //Reads serial data if the serial is available until the timeout
    //is reached:
    do{
        //Checks if there is data in the UART input buffer, if there
        //is it reads it and compare it
        if(SerialAT.available() != 0){
            //Each character is stored in each of the array's position:
            response[i] = SerialAT.read();
            //The position in the array is increased each iteration of
            //the loop
            i++;
        }
    }
    //Checks if the timeout has been reached:
    while(((millis() - previous) < timeout));

  //Stores 11 characters from the 8th position of the response into
  //the response_check array
  memcpy(response_check,&response[8],11);
  //Checks if the response is +HTTPACTION, if that is the case it is
```

```
 //reading the previous command's response and the coordenates'
 //lecure will be incorrect
 if(strcmp(response_check, "+HTTPACTION") == 0){
   Serial.println("UNABLE TO READ THE DESIRED LOCATION");
 }
 //If the response is not +HTTPACTION then the lecture will is
 //correct:
 else{
   Serial.println(response); //Prints the whole response
   //In the whole response, the actual values of latitude and
   //longitude start from position 17 of the array (this will depend
   //on the php code):
   if(response[17]=='-'){ //Checks if the latitude is negative
     if(response[19]=='.'){
     //If it's a number between 0 and -9.999999 store the next 9 //positions
from position 17 in the lat_read variable
       memcpy(lat_read, &response[17], 9);}
     if(response[20]=='.'){
       //If it's a number between -10.000000 and -90.000000 store the
       //next 10 positions from the position 17 in the lat_read
       //variable
       memcpy(lat_read, &response[17], 10);}
   }else{ //If the latitude is positive:
     if(response[18]=='.'){
       //if the number is between 0 and 9.999999 store the next 8
       //positions from the position 17 in the lat_read variable
       memcpy(lat_read, &response[17], 8 );}
     if(response[19]=='.'){
       //if the number is between 10.000000 and 90.000000 store the
       //next 9 positions from the position 17 in the lat_read //variable
       memcpy(lat_read, &response[17], 9 );
     }
   }
   //The longitude value will follow the '>' character (this will
   //depend on the php code):
   position=search_char_position(response,'>');
   //Searches the position of the '>' character within the response
   //array:
    //Checks if the longitude is negative:
   if(response[position+1]=='-'){
     if(response[position+3]=='.'){
       //If it's a number between 0 and -9.999999 store the next 9
       //positions from character '>' in the lat_read variable
       memcpy(lng_read, &response[position+1], 9 );
       //Reads the square side:
       memcpy(square_side_read,&response[position+14],1);}
     if(response[position+4]=='.'){
       //If it's a number between -10.000000 and -99.999999 store the
       //next 10 positions from character '>' in the lat_read variable
       memcpy(lng_read, &response[position+1], 10 );
       //Reads the square side:
       memcpy(square_side_read,&response[position+15],1);}
     if(response[position+5]=='.'){
     //If it's a number between -100.000000 and -180.000000 store the
     //next 11 positions from character '>' in the lat_read variable
       memcpy(lng_read, &response[position+1], 11 );
       //Reads the square side:
       memcpy(square_side_read,&response[position+16],1);}
   }else{
     //If the longitude is positive:
     if(response[position+2]=='.'){
       //If it's a number between 0 and 9.999999 store the next 8
       //positions from character '>' in the lat_read variable
       memcpy(lng_read, &response[position+1], 8 );
       //Reads the square side:
       memcpy(square_side_read,&response[position+13],1);}
```

```
      else if(response[position+3]=='.'){
        //If it's a number between 10.000000 and 99.999999 store the
        //next 10 positions from character '>' in the lat_read variable
        memcpy(lng_read, &response[position+1], 9 );
        //Reads the square side:
        memcpy(square_side_read,&response[position+14],1);}
      else if(response[position+4]=='.'){
        //If it's a number between 100.000000 and 180.000000 store the
        //next 11 positions from character '>' in the lat_read variable
        memcpy(lng_read, &response[position+1], 10 );
        //Reads the square side:
        memcpy(square_side_read,&response[position+15],1);}
    }

    //Convertion of the desired latitude from char array to float:
    float desired_lat_float=std::stof(lat_read);
    //Convertion of the desired longitude from char array to float:
    float desired_lng_float=std::stof(lng_read);
    //Convertion of the desired square side from char array to float:
    float desired_square_side_float=std::stof(square_side_read);
    Serial.print("Desired latitude float:");
    //Prints the desired latitude with 6 decimals digits:
    Serial.println(desired_lat_float,6);
    //Prints the desired latitude with 6 decimals digits:
    Serial.print("Desired longitude float:");
    Serial.println(desired_lng_float,6);
    //Prints the desired square side:
    Serial.print("Desired square side float:");
    Serial.println(desired_square_side_float);
    }
}

//Funtion to communicate between the SIM800L and the mobile app:
void communicationAppSIM()
{
    //Declaration of a latitude coordinate in degrees:
    String latitude = "-55.989737";
    //Declaration of a longitude coordinate in degrees:
    String longitude = "-77.233243";
    String warning = "Test warning";

    //Declaration of the url in which data will be sent to the database
    String urlSend;
    //The complete url will be obtained by adding different parameters:
    urlSend = "http://shipgpsserver.000webhostapp.com/add_current_loc.php?lat=";
  //This url is associtated with the server and the implemented php //code
    urlSend += latitude;
    urlSend += "&lng=";
    urlSend += longitude;
    urlSend += "&warnings=";
    urlSend += warning;
    //The url obtained will be //url="http://shipgpsserver.000webhostapp.com/
 add_current_loc.php?
    //lat=latitude&lng=longitude&warnings=Test warning"

    //Declaration of the url that contains the information that will
    //be read by the SIM800L:
    String urlRead;

     //This url is associtated with the server and the implemented php
     //code: urlRead="http://shipgpsserver.000webhostapp.com/display_desired_loc
 .php";

    sendAT("AT+HTTPINIT", "OK", 2000); //Initiates the HTTP service
    //Specifies the beared profile identifier:
    sendAT("AT+HTTPPARA=\"CID\",1", "OK", 1000);
    //Specifies the server's url that contains the information to be
    //read: "AT+HTTPPARA="URL",""http://shipgpsserver.000webhostapp.com/display_desired_loc.php""\r");
    SerialAT.print("AT+HTTPPARA=\"URL\",\"");
    SerialAT.print(urlRead);
    sendAT("\"", "OK", 1000);

    //Sets the GET HTTP method, if the response is 200 it means OK:
    sendAT("AT+HTTPACTION=0", "0,200", 1000);
    delay(1000);
    //It is important to leave some time between both commands in
    //order to obtain a proper lecture from the HTTPREAD command
    //Reads the content of website with a 10 second timeout:
    sendHHTPREAD("AT+HTTPREAD",10000);

    //Specifies the beared profile identifier:
    sendAT("AT+HTTPPARA=\"CID\",1", "OK", 1000);

    //Specifies the server's url with the coordinates that will be //sent: "AT+HTTPPARA="URL","http://shipgpsserver.000webhostapp.com/add_current_loc.php?lat=latitude&lng=longitude"\r");
    SerialAT.print("AT+HTTPPARA=\"URL\",\"");
    SerialAT.print(urlSend);
    sendAT("\"", "OK", 1000);

    //Sets the GET HTTP method, if the response is 200 it means OK:
    sendAT("AT+HTTPACTION=0", "0,200", 2000);
    sendAT("AT+HTTPTERM", "OK", 1000); //Terminates the HTTP service
}

//Setup function for the SIM800L module:
void setupGSM()
{
  Serial.println("Setup GSM...");

  //Initializes the SIM800L serial:
  SerialAT.begin (115200, SERIAL_8N1, ESP32_RX2_PIN, ESP32_TX2_PIN);
  delay(3000);

  //Prints the module information:
  Serial.println(modem.getModemInfo());
  delay(3000);

  //Initializes the modem by restarting it, it may take a while:
  if (!modem.restart())
  {
    //If the modem initalization fails prints a warning and software
    //resets the board:
    Serial.println("Restarting GSM\nModem failed");
    delay(10000);
    ESP.restart();
    return;
  }
  //Displays if the initialization was correct:
  Serial.println("Modem restart OK");

  //If your SIM card has a code, use modem.simUnlock(GSM_PIN) to
  //unlock it:
  /*if(modem.simUnlock(GSM_PIN))
  {
    Serial.println("Failed to unlock SIM");
    delay(10000);
    ESP.restart();
    return;
  }
  Serial.println("Sim unlock OK");*/

  //Establishes GPRS connection:
  if (!modem.gprsConnect(apn, gprsUser, gprsPass))
  {
    //If the GPRS connection fails prints a warning and software
    //resets the board:
    Serial.println("GPRS connection\nFailed");
    delay(10000);
    ESP.restart();
    return;
  }
  //Displays if the GPRS connection was correct:
  Serial.println("GPRS connect OK");
}

//Function that checks if the SIM800L  GPRS disconnects and tries to //reconnect it:
void verifyGPRSConnection()
{
  if (modem.isGprsConnected()) //Checks if the GPRS connection is OK
  {
    Serial.println("GPRS connected");}
  else //If it is disconnected:
  {
    Serial.println("GPRS disconnected");
    Serial.println("Reconnecting...");

    //Re-establishes GPRS connection:
    if (!modem.gprsConnect(apn, gprsUser, gprsPass))
    {
      Serial.println("GPRS connection failed"); //If the GPRS connection fails
      delay(5000);
    }
    else
    {
      //If the GPRS connection is re-established:
      Serial.println("GPRS connection re-established");
    }
  }
}

void setup()
{
  Serial.begin(9600);  //Initializates the usb serial
  pinMode(ONOFF_BAT_OUTPUT,OUTPUT);
  pinMode(BUTTON_DETECTOR_INPUT_PUSH_BUTTON,INPUT);

  //The ON/OFF pin must be set to HIGH as soon as the program starts:
  digitalWrite(ONOFF_BAT_OUTPUT,HIGH);
  delay(200);
  //The interruption will be executed when a rising egde is detected
  //on the BUTTON_DETECTOR_INPUT_PUSH_BUTTON pin:
  attachInterrupt(BUTTON_DETECTOR_INPUT_PUSH_BUTTON, turn_off, RISING);
  time_aux= millis();  //Initializates the auxiliar time variable
  setupGSM();          //Initializates the modem and establishes connection
}

void loop()
{
  //Every 10 seconds verifies the GPRS connection,
  //reads the server data and transmit data to the server
  if(millis()-time_aux>=20000){
    verifyGPRSConnection();
    communicationAppSIM();
    time_aux=millis();
  }
}
```

Listado C.16 – *ESP32 code to test the communication between the SIM800L and the mobile app*

C.13 State of charge estimation code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_INA219.h>

//GPIO that detects if the push button is pressed or not:
#define BUTTON_DETECTOR_INPUT_PUSH_BUTTON  23
//GPIO that sets the board state (on/off):
#define ONOFF_BAT_OUTPUT                   13
//Sampling time for the state of charge estimation in ms:
#define SAMPLING_TIME_MS                    1
//Sampling time for the state of charge estimation in hours:
#define SAMPLING_TIME_H             277.78E-9

float initial_charge=7000; //Initial battery charge in mAh
float current_battery_charge; //Battery charge at the current time
//Auxiliary variable for the charge estimation's timer:
unsigned long charge_timer_aux;
unsigned long print_timer_aux; //Auxiliary variable for the print timer
float battery_current_mA = 0;  //Battery's current in mA

//Interrupt rutine, it turns off the board:
void turn_off (void){
  digitalWrite(ONOFF_BAT_OUTPUT,LOW);
}

Adafruit_INA219 ina219;  //Declaration of the INA219 object

void setup() {
  //Power switch circuit initialization:
  pinMode(ONOFF_BAT_OUTPUT,OUTPUT);
  pinMode(BUTTON_DETECTOR_INPUT_PUSH_BUTTON,INPUT);
  digitalWrite(ONOFF_BAT_OUTPUT,HIGH);
  delay(200);
  attachInterrupt(BUTTON_DETECTOR_INPUT_PUSH_BUTTON, turn_off, RISING);

  Serial.begin(9600);   //Initializes the serial communication
  if (!ina219.begin())  //Checks if the INA219 sensor is detected:
    {
      //If it's not detected, print the following message and stop the
      //execution of the program:
      Serial.println("Failed to find INA219 chip");
      while (1);
    }
  charge_timer_aux=millis(); //Timer initialization
}

void loop() {
  //For every sample period:
  if(millis()-charge_timer_aux>=SAMPLING_TIME_MS){
    //Implements the state of charge equation:
    battery_current_mA = -(ina219.getCurrent_mA())*4;
    current_battery_charge=battery_current_mA*SAMPLING_TIME_MS+initial_charge;
    charge_timer_aux=millis(); //Updates the timer variable
  }

  //Displays the battery's current and charge every 500 ms:
  if(millis()-print_timer_aux>=500){
    Serial.print("Current: ");
    Serial.print(battery_current_mA);
    Serial.println(" mA");
    Serial.print("Battery charge: ");
    Serial.print(current_battery_charge);
    Serial.println(" mAh");
    print_timer_aux=millis(); //Updates the timer variable
  }
}
```

Listado C.17 – *Code to estimate the battery's state of charge*

C.14 Servo motor test code

```
#include <Arduino.h>
#include <ESP32_Servo.h>

#define SERVO_GPIO 25 //Servo GPIO of the ESP32

Servo servomotor;  //Creates a servo object

int servo_position = 0;//Stores the servo position

void setup() {
  servomotor.attach(SERVO_GPIO);   //Attaches the servo to the GPIO
}

void loop() {
  //Goes from 180 to 0 degrees in 1 degree steps:
  for(servo_position = 180; servo_position>= 0;servo_position-= 1) {
    servomotor.write(servo_position); //Sets the servo motor angle
    delay(15); //Waits 15ms for the servo to reach the position
  }
}
```

Listado C.18 – *Servo motor test code*

C.15 DC motor code

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

//Power switch definitions:
//GPIO that detects if the push button is pressed or not:
#define BUTTON_DETECTOR_INPUT_PUSH_BUTTON  23
//GPIO that sets the board state (on/off):
#define ONOFF_BAT_OUTPUT                   13

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

//Pin that controls the forward movement of the DC motor (PWMF pin in the
//ESP32 diagram):
#define DC_MOTOR_FORWARD_PIN  15
//Pin that controls the backward movement of the DC motor (PWMB pin in the
//ESP32 diagram):
#define DC_MOTOR_BACKWARD_PIN 14

//Creates an OLED display object connected to I2C:
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

//DC motor's state machine states:
```

```
typedef enum{
  INITIAL_PWM_STATE,
  OFF_TIME_PWM_STATE,
  ON_TIME_PWM_STATE
}DC_MOTOR_PWM_STATES;

//Directions of the DC motor:
typedef enum{
  BACKWARD=0,
  FORWARD=1
}DC_MOTOR_DIRECTION;

//DC motor states declaration:
DC_MOTOR_PWM_STATES dc_motor_pwm_state;

//Function to configure the DC motor operation:
bool dc_motor_set(int duty_cycle, int DC_MOTOR_DIRECTION)
{
  //PWM input that will be off or on:
  int on_pwm_input;
  int off_pwm_input;

  //Timer for the on and off operation of the motor:
  unsigned long dc_motor_on_timer;
  unsigned long dc_motor_off_timer;

  //Time in ms that the motor will be on:
  int on_time_PWM=85;
  //Time in ms that the motor will be off:
  int off_time_PWM;

  //If the duty cycle is out of this range, warn it:
  if(duty_cycle>100 || duty_cycle<0){
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print("Invalid duty cycle");
    oled.setCursor(22,20);
    oled.display();
    return false;
  }

  //If the motor direction is forward, associates the on input to the forward
 pin and the off input to the backward pin:
  if(DC_MOTOR_DIRECTION){
    int on_pwm_input=DC_MOTOR_FORWARD_PIN;
    int off_pwm_input=DC_MOTOR_BACKWARD_PIN;
  }else{//If the motor direction is forward, the opposite is true:
    int on_pwm_input=DC_MOTOR_BACKWARD_PIN;
    int off_pwm_input=DC_MOTOR_FORWARD_PIN;
  }

  //Initially sets both inputs to high:
  digitalWrite(off_pwm_input,HIGH);
  digitalWrite(on_pwm_input,HIGH);

  //Calculates the off time depending on the duty cycle:
  if(duty_cycle<100){
    off_time_PWM=on_time_PWM*(100/duty_cycle-1);
  }
  //If the duty cycle is 100%, it will be set to 99%:
  else if(duty_cycle==100){
    duty_cycle=99;
    off_time_PWM=on_time_PWM*(100/duty_cycle-1);
  }

  //Switches between the DC motor operation states:
  switch (dc_motor_pwm_state){
    case INITIAL_PWM_STATE:
      //Initializes the off timer and switches:
      dc_motor_off_timer=millis();
      dc_motor_pwm_state=OFF_TIME_PWM_STATE;

    case OFF_TIME_PWM_STATE:
      //Sets the on input to high and when the off timer ends switches to
      //the next state:
      digitalWrite(on_pwm_input,HIGH);
      if(millis()-dc_motor_off_timer>=off_time_PWM){
        dc_motor_on_timer=millis();
        dc_motor_pwm_state=ON_TIME_PWM_STATE;
      }
    //Sets the on input to low and when the on timer ends switches back to
    //the initial state:
    case ON_TIME_PWM_STATE:
      digitalWrite(on_pwm_input,LOW);
      if(millis()-dc_motor_on_timer>=on_time_PWM){
        dc_motor_pwm_state=INITIAL_PWM_STATE;
      }
  }
  return true;
}

void setup()
{
  //Power switch circuit;
  pinMode(ONOFF_BAT_OUTPUT,OUTPUT);
  pinMode(BUTTON_DETECTOR_INPUT_PUSH_BUTTON,INPUT);
  //H-bridge circuit:
  pinMode(DC_MOTOR_FORWARD_PIN,OUTPUT);
  pinMode(DC_MOTOR_BACKWARD_PIN,OUTPUT);
  //The ON/OFF pin must be set to HIGH as soon as the program starts:
  digitalWrite(ONOFF_BAT_OUTPUT,HIGH);
  //The PWM inputs must be set to 3.3 V at the beginning:
  digitalWrite(DC_MOTOR_FORWARD_PIN,HIGH);
  digitalWrite(DC_MOTOR_BACKWARD_PIN,HIGH);

  //Sets the inital state of the H-bridge state machine:
  dc_motor_pwm_state=INITIAL_PWM_STATE;

  //Initialize OLED display with I2C address 0x3C:
  if (!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)) {
    //If the initialization is not correct stops the program:
    Serial.println("Failed to start SSD1306 OLED");
    while (1);
  }
  delay(200);
  oled.clearDisplay(); //Clears display
}

void loop()
{
//H-bridge test with a 50% duty cycle and forward direction:
dc_motor_set(50,FORWARD);
}
```

Listado C.19 – *Code to control the DC motor through the H-bridge*

C.16 Full code

```
#include <Arduino.h>
```

```
#include <Wire.h>
#include <Adafruit_INA219.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include <Button.h>
#include <TinyGPSPlus.h>
#include <HardwareSerial.h>
#include <ESP32_Servo.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_LSM303.h>
#include <Adafruit_10DOF.h>
#include <ThreeWire.h>
#include <RtcDS1302.h>
#include <math.h>

//Definition of the module we are using, it must be defined before
//including the library:
#define TINY_GSM_MODEM_SIM800
#include <TinyGsmClient.h>

//SIM800L definitions:
#define ESP32_RX2_PIN        16 //RX GPIO of the ESP32's UART 2
#define ESP32_TX2_PIN        17 //TX GPIO of the ESP32's UART 2
 //Define your SIM card code if necessary so it can be unlocked:
#define GSM_PIN ""

//GY-NEO6MV2 definitions:
//RX GPIO of the ESP32's UART 1 (requires reassignment of pins):
#define ESP32_RX1_PIN        18
//TX GPIO of the ESP32's UART 1 (requires reassignment of pins):
#define ESP32_TX1_PIN        19

//Power switch definitions:
//GPIO that detects if the push button is pressed or not:
#define BUTTON_DETECTOR_INPUT_PUSH_BUTTON   23
//GPIO that sets the board state (on/off):
#define ONOFF_BAT_OUTPUT                    13
//Rotary definitions:
//GPIO connected to the A pin of the rotary encoder:
#define ROTARY_A_INPUT_PIN                   2
//GPIO connected to the B pin of the rotary encoder:
#define ROTARY_B_INPUT_PIN                  27
//GPIO connected to the button pin of the rotary encoder:
#define ROTARY_BUTTON_INPUT_PIN              5

//DC Motor definitions:
//GPIO that controls the forward movement of the DC motor (PWMF pin in
//the ESP32 diagram):
#define DC_MOTOR_FORWARD_PIN                15
//GPIO that controls the backward movement of the DC motor (PWMB pin //in the
 ESP32 diagram):
#define DC_MOTOR_BACKWARD_PIN               14

//Program upload LEDs definitions:
//GPIO that controls the 2 LEDs that notifies the program upload
#define UPLOAD_LEDS_OUTPUT_PIN       6

//Servo GPIO of the ESP32:
#define SERVO_GPIO                          25

//Front LED definition:
#define FRONT_LED                            9

//DS1302 definitions:
#define DS1302_DAT 12 //GPIO connected to the data pin
#define DS1302_CLK 33 //GPIO connected to the clock pin
```

```
#define DS1302_RST 32 //GPIO connected to the data reset

//Oled display sizes:
#define SCREEN_WIDTH 128 // OLED display width in pixels
#define SCREEN_HEIGHT 64 // OLED display height in pixels

//State of battery definitions:
//Sampling time for the state of charge estimation in ms:
#define SAMPLING_TIME_MS                    1
//Sampling time for the state of charge estimation in hours:
#define SAMPLING_TIME_H             277.78E-9

//DS1302 variables:
//Defines an instance of the three wire method:
ThreeWire myWire(DS1302_DAT,DS1302_CLK,DS1302_RST);
//Constructs a RTCDS1302 object using the ThreeWire instance:
RtcDS1302<ThreeWire> Rtc(myWire);

//Variables to store the date and time:
uint8_t current_hour, current_minute, month;
unsigned long rtc_timer; //Timer variable for the RTC

//SIM800L variables:
//Declaration of the GPRS credentials:
const char apn[]  = "airtelwap.es"; //Vodafone's APN
const char gprsUser[] = "";         //Only declare it if necessary
const char gprsPass[] = "";         //Only declare it if necessary
unsigned long sim800_time_aux;
float desired_lat_float; //Desired latitude lecture in float type
float desired_lng_float; //Desired longitude lecture in float type
float desired_square_side_float; //Desired square side in float type
//Desired latitude on the previous lecture:
float previous_desired_lat=0;
//Desired longitude on the previous lecture:
float previous_desired_lng=0;
//Side of thee square around the desired location [m]:
int square_side=0;
String warning = " ";

//Creates a HardwareSerial instance for the UART 2 :
HardwareSerial SerialAT(2);
//Creates a TinyGsm instance with the name of the UART 1 serial:
TinyGsm modem(SerialAT);

//GY-NEO6MV2 variables:
HardwareSerial neogps(1);
TinyGPSPlus gps;               //Creates a TinyGPSPlus instance
//Flag to indicate if a new gps data was read:
bool new_gps_data_flag = false;
double latitude_gps=0,longitude_gps=0;
double initial_latitude_gps=0,initial_longitude_gps=0;

Adafruit_INA219 ina219;  //Declaration of the INA219 object

//INA219 variables:
float busvoltage = 0;
float current_mA = 0;
unsigned long ina_timer;

//IMU variables:
//Assign a unique ID to the sensors:
Adafruit_10DOF                imu_sensor   =Adafruit_10DOF();
Adafruit_LSM303_Accel_Unified accelerometer=Adafruit_LSM303_Accel_Unified(30301)
 ;
Adafruit_LSM303_Mag_Unified magnetometer = Adafruit_LSM303_Mag_Unified(30302);
```

```
//Structures to provide a single sensor event in a common format:
sensors_event_t accelerometer_event;
sensors_event_t magnetometer_event;
sensors_vec_t   orientation;

//Variable to store the ship's compass:
int ship_compass;
//Variable to store the ship's roll:
int ship_roll;

// Creation of an OLED display object connected to I2C:
Adafruit_SSD1306 oled(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -1);

//Servo motor variables:
Servo servomotor;        //Creates a servo object
int servo_position = 70; //Stores the servo position

//DC motor's state machine states:
typedef enum{
  INITIAL_PWM_STATE,
  OFF_TIME_PWM_STATE,
  ON_TIME_PWM_STATE
}DC_MOTOR_PWM_STATES;

//Directions of the DC motor:
typedef enum{
  BACKWARD=0,
  FORWARD=1
}DC_MOTOR_DIRECTION;

//DC motor states declaration:
DC_MOTOR_PWM_STATES dc_motor_pwm_state;

//Navigation machine state:
typedef enum{
  NAVIGATION_STATE_1,
  NAVIGATION_STATE_2,
  NAVIGATION_STATE_3,
  NAVIGATION_STATE_4,
  NAVIGATION_STATE_5,
  NAVIGATION_STATE_6,
  NAVIGATION_STATE_7,
  NAVIGATION_STATE_8,
  NAVIGATION_STATE_9,
  NAVIGATION_STATE_10,
  NAVIGATION_STATE_11,
  NAVIGATION_STATE_12,
  NAVIGATION_STATE_13
}NAVIGATION_STATES;

//Navigation states declaration:
NAVIGATION_STATES navigation_state;

//Navigation variables
//Time that the boat takes to cover
//the distance of the square side:
unsigned long square_side_time, half_square_side_time, distance_time;
unsigned long square_side_timer_aux,half_square_side_timer_aux;
unsigned long distance_timer_aux;
const int boat_speed=0.31; //Boat speed with a 0.5 duty cycle[m/s]
float desired_orientation,longitude_distance,latitude_distance;
double desired_angle;
float distance;

//State of charge variables:
float initial_charge=7000;       //Initial battery charge in mAh
float current_battery_charge;    //Current battery charge in mAh
float battery_current_mA = 0;    //Battery's current in mA
//Auxiliary variable for the charge estimation's timer:
unsigned long charge_timer_aux;

//Power switch interrupt rutine, it turns off the board
void turn_off (void){
  digitalWrite(ONOFF_BAT_OUTPUT,LOW);
}

//Functions (they are all developed after the loop):
//H-bridge functions:
bool dc_motor_set(int duty_cycle, int DC_MOTOR_DIRECTION);
//State of charge functions:
void state_of_charge_estimation();
//DS1302 functions:
void DS1302_initialization();
void DS1302_time_check();
//GPS functions:
void communicate_gps();
void get_gps_data();
//IMU functions:
void IMU_initialization();
void get_boat_orientation();
//SIM800L functions:
void setupGSM();
void verifyGPRSConnection();
bool sendAT(const char* ATcommand, const char* expected_answer, unsigned int
 timeout);
void sendHHTPREAD(const char* ATcommand, unsigned int timeout);
int  communicationAppSIM(void);
int  search_char_position(char response[],char searched_character);
//Navigation functions:
void navigation(void);

void setup(){
  Serial.begin(9600);

  //Power switch circuit:
  pinMode(ONOFF_BAT_OUTPUT,OUTPUT);
  pinMode(BUTTON_DETECTOR_INPUT_PUSH_BUTTON,INPUT);
  //The ON/OFF pin must be set to HIGH as soon as the program starts:
  digitalWrite(ONOFF_BAT_OUTPUT,HIGH);
  //Necessary delay so the interrupt doesn't detect the first push of //the
 button:
  delay(200);
  //The interruption will be executed when a rising egde is detected //on the
 BUTTON_DETECTOR_INPUT_PUSH_BUTTON pin:
  attachInterrupt(BUTTON_DETECTOR_INPUT_PUSH_BUTTON, turn_off, RISING);

  //Rotary circuit:
  //Configures the input pins of the rotary as internal pull-ups:
  pinMode(ROTARY_A_INPUT_PIN,INPUT_PULLUP);
  pinMode(ROTARY_B_INPUT_PIN,INPUT_PULLUP);
  pinMode(ROTARY_BUTTON_INPUT_PIN,INPUT_PULLUP);

  //INA219 circuit:
  if(!ina219.begin()){ //Checks if the INA219 sensor is detected:
    //If it's not detected, print the following message and stop the
    //execution of the program:
    Serial.println("Failed to find INA219");
    while (1);}
    Serial.println("INA219 initialized");

  //IMU circuit:
  IMU_initialization();
  //OLED circuit:
  //Initialize OLED display with I2C address 0x3C:
  if(!oled.begin(SSD1306_SWITCHCAPVCC, 0x3C)){
    Serial.println(F("Failed to start SSD1306 OLED"));
    while (1);
  }

  delay(200);          // Waits two seconds for initializing
  oled.clearDisplay(); // Clear display

  //Servo motor circuit:
  pinMode(SERVO_GPIO,OUTPUT);
  servomotor.attach(SERVO_GPIO); //Attaches the servo to its GPIO

  //DC Motor circuit:
  pinMode(DC_MOTOR_FORWARD_PIN,OUTPUT);
  pinMode(DC_MOTOR_BACKWARD_PIN,OUTPUT);
  digitalWrite(DC_MOTOR_FORWARD_PIN,HIGH);
  digitalWrite(DC_MOTOR_BACKWARD_PIN,HIGH);
  //Sets the inital state of the H-bridge state machine:
  dc_motor_pwm_state=INITIAL_PWM_STATE;

  //Front LED:
  pinMode(FRONT_LED,OUTPUT);
  digitalWrite(FRONT_LED,LOW);

  //DS1302 initialization:
  DS1302_initialization();

  //SIM800L initialization:
  setupGSM();

  //GPS initialization:
  //Begins UART1 communication for the  GY-NEO6MV2:
  neogps.begin(9600, SERIAL_8N1, ESP32_RX1_PIN, ESP32_TX1_PIN);

  //Timer variables initialization:
  ina_timer=millis();
  sim800_time_aux=millis();
}

void loop() {
  //Obtains the battery current and voltage:
  busvoltage = ina219.getBusVoltage_V();
  //The current is multiplied by 4 because we are using 4 parallel
  //resistors:
  current_mA = 4*(ina219.getCurrent_mA());

  //If the current is too high, goes to the navigation state 13:
  if(current_mA>11000){
    navigation_state=NAVIGATION_STATE_13;
  }

  //Checks the time and turns on the front LED if necessary:
  DS1302_time_check();

  //Estimates the state of charge:
  state_of_charge_estimation();

  //Reads raw data from the GPS:
  communicate_gps();

  if(millis()-sim800_time_aux>=10000){
    verifyGPRSConnection();
    communicationAppSIM();
    sim800_time_aux=millis();
  }

  //Executes the navigation algorithm:
  navigation();
}

//------------------FUNCTIONS-----------------
//Function to configure the DC motor operation:
bool dc_motor_set(int duty_cycle, int DC_MOTOR_DIRECTION)
{
  //PWM input that will be off or on:
  int on_pwm_input;
  int off_pwm_input;

  //Timer for the on and off operation of the motor:
  unsigned long dc_motor_on_timer;
  unsigned long dc_motor_off_timer;

  //Time in ms that the motor will be on:
  int on_time_PWM=85;
  //Time in ms that the motor will be off:
  int off_time_PWM;

  //If the duty cycle is out of this range, warn it:
  if(duty_cycle>100 || duty_cycle<0){
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print("Invalid duty cycle");
    oled.setCursor(22,20);
    oled.display();
    return false;
  }

  //If the motor direction is forward, associates the on input to the
  //forward pin and the off input to the backward pin:
  if(DC_MOTOR_DIRECTION){
    int on_pwm_input=DC_MOTOR_FORWARD_PIN;
    int off_pwm_input=DC_MOTOR_BACKWARD_PIN;
  }else{//If the motor direction is forward, the opposite is true:
    int on_pwm_input=DC_MOTOR_BACKWARD_PIN;
    int off_pwm_input=DC_MOTOR_FORWARD_PIN;
  }

  //Initially sets both inputs to high:
  digitalWrite(off_pwm_input,HIGH);
  digitalWrite(on_pwm_input,HIGH);

  //Calculates the off time depending on the duty cycle:
  if(duty_cycle<100){
    off_time_PWM=on_time_PWM*(100/duty_cycle-1);
  }
  //If the duty cycle is 100%, it will be set to 99%:
  else if(duty_cycle==100){
    duty_cycle=99;
    off_time_PWM=on_time_PWM*(100/duty_cycle-1);
  }

  //Switches between the DC motor operation states:
  switch (dc_motor_pwm_state){
    case INITIAL_PWM_STATE:
      //Initializes the off timer and switches:
      dc_motor_off_timer=millis();
      dc_motor_pwm_state=OFF_TIME_PWM_STATE;
    case OFF_TIME_PWM_STATE:
      //Sets the on input to high and when the off timer ends switches
      //to the next state:
      digitalWrite(on_pwm_input,HIGH);
      if(millis()-dc_motor_off_timer>=off_time_PWM){
        dc_motor_on_timer=millis();
        dc_motor_pwm_state=ON_TIME_PWM_STATE;
        }
    //Sets the on input to low and when the on timer ends switches
    //back to the initial state:
    case ON_TIME_PWM_STATE:
      digitalWrite(on_pwm_input,LOW);
      if(millis()-dc_motor_on_timer>=on_time_PWM){
        dc_motor_pwm_state=INITIAL_PWM_STATE;
      }
  }
  return true;
}
//Function to estimate the battery state of charge:
void state_of_charge_estimation(){
  //For every sample period:
  if(millis()-charge_timer_aux>=SAMPLING_TIME_MS){
    //Implements the state of charge equation:
    battery_current_mA = -(ina219.getCurrent_mA())*4;
    current_battery_charge=battery_current_mA*SAMPLING_TIME_MS+initial_charge;
    charge_timer_aux=millis(); //Updates the timer variable
  }

  //If the battery capacity reaches 40%, notify the user and returns
  //to the initial position:
   if(current_battery_charge<=2800){
    warning="Low battery, returning";
    desired_lat_float=initial_latitude_gps;
    desired_lng_float=initial_longitude_gps;
  }

  //If the battery capacity reaches 25%, notify the user:
  if(current_battery_charge<=1750){
    warning="Boat will turn off soon";
  }

  //If the battery capacity reaches 20%, turns the board off:
  if(current_battery_charge<=1400){
    digitalWrite(ONOFF_BAT_OUTPUT,LOW);
  }
}
//Function to initializate the DS1302 RTC:
void DS1302_initialization(){
    Rtc.Begin(); //RTC library initialization
    //Creates a RtcDateTime instance from the compilation date and
    //time:
    RtcDateTime compiled = RtcDateTime(__DATE__, __TIME__);
    Serial.println(); //Prints an empty line

    //Checks if the date and time values of the RTC are valid:
    if(!Rtc.IsDateTimeValid()){
        //If they are not valid, prints a warning and sets the RTC
        //time to the time and date of compilation:
        Serial.println("RTC lost confidence in the DateTime!");
        Rtc.SetDateTime(compiled);
    }
    //Checks if the RTC was write protected:
    if (Rtc.GetIsWriteProtected()){
        //If it was, prints a warning en disables the write protection:
        Serial.println("RTC was write protected, enabling writing now");
        Rtc.SetIsWriteProtected(false);
    }
    //Checks if the clock is running on the RTC:
    if (!Rtc.GetIsRunning()){
        //If it wasn't, prints a warning a starts running it:
        Serial.println("RTC was not actively running, starting now");
        Rtc.SetIsRunning(true);
    }
    //Creates a RTCDateTime instance and stores the the RTC's date
    //and time:
    RtcDateTime now = Rtc.GetDateTime();

    //Cheks if the RTC time is older than the compiled time:
    if (now < compiled){
        //If that's the case, prints a warning a sets the RTC time to
        //the time and date of compilation:
        Serial.println("RTC is older than compile time!  (Updating DateTime)");
        Rtc.SetDateTime(compiled);
    }
    //Checks if the RTC time is newer than the compiled time:
    else if (now > compiled){
        //If that's the case, prints a notice:
        Serial.println("RTC is newer than compile time. (this is expected)");
    }
    //Checks if the RTC time is the same as the compiled time:
    else if (now == compiled)
    {
        //If that's the case, prints a notice (it is unlikely but
        //it's not a problem):
        Serial.println("RTC is the same as compile time! (not expected but all
 is fine)");
    }
    rtc_timer=millis();
}
//Function to control the front LED:
void DS1302_time_check(){
  if(millis()-rtc_timer>=60000){
    //Updates the time and date of the now instance:
    RtcDateTime now = Rtc.GetDateTime();
    //Stores the date and time in their corresponding date and time
    //variables:
    month=now.Month();
    current_hour=now.Hour();
    current_minute=now.Minute();
    rtc_timer=millis();
  }

  if(month>=10 || month<=3){
    if(current_hour>=18 || current_hour<=9){
      digitalWrite(FRONT_LED,HIGH);
    }else{
      digitalWrite(FRONT_LED,LOW);
    }
  }
  else if(current_hour>=20 || current_hour<=8){
    digitalWrite(FRONT_LED,HIGH);
  }else{
    digitalWrite(FRONT_LED,LOW);
  }
}
//Function to read raw data from the GPS module:
void communicate_gps(){
  //During one second, if the UART1 is available it will read the GPS
  //data:
  for(unsigned long gps_timer_aux=millis();millis()-gps_timer_aux<1000;){
    while (neogps.available()){
      //Reads and decodes the raw NMEA data to GPS coordinates:
      if(gps.encode(neogps.read())){
        new_gps_data_flag = true; //Updates de gps flag
      }
    }
  }
  //If a new GPS data has been received, calls the print_gps_data
  //function:
  if(new_gps_data_flag){
    new_gps_data_flag = false; //Updates gps flag
    get_gps_data();
  }
  //If no data has been received, prints a warning:
  else{
    oled.clearDisplay();
    oled.setTextColor(SSD1306_WHITE);
    oled.setCursor(2, 5);
    oled.setTextSize(1);
    oled.print("Warning: no GPS data");
    oled.display();
  }
}
//Function that gets the GPS latitude and longitude:
void get_gps_data(){
  //If the gps location is valid, prints the latitude and longitude:
  if(gps.location.isValid()){

    if(latitude_gps==0 && longitude_gps==0){
      initial_latitude_gps=gps.location.lat();
      initial_longitude_gps=gps.location.lng();
      latitude_gps=initial_latitude_gps;
      longitude_gps=initial_longitude_gps;
    }else{
    latitude_gps=gps.location.lat();
    longitude_gps=gps.location.lng();
    }
  }
  //If the GPS location is not valid, warn it:
  else{
    oled.setCursor(2, 5);
    oled.setTextSize(1);
    oled.print("Invalid location");
    oled.display();
  }
}
//Function to initialize the magnetometer and accelerometer:
void IMU_initialization()
{
  //Initializes the accelerometer:
  if(!accelerometer.begin())
  {
    //If it is not detected, shows a warning and stops the program:
    Serial.println(F("LSM303 acceleromter not detected"));
    while(1);
  }
  //Initializes the magnetometer:
  if(!magnetometer.begin())
  {
    //If it is not detected, shows a warning and stops the program:
    Serial.println("LSM303 magnetometer not detected");
    while(1);
  }
}
//Function to get the boat's orientation through the IMU:
void get_boat_orientation(){
  //Gets a new accelerometer event:
```

```
  accelerometer.getEvent(&accelerometer_event);
  //Gets the roll and the pitch:
  if(imu_sensor.accelGetOrientation(&accelerometer_event, &orientation))
  {
    //If the operation is correct, gets the roll:
    ship_roll=orientation.roll;
    //If the roll is less than 16  :
    if(ship_roll<=15){
      navigation_state=NAVIGATION_STATE_12;
    }

  }
  //Gets a new magnetometer event:
  magnetometer.getEvent(&magnetometer_event);
  //Gets the boat's heading:
  if (imu_sensor.magGetOrientation(SENSOR_AXIS_Y, &magnetometer_event, &
 orientation)){
    //Subtracts 90 to the heading so it is equal to the compass' angle:
    ship_compass=orientation.heading-90;
    //Adjusts the compass value so it ranges from 0 to 360  :
    if (ship_compass<0){
      ship_compass = 360 + ship_compass;
    }
  }
}
//Setup function for the SIM800L module:
void setupGSM(){
  Serial.println("Setup GSM...");

  //Initializes the SIM800L serial:
  SerialAT.begin (115200, SERIAL_8N1, ESP32_RX2_PIN, ESP32_TX2_PIN);
  delay(3000);

  //Prints the module information:
  Serial.println(modem.getModemInfo());
  delay(3000);

  //Initializes the modem by restarting it, it may take a while:
  if (!modem.restart()){
    //If the modem initalization fails prints a warning and software
    //resets the board:
    Serial.println("Restarting GSM\nModem failed");
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print("Modem failed, restart");
    oled.display();
    delay(10000);
    ESP.restart();
    return;
  }
  //Displays if the initialization was correct:
  Serial.println("Modem restart OK");
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print("Restart OK");
    oled.display();

  //If your SIM card has a code, use modem.simUnlock(GSM_PIN) to
  //unlock it:
  /*if(modem.simUnlock(GSM_PIN)){
    Serial.println("Failed to unlock SIM");
    delay(10000);
    ESP.restart();
    return;
  }
  Serial.println("Sim unlock OK");*/

  //Establishes GPRS connection:
  if (!modem.gprsConnect(apn, gprsUser, gprsPass)){
    //If the GPRS connection fails prints a warning and software
    //resets the board:
    Serial.println("GPRS connection\nFailed");
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print("GPRS failed");
    oled.display();
    delay(10000);
    ESP.restart();
    return;
  }
  //Displays if the GPRS connection was correct:
  Serial.println("GPRS connect OK");
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(22,6);
    oled.print("GPRS OK");
    oled.display();
}
//Function that checks if the SIM800L  GPRS disconnects and tries to //reconnect
  it:
void verifyGPRSConnection(){
  //Checks if the GPRS connection is OK:
  if (modem.isGprsConnected()){
    Serial.println("GPRS connected");}
  else{ //If it is disconnected:
    Serial.println("GPRS disconnected");
    Serial.println("Reconnecting...");
    //Re-establishes GPRS connection:
    if(!modem.gprsConnect(apn, gprsUser, gprsPass)){
      //If the GPRS connection fails:
      Serial.println("GPRS connection failed");
    }
    else{
      //If the GPRS connection is re-established:
      Serial.println("GPRS connection re-established");
    }
  }
}
//Function that sends the specified AT command with a timeout and //checks if
 the response is correct:
bool sendAT(const char* ATcommand, const char* expected_answer, unsigned int
 timeout){

    uint8_t i=0;            //Auxiliar variable for the do while loop
    //Flag variable to indicate if the command was answered:
    bool answerFlag=0;
    //Char array that stores the answer to the command:
    char answer[100];
    //Auxiliar timer variable that stores the time before the do while
    //loop started:
    unsigned long previous;

    //Initializes the answer array to null characters:
    memset(answer, '\0', 100);
    //Cleans the input buffer:
```

```
    while( SerialAT.available() > 0) SerialAT.read();
    delay(100);

    //Checks if the AT command's first character is not null has
    //content and sends it:
    if (ATcommand[0] != '\0'){
      SerialAT.println(ATcommand); //Send the AT command
    }

    previous=millis(); //Stores the time at which the do loop started

    //Reads serial data if the serial is available until an answer is
    //received or the timeout is reached:
    do{
       //Checks if there is data in the UART input buffer, if there is
       //it reads it and compare it with the expected answer:
        if(SerialAT.available() != 0){
            //Each character is stored in each of the array's position:
            answer[i] = SerialAT.read();
            //The position in the array is increased each iteration of
            //the loop:
            i++;
            //If the desired answer is received it is indicated
            //through the answer flag
            if(strstr(answer, expected_answer) != NULL){
                answerFlag = 1;
            }
        }
    }
    //Checks if the expected answer is received or if the timeout has
    //been reached:
    while((answerFlag == 0) && ((millis() - previous) < timeout));

  Serial.println(answer); //Prints the answer
  return answerFlag; //Returns the answer flag
}
//Specific function to send the HHTPREAD comment and extract the data:
void sendHHTPREAD(const char* ATcommand, unsigned int timeout){

    uint8_t i=0;              //Auxiliar variable for the do while loop
    char response[100];       //Char array that stores the response to the
 command
    char response_check[12]; //Part of the response array that will be used to
 check the response to the command
    char lat_read[11];        //Desired latitude lecture
    char lng_read[12];        //Desired longitude lecture
    char square_side_read[2];//Desired square side lecture
    unsigned long previous;  //Auxiliar timer variable that stores the time
 before the do while loop started
    int position;             //Position of a specific character within an char
 array
    previous_desired_lat=desired_lat_float; //Stores the last desired latitude
 lecture
    previous_desired_lng=desired_lng_float; //Stores the last desired longitude
 lecture

    //Initializes the char arrays to null characters
    memset(response, '\0', 100);
    memset(response_check,'\0',12);
    memset(lat_read,'\0',11);
    memset(lng_read,'\0',12);
    memset(square_side_read,'\0',2);

    while( SerialAT.available() > 0) SerialAT.read(); //Cleans the input buffer
    delay(100);
  //Checks if the AT command's first character is not null has content and
sends it:
  if (ATcommand[0] != '\0'){
   SerialAT.println(ATcommand); //Sends the AT command
  }

  previous = millis(); //Stores the time at which the do loop started

  //Reads serial data if the serial is available until the timeout is reached:
  do{
    //Checks if there is data in the UART input buffer, if there is it reads
 it and compare it:
    if(SerialAT.available() != 0){
      response[i] = SerialAT.read(); //Each character is stored in each of
 the array's position
      i++; //The position in the array is increased each iteration of the
loop
    }
  }while(((millis() - previous) < timeout)); //Checks if the timeout has been
reached

 memcpy(response_check,&response[8],11); //Stores 11 characters from the 8th
position of the response into the response_check array
 if(strcmp(response_check, "+HTTPACTION") == 0){ //Checks if the response is +
HTTPACTION, if that is the case it is reading the previous command's response
and the coordenates' lecure will be incorrect
  Serial.println("UNABLE TO READ THE DESIRED LOCATION");
  return;
 }
 else{ //If the response is not +HTTPACTION then the lecture will is correct:
  Serial.println(response); //Prints the whole response
  //In the whole response, the actual values of latitude and longitude start
from position 17 of the array (this will depend on the php code):
  if(response[17]=='-'){ //Checks if the latitude is negative
   if(response[19]=='.'){ //If it's a number between 0 and -9.999999 store
the next 9 positions from position 17 in the lat_read variable
    memcpy(lat_read, &response[17], 9);}
   if(response[20]=='.'){ //If it's a number between -10.000000 and
-90.000000 store the next 10 positions from the position 17 in the lat_read
variable
    memcpy(lat_read, &response[17], 10);}
  }else{ //If the latitude is positive:
   if(response[18]=='.'){ //if the number is between 0 and 9.999999 store the
 next 8 positions from the position 17 in the lat_read variable
    memcpy(lat_read, &response[17], 8 );}
   if(response[19]=='.'){ //if the number is between 10.000000 and 90.000000
store the next 9 positions from the position 17 in the lat_read variable
    memcpy(lat_read, &response[17], 9 );
   }
  }
  //The longitude value will follow the '>' character (this will depend on the
 php code):
  position=search_char_position(response,'>'); //Searches the position of the
'>' character within the response array
  if(response[position+1]=='-'){ //Checks if the longitude is negative
   if(response[position+3]=='.'){ //If it's a number between 0 and -9.999999
store the next 9 positions from character '>' in the lat_read variable
    memcpy(lng_read, &response[position+1], 9 );
    memcpy(square_side_read,&response[position+14],1);}
   if(response[position+4]=='.'){ //If it's a number between -10.000000 and
-99.999999 store the next 10 positions from character '>' in the lat_read
variable
    memcpy(lng_read, &response[position+1], 10 );
    memcpy(square_side_read,&response[position+15],1);}
   if(response[position+5]=='.'){ //If it's a number between -100.000000 and
-180.000000 store the next 11 positions from character '>' in the lat_read
variable
      memcpy(lng_read, &response[position+1], 11 );
      memcpy(square_side_read,&response[position+16],1);}
    }else{ //If the longitude is poitive:
      if(response[position+2]=='.'){ //If it's a number between 0 and 9.999999 store the next 8 positions from character '>' in the lat_read variable
        memcpy(lng_read, &response[position+1], 8 );
        memcpy(square_side_read,&response[position+13],1);}
      else if(response[position+3]=='.'){ //If it's a number between 10.000000 and 99.999999 store the next 10 positions from character '>' in the lat_read variable
        memcpy(lng_read, &response[position+1], 9 );
        memcpy(square_side_read,&response[position+14],1);}
      else if(response[position+4]=='.'){ //If it's a number between 100.000000 and 180.000000 store the next 11 positions from character '>' in the lat_read variable
        memcpy(lng_read, &response[position+1], 10 );
        memcpy(square_side_read,&response[position+15],1);}
    }

    float desired_lat_float = std::stof(lat_read); //Convertion of the desired latitude from char array to float
    float desired_lng_float = std::stof(lng_read); //Convertion of the desired longitude from char array to float
    float desired_square_side_float=std::stof(square_side_read); //Convertion of the desired square side from char array to float
    Serial.print("Desired latitude float:");
    Serial.println(desired_lat_float,6); //Prints the desired latitude with 6 decimals digits
    Serial.print("Desired longitude float:");
    Serial.println(desired_lng_float,6); //Prints the desired latitude with 6 decimals digits
    Serial.print("Desired square side float:");
    Serial.println(desired_square_side_float); //Prints the desired square side
    }
}
//Funtion to communicate between the SIM800L and the mobile app:
int communicationAppSIM()
{
    String latitude = String(latitude_gps,6);  //Declaration of a latitude coordinate in degrees
    String longitude = String(longitude_gps,6); //Declaration of a longitude coordinate in degrees

    String urlSend; //Declaration of the url in which data will be sent to the database
    //The complete url will be obtained by adding different parameters:
    urlSend = "http://shipgpsserver.000webhostapp.com/add_current_loc.php?lat="; //This url is associtated with the server and the implemented php code
    urlSend += latitude;
    urlSend += "&lng=";
    urlSend += longitude;
    urlSend += "&warnings=";
    urlSend += warning;
    //The url obtained will be url="http://shipgpsserver.000webhostapp.com/add_current_loc.php?lat=latitude&lng=longitude&warnings=Test warning"

    String urlRead; //Declaration of the url that contains the information that will be read by the SIM800L
    urlRead="http://shipgpsserver.000webhostapp.com/display_desired_loc.php"; // This url is associtated with the server and the implemented php code

    sendAT("AT+HTTPINIT", "OK", 2000); //Initiates the HTTP service
    sendAT("AT+HTTPPARA=\"CID\",1", "OK", 1000); //Specifies the beared profile identifier

    //Specifies the server's url that contains the information to be read: "AT+HTTPPARA="URL",""http://shipgpsserver.000webhostapp.com/display_desired_loc.php""\r");
    SerialAT.print("AT+HTTPPARA=\"URL\",\"");
    SerialAT.print(urlRead);
    sendAT("\"", "OK", 1000);

    sendAT("AT+HTTPACTION=0","0,200", 1000); //Sets the GET HTTP method, if the response is 200 it means OK

    delay(1000); //It is important to leave some time between both commands in order to obtain a proper lecture from the HTTPREAD command
    sendHHTPREAD("AT+HTTPREAD",10000); //Reads the content of website with a 10 second timeout

    sendAT("AT+HTTPPARA=\"CID\",1","OK", 1000); //Specifies the beared profile identifier

    //Specifies the server's url with the coordinates that will be sent: "AT+HTTPPARA="URL","http://shipgpsserver.000webhostapp.com/add_current_loc.php?lat=latitude&lng=longitude"\r");
    SerialAT.print("AT+HTTPPARA=\"URL\",\"");
    SerialAT.print(urlSend);
    sendAT("\"","OK",1000);

    sendAT("AT+HTTPACTION=0","0,200",2000); //Sets the GET HTTP method, if the response is 200 it means OK
    sendAT("AT+HTTPTERM","OK",1000); //Terminates the HTTP service

    return 1;
}
//Function that return the position of certain character in a char array:
int search_char_position(char response[],char searched_character){

  int length = strlen(response); //Obtains the number of characters of the array
   for(int i=0; i<length; i++) { //For each position of the array checks if the character is equal to the one searched
      if(response[i] == searched_character) {
         return i; //If the character is contained in the array it returns its position
      }
   }
  return -1; //If the character isn't contained in the array it returns -1
}
//Navigation algorithm:
void navigation(void){
  //The state 2 is executed if a new desired location is read:
  if(desired_lat_float!=previous_desired_lat || desired_lng_float!=previous_desired_lng){
    navigation_state=NAVIGATION_STATE_2;
  }

  //Defines the maximum position error, which corresponds to 2 m approx.
  const float maximum_position_error=0.00002;
  //Defines the maximum position error when describing a square around the position, which corresponds to 10 m approx.
  const float maximum_square_error=0.00009;

  switch (navigation_state){
    case NAVIGATION_STATE_1:
      //Sets the rudder in the middle position:
      servo_position=70;
      servomotor.write(servo_position);
      //The boat does not move:
      dc_motor_set(0,FORWARD);
      //If a new desired location is read, switches to state 2:
```

```
      if(desired_lat_float!=previous_desired_lat || desired_lng_float!=previous_desired_lng){
        navigation_state=NAVIGATION_STATE_2;
      }

    case NAVIGATION_STATE_2:
      //Checks if the boat is not the desired position with a 2 m error range:
      if(abs(latitude_gps-desired_lat_float)>maximum_position_error || abs(longitude_gps-desired_lng_float)>maximum_position_error){
        //If it is not around the desired position, switches to state 6:
        navigation_state=NAVIGATION_STATE_6;
      }else{
        //Calculates the time it takes to cover half of the square side around the position:
        half_square_side_time=square_side/2/boat_speed;
        //Switches to state 3:
        navigation_state=NAVIGATION_STATE_3;
        //Initalizates the timer:
        half_square_side_timer_aux=millis();
      }

    //The boat moves from the desired position to the center of one of the square sides:
    case NAVIGATION_STATE_3:
      //Sets the rudder in the middle position:
      servo_position=70;
      servomotor.write(servo_position);
      //Turns the DC motor on with a 50% duty cycle:
      dc_motor_set(50,FORWARD);
      //If the time finishes, starts turning the boat:
      if(millis()-half_square_side_timer_aux>=half_square_side_time){
        //Switches to state 4:
        navigation_state=NAVIGATION_STATE_4;
        //Turns the DC motor on with a 10% duty cycle:
        dc_motor_set(10,FORWARD);
        //Turns the rudder to one of the sides:
        servo_position=0;
        servomotor.write(servo_position);
        //Subtracts 90 to the compass to turn the boat 90  :
        desired_orientation=ship_compass-90;
        //Adjusts the orientation value so it ranges from 0 to 360  :
        if(desired_orientation<0){
          desired_orientation=360+desired_orientation;
        }
      }
      //If it is too far from the desired position, switches to state 9:
      if(abs(latitude_gps-desired_lat_float)>maximum_square_error || abs(longitude_gps-desired_lng_float)>maximum_square_error){
        navigation_state=NAVIGATION_STATE_9;
      }

    //Turns 90  and starts moving straight:
    case NAVIGATION_STATE_4:
      servo_position=0;
      servomotor.write(servo_position);
      dc_motor_set(10,FORWARD);
      //If the desired orientation is reached within a 2 degrees error:
      if(abs(desired_orientation-ship_compass)<=2){
        //Makes the boat move straight:
        servo_position=70;
        servomotor.write(servo_position);
        dc_motor_set(50,FORWARD);
        half_square_side_time=square_side/2/boat_speed;
        navigation_state=NAVIGATION_STATE_5;
        half_square_side_timer_aux=millis();
      }
```

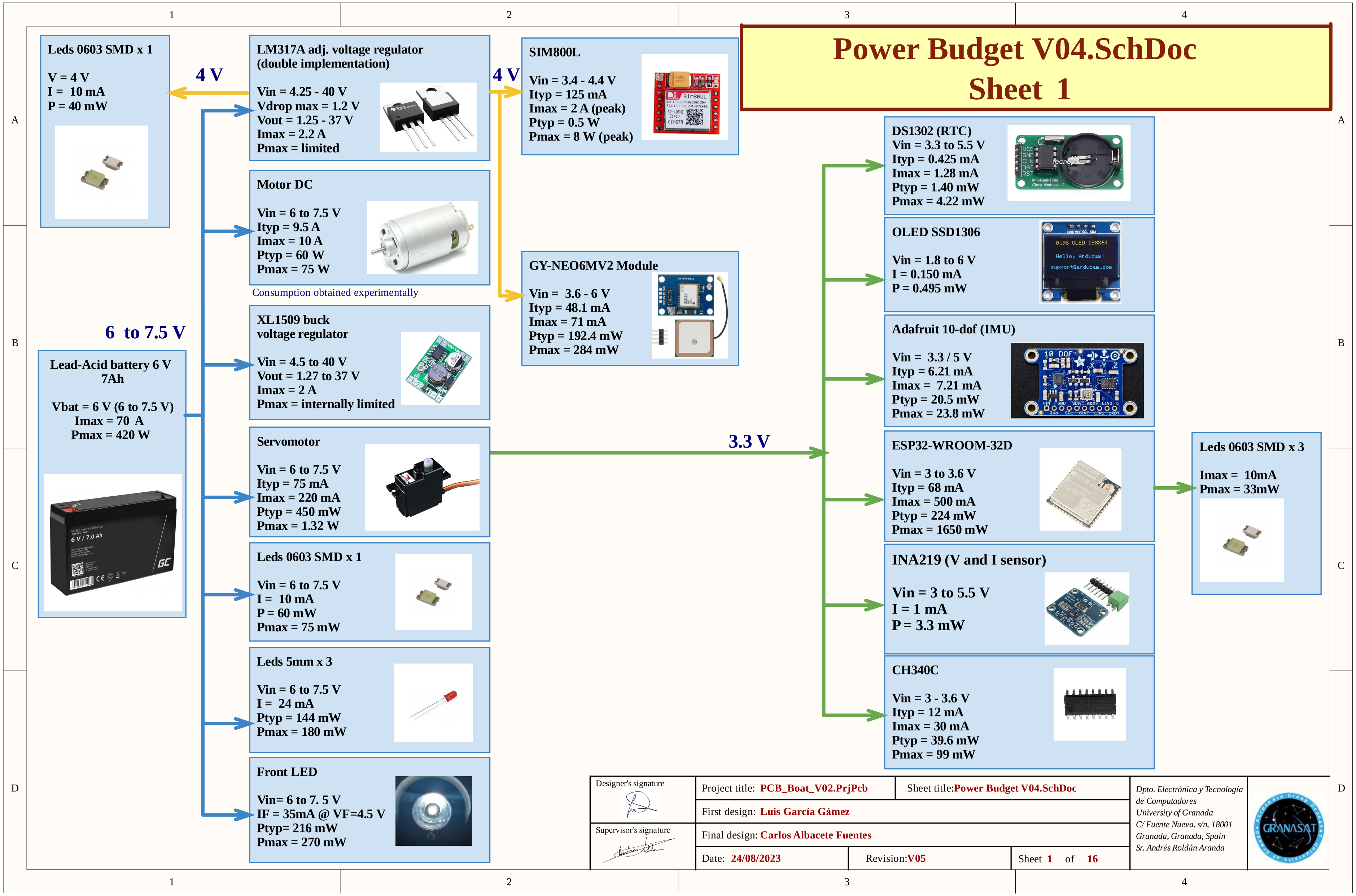
```
     //If it is too far from the desired position, switches to state 9:
     if(abs(latitude_gps-desired_lat_float)>maximum_square_error || abs(
longitude_gps-desired_lng_float)>maximum_square_error){
       navigation_state=NAVIGATION_STATE_9;
     }

   //The boat goes from the half of the square side to one of the square
corners:
   case NAVIGATION_STATE_5:
     servo_position=70;
     servomotor.write(servo_position);
     dc_motor_set(50,FORWARD);
     //If the time to cover half of the square side ends, starts turning the
boat:
     if(millis()-half_square_side_timer_aux>=half_square_side_time){
       navigation_state=NAVIGATION_STATE_6;
       dc_motor_set(10,FORWARD);
       servo_position=0;
       servomotor.write(servo_position);
       //Subtracts 90 to the compass to turn the boat 90  :
       desired_orientation=ship_compass-90;
       //Adjusts the orientation value so it ranges from 0 to 360  :
       if(desired_orientation<0){
         desired_orientation=360+desired_orientation;
       }
     }
     //If it is too far from the desired position, switches to state 9:
     if(abs(latitude_gps-desired_lat_float)>maximum_square_error || abs(
longitude_gps-desired_lng_float)>maximum_square_error){
       navigation_state=NAVIGATION_STATE_9;
     }

     //Turns 90  and starts moving straight:
     case NAVIGATION_STATE_6:
     servo_position=0;
     servomotor.write(servo_position);
     dc_motor_set(10,FORWARD);
     //If the desired orientation is reached within a 2 degrees error:
     if(abs(desired_orientation-ship_compass)<2){
       servo_position=70;
       servomotor.write(servo_position);
       dc_motor_set(50,FORWARD);
       //Calculates the time to cover the square side:
       square_side_time=square_side/boat_speed;
       navigation_state=NAVIGATION_STATE_7;
       square_side_timer_aux=millis();
     }
     //If it is too far from the desired position, switches to state 9:
     if(abs(latitude_gps-desired_lat_float)>maximum_square_error || abs(
longitude_gps-desired_lng_float)>maximum_square_error){
       navigation_state=NAVIGATION_STATE_9;
     }

     //The boat moves from one corner to the next one of the square:
     case NAVIGATION_STATE_7:
       servo_position=70;
       servomotor.write(servo_position);
       dc_motor_set(50,FORWARD);
       //If the time to cover the square side ends, starts turning the boat:
       if(millis()-square_side_timer_aux>=square_side_time){
         navigation_state=NAVIGATION_STATE_8;
         dc_motor_set(10,FORWARD);
         servo_position=0;
         servomotor.write(servo_position);
         //Subtracts 90 to the compass to turn the boat 90  :
         desired_orientation=ship_compass-90;
         //Adjusts the orientation value so it ranges from 0 to 360  :
         if(desired_orientation<0){
           desired_orientation=360+desired_orientation;
         }
       }
       //If it is too far from the desired position, switches to state 9:
       if(abs(latitude_gps-desired_lat_float)>maximum_square_error || abs(
longitude_gps-desired_lng_float)>maximum_square_error){
         navigation_state=NAVIGATION_STATE_9;
       }

     //Turns 90  , starts moving straight and goes back to state 7 unless it
deviates too much from the desired location
     //or a new desired location is specified:
       case NAVIGATION_STATE_8:
       servo_position=0;
       servomotor.write(servo_position);
       dc_motor_set(10,FORWARD);
       //If the desired orientation is reached within a 2 degrees error:
       if(abs(desired_orientation-ship_compass)<2){
         servo_position=70;
         servomotor.write(servo_position);
         dc_motor_set(50,FORWARD);
         //Calculates the time to cover the square side:
         square_side_time=square_side/boat_speed;
         navigation_state=NAVIGATION_STATE_7;
         square_side_timer_aux=millis();
       }
       //If it is too far from the desired position, switches to state 9:
       if(abs(latitude_gps-desired_lat_float)>maximum_square_error || abs(
longitude_gps-desired_lng_float)>maximum_square_error){
         navigation_state=NAVIGATION_STATE_9;
       }

     //If the boat is not around the desired location, calculates the angle and
 distance to it:
     case NAVIGATION_STATE_9:
       //Calculates the distance in the x and y axis:
       latitude_distance=desired_lat_float-latitude_gps;
       longitude_distance=desired_lng_float-longitude_gps;
       //Desired boat's angle calculation in sexagesimal degrees:
       desired_angle=atan((double)longitude_distance/latitude_distance)*180/
M_PI;
       //If the desired location is in the first quadrant:
       if(latitude_distance>0 && longitude_distance>0){
         desired_orientation=90-desired_angle;
       }
       //If the desired location is in the second quadrant:
       else if(latitude_distance<0 && longitude_distance>0){
         desired_orientation=270-desired_angle;
       }
       //If the desired location is in the third quadrant:
       else if(latitude_distance<0 && longitude_distance<0){
         desired_orientation=270-desired_angle;
       }
       //If the desired location is in the fourth quadrant:
       else if(latitude_distance>0 && longitude_distance<0){
         desired_orientation=90-desired_angle;
       }
       //Calculates the distance in a straight line to the desired location:
       distance=sqrt(latitude_distance*latitude_distance+longitude_distance*
longitude_distance);
       servo_position=0;
       servomotor.write(servo_position);
       dc_motor_set(10,FORWARD);
       navigation_state=NAVIGATION_STATE_10;
```

```

      //The boat is turned until it reaches the necessary orientation:
      case NAVIGATION_STATE_10:
        servo_position=0;
        servomotor.write(servo_position);
        dc_motor_set(10,FORWARD);
        //If the desired orientation is reached within a 2 degrees error:
        if(abs(desired_orientation-ship_compass)<2){
          servo_position=70;
          servomotor.write(servo_position);
          dc_motor_set(50,FORWARD);
          //Calculates the time it takes to reach the place:
          distance_time=distance/boat_speed;
          navigation_state=NAVIGATION_STATE_11;
          distance_timer_aux=millis();
        }

      //Moves the boat to the desired location and goes back to state 2:
      case NAVIGATION_STATE_11:
        servo_position=70;
        servomotor.write(servo_position);
        dc_motor_set(50,FORWARD);
        if(millis()-distance_timer_aux>=distance_time){
          navigation_state=NAVIGATION_STATE_2;
        }

      //State where the boat has capsized:
      case NAVIGATION_STATE_12:
        servo_position=70;
        servomotor.write(servo_position);
        dc_motor_set(0,FORWARD);
        warning="Boat capsized";
        if(ship_roll>15){
          navigation_state=NAVIGATION_STATE_2;
        }

      //State where the boat is stopped due to an overcurrent:
      case NAVIGATION_STATE_13:
      dc_motor_set(0,FORWARD);
      warning="Current is too high";
  }
}
```

Listado C.20 – *First complete version of the boat’s firmware*

Appendix D

Electronics schematics



Power Budget V04.SchDoc
Sheet 1

Leds 0603 SMD x 1
V = 4 V
I = 10 mA
P = 40 mW

Lead-Acid battery 6 V 7Ah
Vbat = 6 V (6 to 7.5 V)
Imax = 70 A
Pmax = 420 W

LM317A adj. voltage regulator (double implementation)
Vin = 4.25 - 40 V
Vdrop max = 1.2 V
Vout = 1.25 - 37 V
Imax = 2.2 A
Pmax = limited

Motor DC
Vin = 6 to 7.5 V
Ityp = 9.5 A
Imax = 10 A
Ptyp = 60 W
Pmax = 75 W

Consumption obtained experimentally

XL1509 buck voltage regulator
Vin = 4.5 to 40 V
Vout = 1.27 to 37 V
Imax = 2 A
Pmax = internally limited

Servomotor
Vin = 6 to 7.5 V
Ityp = 75 mA
Imax = 220 mA
Ptyp = 450 mW
Pmax = 1.32 W

Leds 0603 SMD x 1
Vin = 6 to 7.5 V
I = 10 mA
P = 60 mW
Pmax = 75 mW

Leds 5mm x 3
Vin = 6 to 7.5 V
I = 24 mA
Ptyp = 144 mW
Pmax = 180 mW

Front LED
Vin= 6 to 7. 5 V
IF = 35mA @ VF=4.5 V
Ptyp= 216 mW
Pmax = 270 mW

SIM800L
Vin = 3.4 - 4.4 V
Ityp = 125 mA
Imax = 2 A (peak)
Ptyp = 0.5 W
Pmax = 8 W (peak)

GY-NEO6MV2 Module
Vin = 3.6 - 6 V
Ityp = 48.1 mA
Imax = 71 mA
Ptyp = 192.4 mW
Pmax = 284 mW

DS1302 (RTC)
Vin = 3.3 to 5.5 V
Ityp = 0.425 mA
Imax = 1.28 mA
Ptyp = 1.40 mW
Pmax = 4.22 mW

OLED SSD1306
Vin = 1.8 to 6 V
I = 0.150 mA
P = 0.495 mW

Adafruit 10-dof (IMU)
Vin = 3.3 / 5 V
Ityp = 6.21 mA
Imax = 7.21 mA
Ptyp = 20.5 mW
Pmax = 23.8 mW

ESP32-WROOM-32D
Vin = 3 to 3.6 V
Ityp = 68 mA
Imax = 500 mA
Ptyp = 224 mW
Pmax = 1650 mW

INA219 (V and I sensor)
Vin = 3 to 5.5 V
I = 1 mA
P = 3.3 mW

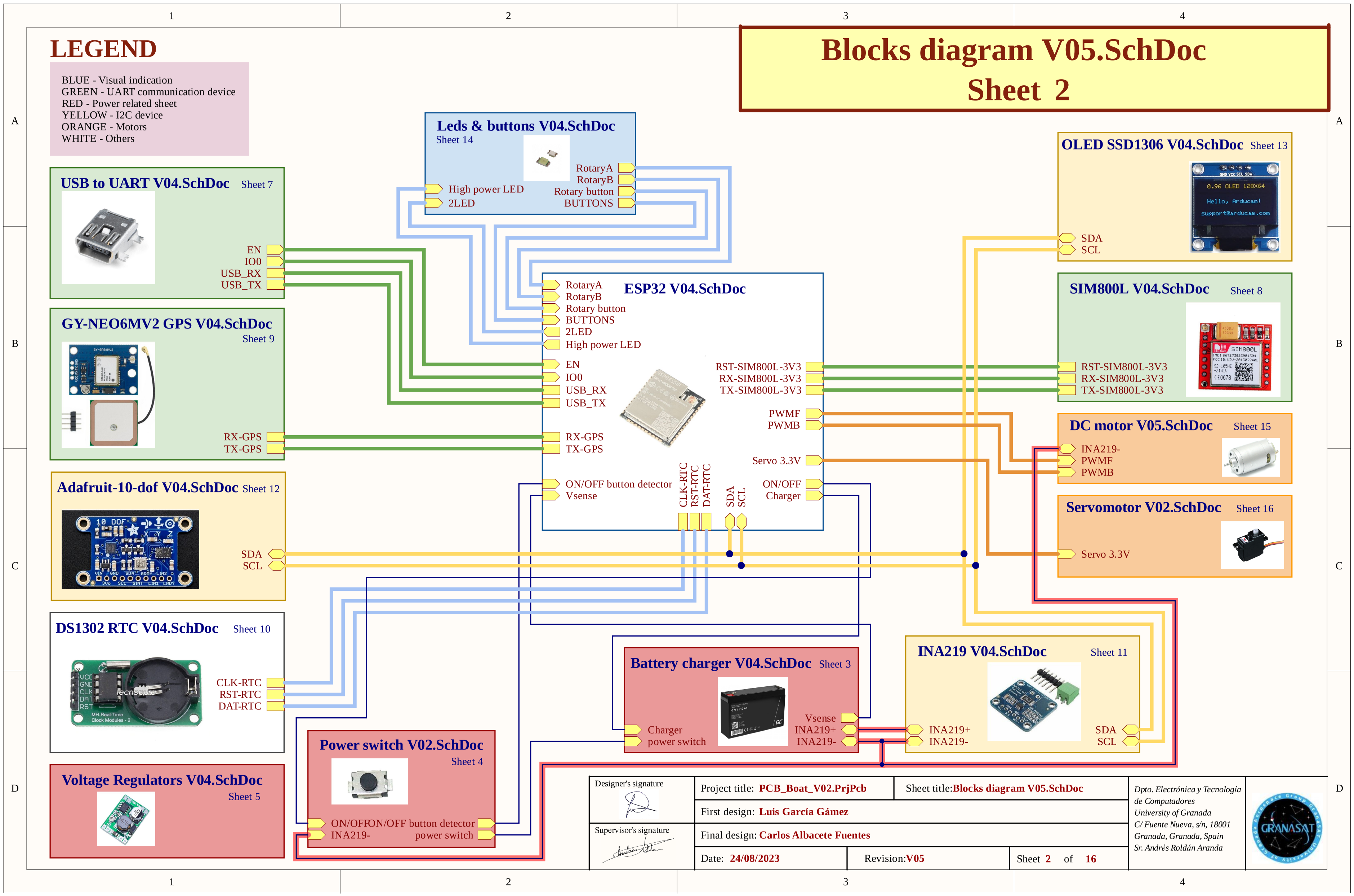
CH340C
Vin = 3 - 3.6 V
Ityp = 12 mA
Imax = 30 mA
Ptyp = 39.6 mW
Pmax = 99 mW

Leds 0603 SMD x 3
Imax = 10mA
Pmax = 33mW

Designer's signature
Supervisor's signature

Project title: PCB_Boat_V02.PrjPcb
Sheet title:Power Budget V04.SchDoc
First design: Luis García Gámez
Final design: Carlos Albacete Fuentes
Date: 24/08/2023
Revision:V05
Sheet 1 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda


Blocks diagram V05.SchDoc
Sheet 2
LEGEND
BLUE - Visual indication
GREEN - UART communication device
RED - Power related sheet
YELLOW - I2C device
ORANGE - Motors
WHITE - Others
USB to UART V04.SchDoc Sheet 7
EN
IO0
USB_RX
USB_TX
GY-NEO6MV2 GPS V04.SchDoc Sheet 9
RX-GPS
TX-GPS
Adafruit-10-dof V04.SchDoc Sheet 12
SDA
SCL
DS1302 RTC V04.SchDoc Sheet 10
CLK-RTC
RST-RTC
DAT-RTC
Voltage Regulators V04.SchDoc Sheet 5
Leds & buttons V04.SchDoc Sheet 14
High power LED
2LED
RotaryA
RotaryB
Rotary button
BUTTONS
ESP32 V04.SchDoc
RotaryA
RotaryB
Rotary button
BUTTONS
2LED
High power LED
EN
IO0
USB_RX
USB_TX
RX-GPS
TX-GPS
ON/OFF button detector
Vsense
CLK-RTC
RST-RTC
DAT-RTC
SDA
SCL
RST-SIM800L-3V3
RX-SIM800L-3V3
TX-SIM800L-3V3
PWMF
PWMB
Servo 3.3V
ON/OFF
Charger
OLED SSD1306 V04.SchDoc Sheet 13
SDA
SCL
SIM800L V04.SchDoc Sheet 8
RST-SIM800L-3V3
RX-SIM800L-3V3
TX-SIM800L-3V3
DC motor V05.SchDoc Sheet 15
INA219-
PWMF
PWMB
Servomotor V02.SchDoc Sheet 16
Servo 3.3V
Battery charger V04.SchDoc Sheet 3
Charger
power switch
Vsense
INA219+
INA219-
INA219 V04.SchDoc Sheet 11
INA219+
INA219-
SDA
SCL
Power switch V02.SchDoc Sheet 4
ON/OFFON/OFF button detector
INA219-
power switch
Designer's signature
Supervisor's signature
Project title: PCB_Boat_V02.PrjPcb
Sheet title:Blocks diagram V05.SchDoc
First design: Luis García Gámez
Final design: Carlos Albacete Fuentes
Date: 24/08/2023
Revision:V05
Sheet 2 of 16
Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda
GRANASAT
1
2
3
4
A
B
C
D

ESP32 V04.SchDoc
Sheet 3

ESP32-WROOM-32D

Vin = 3 - 3.6 V
Ityp = 68 mA
Imax = 500 mA
Ptyp = 224 mW
Pmax = 1650 mW

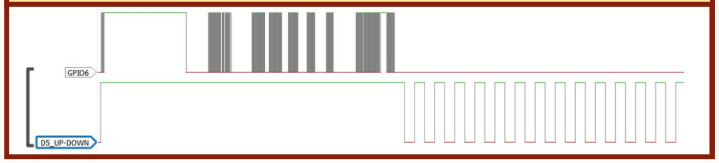
GPIOs 34 to 39 can only be configured as analog inputs

PWMF and PWMB control the forward and backward operation of the DC motor respectively.

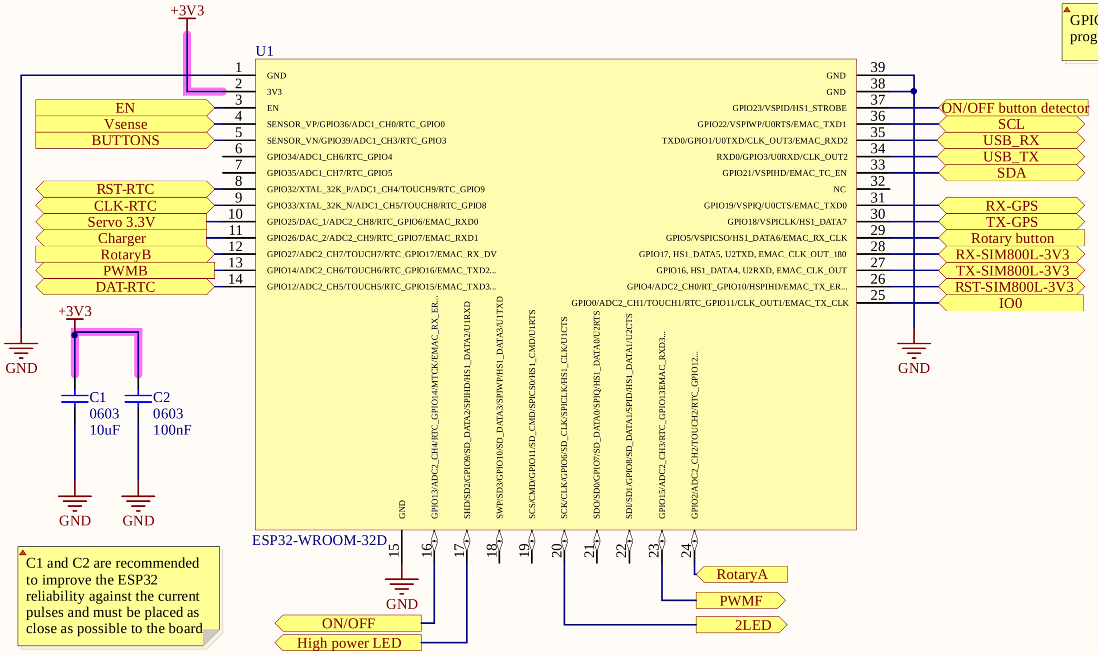
Charger: Analog signal to control the MOSFET's gate

Servo: PWM signal controlled by the servo library.

GPIO6 behaviour during and after the program upload

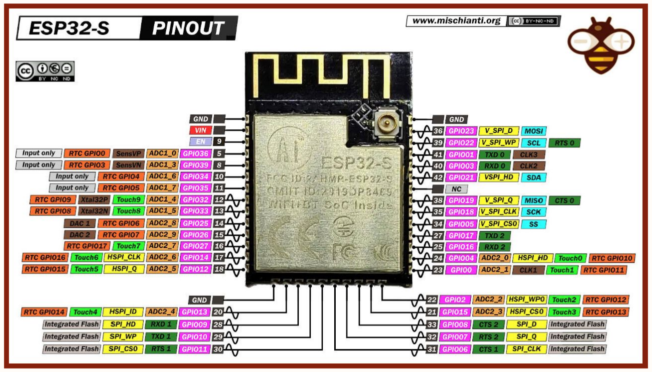


GPIOs 6 to 11 will toggle their output voltage during the program upload. Then, the 2LED pin will notify the program upload and can be used for other purposes.


+3V3
U1
1 GND
2 3V3
3 EN
4 SENSOR_VP/GPIO36/ADC1_CH0/RTC_GPIO0
5 SENSOR_VN/GPIO39/ADC1_CH3/RTC_GPIO3
6 GPIO34/ADC1_CH6/RTC_GPIO4
7 GPIO35/ADC1_CH7/RTC_GPIO5
8 GPIO32/XTAL_32K_P/ADC1_CH4/TOUCH9/RTC_GPIO9
9 GPIO33/XTAL_32K_N/ADC1_CH5/TOUCH8/RTC_GPIO8
10 GPIO25/DAC_1/ADC2_CH8/RTC_GPIO6/EMAC_RXD0
11 GPIO26/DAC_2/ADC2_CH9/RTC_GPIO7/EMAC_RXD1
12 GPIO27/ADC2_CH7/TOUCH7/RTC_GPIO17/EMAC_RX_DV
13 GPIO14/ADC2_CH6/TOUCH6/RTC_GPIO16/EMAC_TXD2...
14 GPIO12/ADC2_CH5/TOUCH5/RTC_GPIO15/EMAC_TXD3...
15 GND
16 GPIO13/ADC2_CH4/RTC_GPIO14/MTCK/EMAC_RX_ER...
17 SHD/SD2/GPIO9/SD_DATA2/SPIHD/HS1_DATA2/U1RXD
18 SWP/SD3/GPIO10/SD_DATA3/SPIWP/HS1_DATA3/U1TXD
19 SCS/CMD/GPIO11/SD_CMD/SPICS0/HS1_CMD/U1RTS
20 SCK/CLK/GPIO6/SD_CLK/SPICLK/HS1_CLK/U1CTS
21 SDO/SD0/GPIO7/SD_DATA0/SPIQ/HS1_DATA0/U2RTS
22 SDI/SD1/GPIO8/SD_DATA1/SPID/HS1_DATA1/U2CTS
23 GPIO15/ADC2_CH3/RTC_GPIO13EMAC_RXD3...
24 GPIO2/ADC2_CH2/TOUCH2/RTC_GPIO12...
25 GPIO0/ADC2_CH1/TOUCH1/RTC_GPIO11/CLK_OUT1/EMAC_TX_CLK
26 GPIO4/ADC2_CH0/RT_GPIO10/HSPIHD/EMAC_TX_ER...
27 GPIO16, HS1_DATA4, U2RXD, EMAC_CLK_OUT
28 GPIO17, HS1_DATA5, U2TXD, EMAC_CLK_OUT_180
29 GPIO5/VSPICSO/HS1_DATA6/EMAC_RX_CLK
30 GPIO18/VSPICLK/HS1_DATA7
31 GPIO19/VSPIQ/U0CTS/EMAC_TXD0
32 NC
33 GPIO21/VSPIHD/EMAC_TC_EN
34 RXD0/GPIO3/U0RXD/CLK_OUT2
35 TXD0/GPIO1/U0TXD/CLK_OUT3/EMAC_RXD2
36 GPIO22/VSPIWP/U0RTS/EMAC_TXD1
37 GPIO23/VSPID/HS1_STROBE
38 GND
39 GND
ESP32-WROOM-32D
EN
Vsense
BUTTONS
RST-RTC
CLK-RTC
Servo 3.3V
Charger
RotaryB
PWMB
DAT-RTC
ON/OFF button detector
SCL
USB_RX
USB_TX
SDA
RX-GPS
TX-GPS
Rotary button
RX-SIM800L-3V3
TX-SIM800L-3V3
RST-SIM800L-3V3
IO0
RotaryA
PWMF
2LED
ON/OFF
High power LED
GND
GND
GND
GND
GND
+3V3
C1
0603
10uF
C2
0603
100nF

C1 and C2 are recommended to improve the ESP32 reliability against the current pulses and must be placed as close as possible to the board

The ESP32-WROOM-32D contains an on-board antenna that should be placed outside the board to reduce interferences.

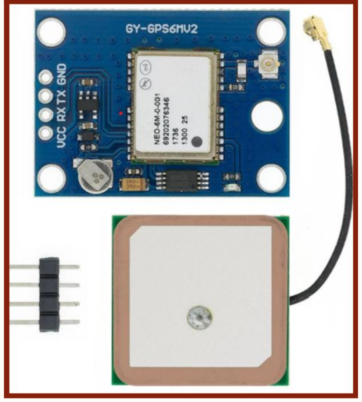


GPIOs 6 to 11 (pins 17 to 22) of the ESP32 are connected to the integrated SPI flash so they are not recommended to be used. During the upload these pins will output multiple voltage pulses and the user will not have control over them. However, once the upload is finished the will behave as regular output pins.
These must never be configured as input pins as it could generate a short circuit.
It is important not to use these pins to control the H bridge since they could shortcircuit its branches during the program upload when the user has no control over these pins' outputs.

Designer's signature	Project title: PCB_Boat_V02.PrjPcb	Sheet title:ESP32 V04.SchDoc	Dpto. Electrónica y Tecnología de Computadores University of Granada C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain Sr. Andrés Roldán Aranda
	First design: Luis García Gámez		
Supervisor's signature	Final design: Carlos Albacete Fuentes		
	Date: 24/08/2023 Revision:V05	Sheet 3 of 16	

GY-NEO6MV2 GPS V04.SchDoc
Sheet 4

GY-NEO6MV2 Module



It is important to distinguish between the GY-NEO6MV2 module, which is the module that will be implemented in this circuit and the NEO-6Q/M module, which is the actual GPS module that is contained inside the GY-NEO6MV2 module.

NEO-6Q/M Module

GY-NEO6MV2 module

Vin = 3.6 - 6 V
Ityp = 48.1 mA
Imax = 71 mA
Ptyp = 192.4 mW
Pmax = 284 mW

NEO-6Q/M

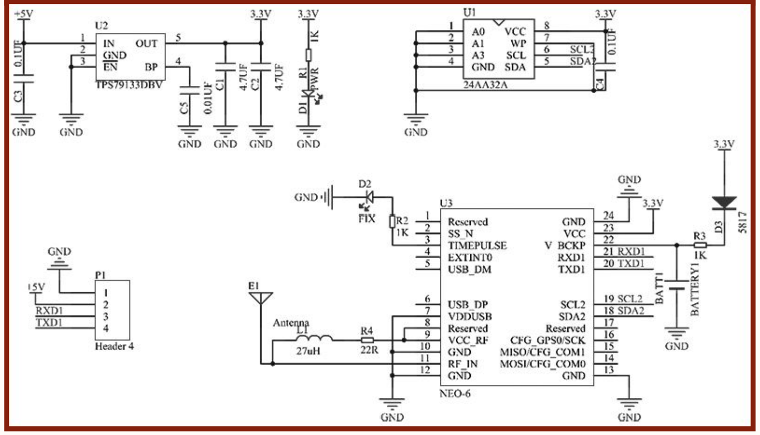
Vin = 2.7 - 3.6 V
Ityp= 47 mA
Imax = 67 mA
Ptyp = 155 mW
Pmax = 221 mW

GY-NEO6MV2 board features the u-blox NEO-6M GPS module with antenna and built-in EEPROM.

Ceramic antenna
EEPROM for saving the configuration data when powered off
Backup battery
LED signal indicator
TTL level, compatible with 3.3V/5V system.

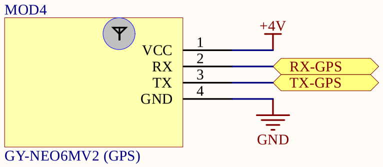
The backup battery is useful to keep the device connected to the GPS satellites. This connection might take up to 20 minutes to be established so keeping it active can save a lot of time when the device turns on.

GY-NEO6MV2 Module's schematic



The ceramic antenna will be placed in the auxiliary board of the PCB in order to be in a plane that is parallel to the geographical horizon.

The ceramic antenna must be placed facing towards the sky to receive satellite signals.



The GPS antenna can be active or passive but it requires a gain between 15 and 50 dB and a maximum noise figure of 1.5 dB.

To ensure a proper operation, this module will be supplied from the 4 V regulator instead of 3.3 V.

This shematic should only be used as a reference since it was not provided by the manufacturer. Some of the components differ from the actual ones.

Designer's signature
Supervisor's signature

Project title: PCB_Boat_V02.PrjPcb
Sheet title:GY-NEO6MV2 GPS V04.SchDoc
First design: Luis García Gámez
Final design: Carlos Albacete Fuentes
Date: 24/08/2023
Revision:V05
Sheet 4 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda

SIM800L V04.SchDoc
Sheet 5

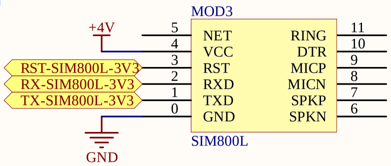
SIM800L module

Vin = 3.4 - 4.4 V
Ityp = 125 mA
Imax = 2 A (peak)
Ptyp = 0.5 W
Pmax = 8 W (peak)

The SIM800L is a quad-band GSM/GPRS module. It has a microSIM slot, antenna for the network signal, microphone, speaker pin outs and ring.

In our project, it will allow data transmission between the user and the boat through a mobile app.

This module is very sensitive regarding its power supply and will not operate if the 2 A current peaks cannot be precisely provided.



Level shifters from last year's project have been removed as the SIM800L module can operate with logic levels that range from 3 to 5V.

The typical consumption of the SIM800L has been estimated experimentally by measuring the current during its different operations.

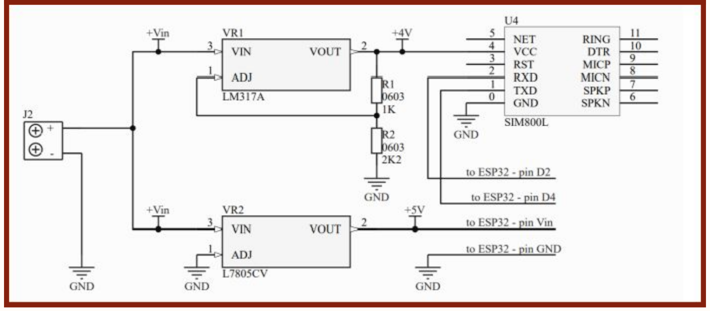
We could estimate the average current as the consumption when it is operating in the HTTP services, approximately 125 mA.

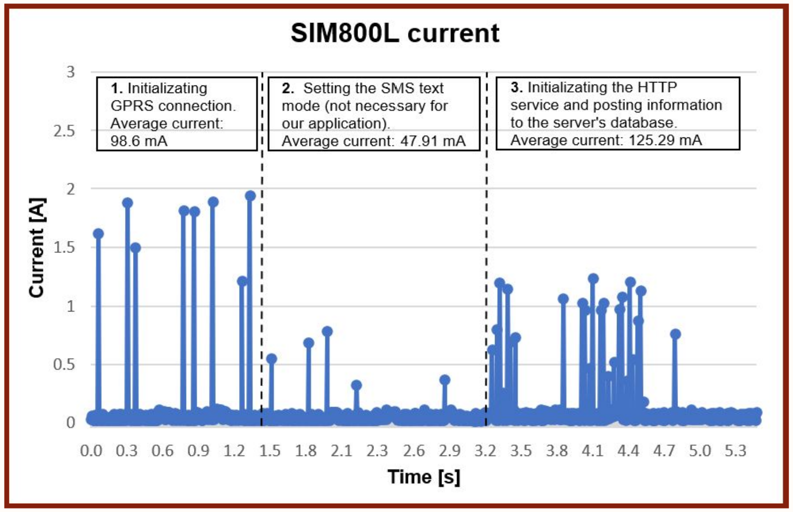
It can reach 2 A current peaks but this value will not be used estimate the battery duration.

SIM800L Module

SIM800L test circuit

SIM800L test circuit schematic





1. Initializating GPRS connection. Average current: 98.6 mA

2. Setting the SMS text mode (not necessary for our application). Average current: 47.91 mA

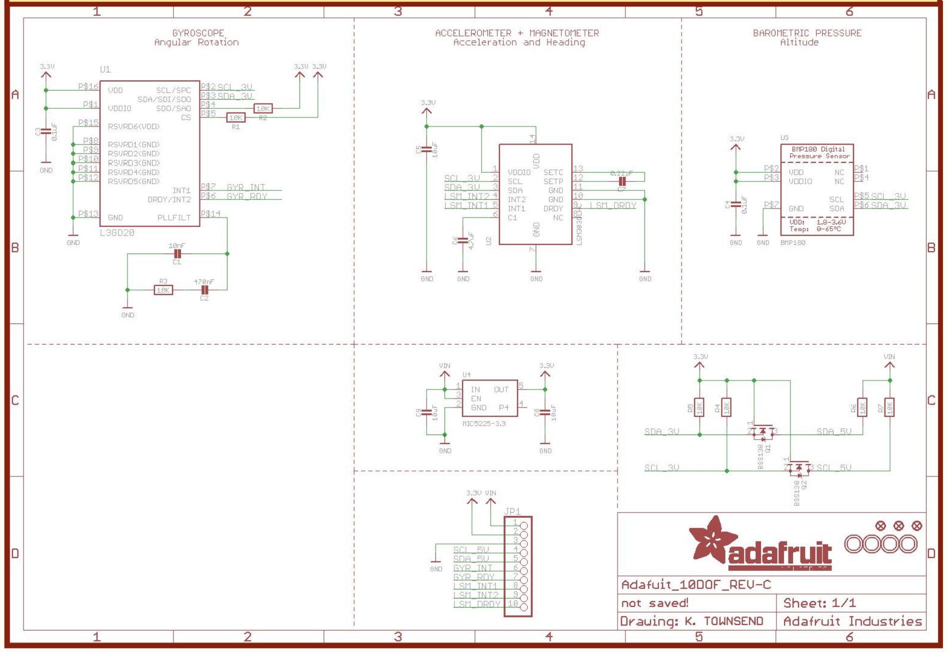
3. Initializating the HTTP service and posting information to the server's database. Average current: 125.29 mA

Designer's signature	Project title: PCB_Boat_V02.PrjPcb	Sheet title:SIM800L V04.SchDoc
	First design: Luis García Gámez	
Supervisor's signature	Final design: Carlos Albacete Fuentes	
	Date: 24/08/2023 Revision:V05	Sheet 5 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda

Adafruit-10-dof V04.SchDoc
Sheet 6

Adafruit 10 DOF schematic



Adafruit 10 DOF module

All the I2C devices include 10 kΩ pull-up resistors, so it is necesary to remove some of them from the main board. Otherwise its actual value would be too low since they are in paralel.

The pull-up resistors should only be left in the component that is further from the I2C pins of the ESP32, which is the Adafruit 10-DOF.

Capacitors are not necessary (all of them already have capacitor between VCC and GND integrated in main board)

A level shifter might be necesary to change the voltage of all the SCL and SDA signals to 3.3 V (some might be 5V). However, as we supply all the devices with 3.3 V, it won't be necessary.

Adafruit 10-DOF (IMU)

Vin = 3.3 / 5 V
Ityp = 6.21 mA
Imax = 7.21 mA
Ptyp = 20.5 mW
Pmax = 23.8 mW

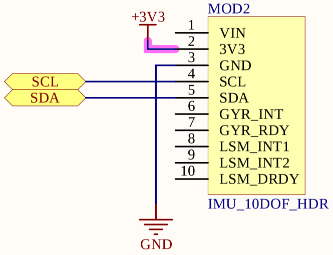
Adafruit's 10DOF (10 Degrees of Freedom) breakout board allows you to capture ten distinct types of motion or orientation related data.

It has several differents measurement devices:

LSM303DLHC - a 3-axis accelerometer (up to +/-16g) and a 3-axis magnetometer (up to +/-8.1 gauss) on a single die.

L3GD20 - a 3-axis gyroscope (up to +/-2000 dps).

BMP180 - A barometric pressure sensor (300 to 1100 hPa) that can be used to calculate altitude, with an additional on-board temperature sensor.



The interruption pins will only be added if there is enough free pins in the ESP32 as they are not strictly necessary.

In this prototype we are short on pins so they will not be used.

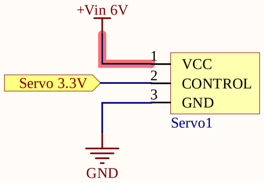
Designer's signature	Project title: **PCB_Boat_V02.PrjPcb**	Sheet title:**Adafruit-10-dof V04.SchDoc**	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	First design: **Luis García Gámez**		
Supervisor's signature	Final design: **Carlos Albacete Fuentes**		
	Date: **24/08/2023**	Revision:**V05**	Sheet **6** of **16**

Servomotor V02.SchDoc
Sheet 7

The servo has three 0.08 mm2 cables that will be connected to the board through a 3 pin header of 2.54 mm.

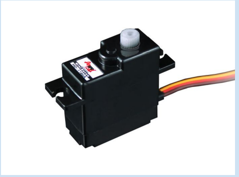
It will be supplied at the battery's voltage, so the XL1509 regulator that provided 5 V will be removed.

The level shifters from the previous design have been removed as it has checked experimentally that the servo motor supports 3.3 V logic level.

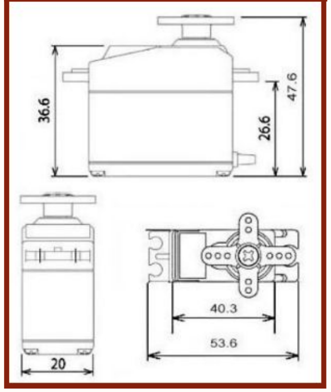


Servomotor

Vin = 6 V
Ityp = 75 mA
Imax = 220 mA
Ptyp = 450 mW
Pmax = 1.32 W

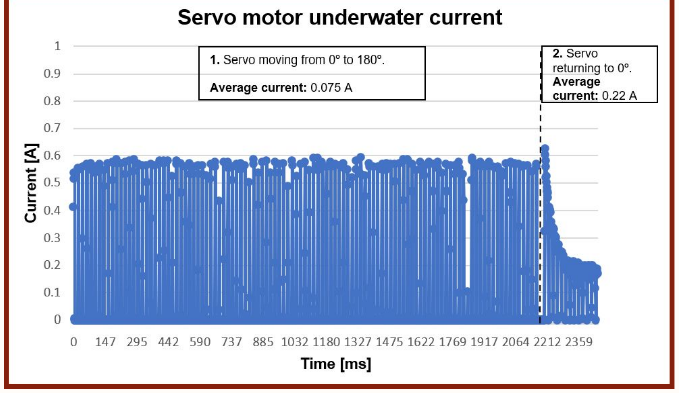


Servo dimensions



The boat's servo motor dimensions are the same as the MG996R servo.

Servo's current during typical operation

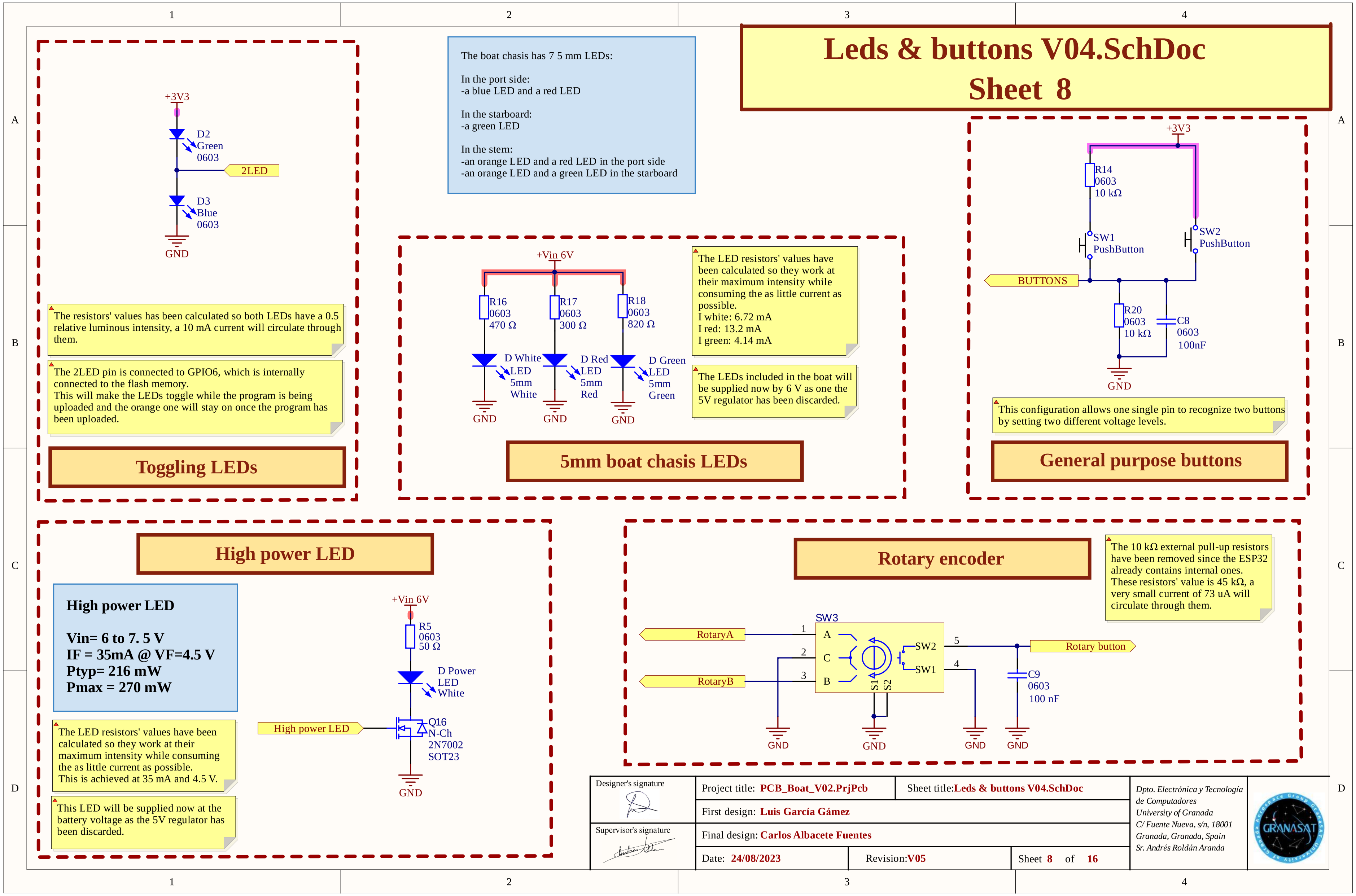


The typical consumption of the servo motor has been estimated experimentally by measuring th current during its different operations.

Assuming that the servo motor is operating at normal conditions, the typical average current can be approximated to 0.075 A and the maximum to 0.22 A.

It can reach 0.6 A current peaks but this value will not be used estimate the battery duration.

Designer's signature	Project title: PCB_Boat_V02.PrjPcb		Sheet title:Servomotor V02.SchDoc	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	First design: Carlos Albacete Fuentes			
Supervisor's signature	Final design: Carlos Albacete Fuentes			
	Date: 24/08/2023	Revision:V05	Sheet 7 of 16	



Leds & buttons V04.SchDoc
Sheet 8

The boat chasis has 7 5 mm LEDs:

In the port side:
-a blue LED and a red LED

In the starboard:
-a green LED

In the stern:
-an orange LED and a red LED in the port side
-an orange LED and a green LED in the starboard

Toggling LEDs

The resistors' values has been calculated so both LEDs have a 0.5 relative luminous intensity, a 10 mA current will circulate through them.

The 2LED pin is connected to GPIO6, which is internally connected to the flash memory.
This will make the LEDs toggle while the program is being uploaded and the orange one will stay on once the program has been uploaded.

5mm boat chasis LEDs

The LED resistors' values have been calculated so they work at their maximum intensity while consuming the as little current as possible.
I white: 6.72 mA
I red: 13.2 mA
I green: 4.14 mA

The LEDs included in the boat will be supplied now by 6 V as one the 5V regulator has been discarded.

General purpose buttons

This configuration allows one single pin to recognize two buttons by setting two different voltage levels.

High power LED

High power LED

Vin= 6 to 7. 5 V
IF = 35mA @ VF=4.5 V
Ptyp= 216 mW
Pmax = 270 mW

The LED resistors' values have been calculated so they work at their maximum intensity while consuming the as little current as possible.
This is achieved at 35 mA and 4.5 V.

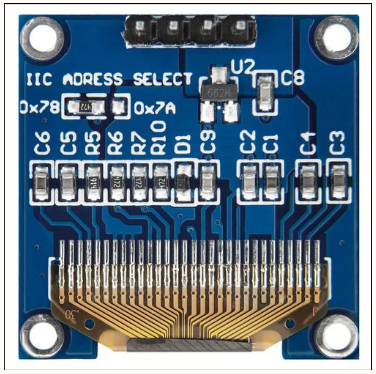
This LED will be supplied now at the battery voltage as the 5V regulator has been discarded.

Rotary encoder

The 10 kΩ external pull-up resistors have been removed since the ESP32 already contains internal ones.
These resistors' value is 45 kΩ, a very small current of 73 uA will circulate through them.

Designer's signature	Project title: PCB_Boat_V02.PrjPcb	Sheet title:Leds & buttons V04.SchDoc	Dpto. Electrónica y Tecnología de Computadores
	First design: Luis García Gámez		University of Granada
Supervisor's signature	Final design: Carlos Albacete Fuentes		C/ Fuente Nueva, s/n, 18001 Granada, Granada, Spain
	Date: 24/08/2023	Revision:V05	Sheet 8 of 16	Sr. Andrés Roldán Aranda

OLED SSD1306 V04.SchDoc
Sheet 9



The I2C address of this device is set by default to 0x3C.

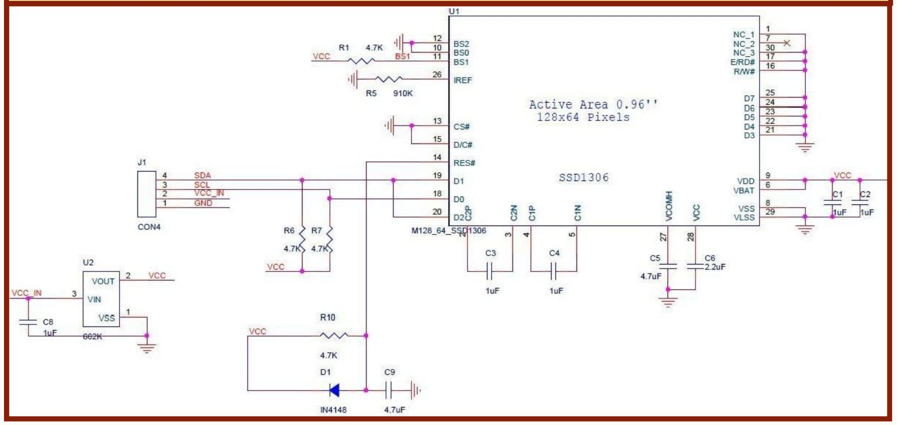
All the I2C devices include 10 kΩ pull-up resistors, so it is necesary to remove some of them from the main board. Otherwise its actual value would be too low since they are in paralel.

The pull-up resistors should only be left in the component that is further from the I2C pins of the ESP32, which is the Adafruit 10-DOF.

Capacitors are not necessary (all of them already have capacitor between VCC and GND integrated in main board)

A level shifter might be necesary to change the voltage of all the SCL and SDA signals to 3.3 V (some might be 5V). However, as we supply all the devices with 3.3 V, it won't be necessary.

OLED SSD1306 schematic

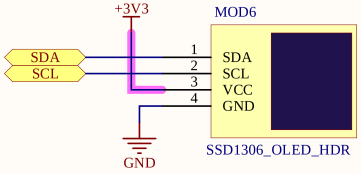


OLED SSD1306

Vin = 1.8 to 6 V
Ityp = 0.150 mA
Imax = 0.150 mA
Ptyp = 0.495 mW
Pmax = 0.495 mW

OLED display with I2C commnication.
It has two colors: yellow and blue.

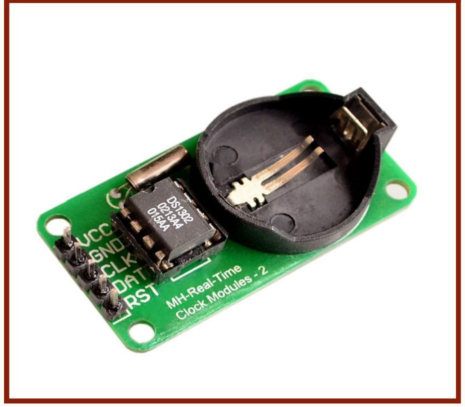
Resolution: 128x64
Size of the screen: 0.96 inches



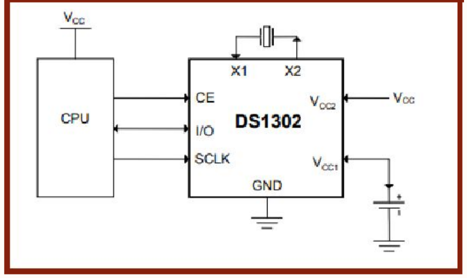
Designer's signature	Project title: **PCB_Boat_V02.PrjPcb**	Sheet title: **OLED SSD1306 V04.SchDoc**	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	First design: **Luis García Gámez**		
Supervisor's signature	Final design: **Carlos Albacete Fuentes**		
	Date: **24/08/2023**	Revision: **V05**	Sheet **9** of **16**

DS1302 RTC V04.SchDoc
Sheet 10

DS1302 Module



DS1302 Module's schematic



DS1302 Module (RTC)

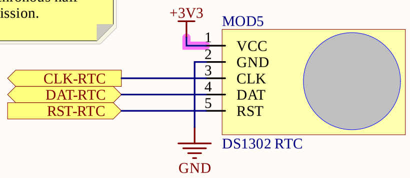
Vin = 3.3 - 5.5 V
Ityp = 0.425 mA
Imax = 1.28 mA
Ptyp = 1.40 mW
Pmax = 4.22 mW

The DS1302 trickle-charge timekeeping chip contains a real-time clock/calendar and 31 bytes of static RAM. It communicates with a microprocessor via a simple serial interface.

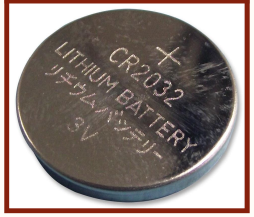
The DS1302 has a built-in power-sense circuit that detects power failures and automatically switches to the backup supply. Timekeeping operation continues while the part operates from the backup supply.

The RTC pins have been moved from GPIO 6,7 and 8 to 33, 12 and 32 since their connection to the flash memory didn't allow a correct 3 wire serial communication with the DS1302 module, instead GPIO8 will be used for 2 LEDs.

It interfaces with the microcontroller using a 3 wire synchronous half duplex data transmission.

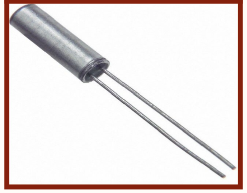


CR2032 Battery



The module contains an external standard 32.768kHz Quartz crystal connected to the X1 and X2 pins of the chip.

32.768kHz crystal



C32032 is a 3 V Lithium coin battery. It supplies the module when the ESP32 is turned off to keep track of time. Its typical life is about ten years.

When operating in low-ower mode, it consumes 200 nA at 3 V.
If the ESP32 is used as a RTC in deep sleep mode it consumes 15 uA.

Designer's signature

Supervisor's signature

Project title: PCB_Boat_V02.PrjPcb
Sheet title:DS1302 RTC V04.SchDoc
First design: Luis García Gámez
Final design: Carlos Albacete Fuentes
Date: 24/08/2023
Revision:V05
Sheet 10 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda

Battery charger V04.SchDoc
Sheet 11

Lead acid battery

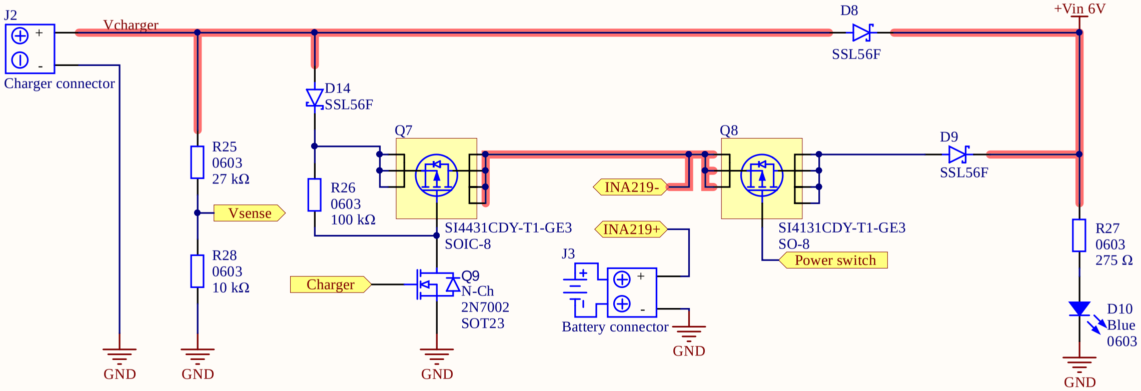
Screw connector

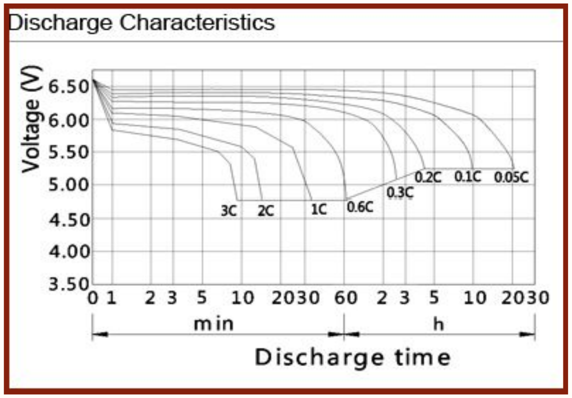
Lead-Acid battery 6V 7Ah

Vbat = 6 V (6.5 to 7 V)
Imax = 70 A (more than 10 A is not recommended)
Pmax = 490 W

The battery and the external power source for the charger will be connected to the board through 10 mm screw connectors.

Diode D8 allows to supply the circuit from the battery charger without the need of a battery. It will only be used during the debugging process and removed after to avoid short circuit between the charger and the battery.





The H bridge is connected directly to the INA219- port so the current that flows through its diodes when it is controlled by PWM can return back to the battery, slightly charging it. The H bridge doesn't need reverse polarity protection.

The 6 label is not exactly true as the battery voltage varies through time and the Vcharger will be higher than 6V to charge the battery (around 7.5 V is a proper value)

This purpose of this circuit is to supply energy from two possible power sources to the other components of the board without creating any shortcut between the two sources.

In addition, we have to be able to charge the battery with the external source. To achieve that, the charger pin will control the current through the pass transistor with a PWM signal. While charging, the pass transistor controlled by the power switched must be open so that the current flows into the battery.

The diodes are low voltage drop Schottky barrier that works as a reverse polarity protection of the circuit. They are also used to prevent current to flow to undesired parts of the circuit while it is charging.

Vsense is a pin that will be read by the ESP32 in order to determine if there is an external sourcer connected or not, as well as its value.

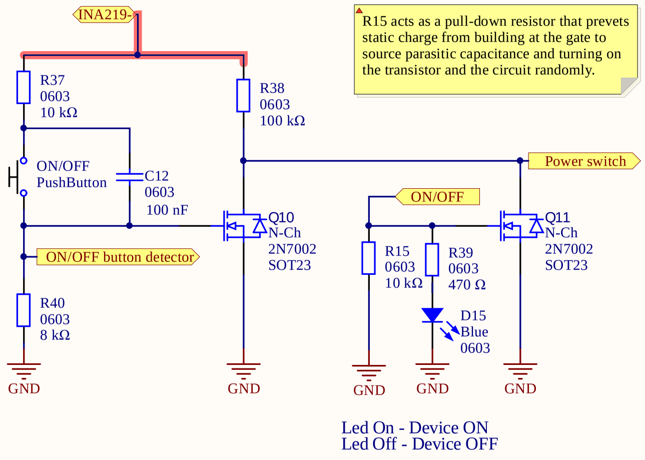
ADC input = 0 - 3.3 V but Vcharger will be higher so we must choose the resistances in order to obtain a Vsense that varies between 0 and 3.3 V .
Vsense = Rdown/(Rup + Rdown)*Vcharger

If Rup = 27 kΩ , Rdown = 10 kΩ -->
Vcharger max = 3.3/(10k / 10k+27k) = 12.2 V

D14 is a reverse polarity protection for the charger connector and D9 for the battery.

Designer's signature	Project title: **PCB_Boat_V02.PrjPcb**	Sheet title: **Battery charger V04.SchDoc**	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	First design: **Luis García Gámez**		
Supervisor's signature	Final design: **Carlos Albacete Fuentes**		
	Date: **24/08/2023**	Revision: **V05**	Sheet **11** of **16**

Power switch V02.SchDoc
Sheet 12



R15 acts as a pull-down resistor that prevets static charge from building at the gate to source parasitic capacitance and turning on the transistor and the circuit randomly.

Since the ESP32 have to keep the pass transistor on to power itself, when it is been programmed it won't have a power supply other than the external source of Vcharger (or keep the button pressed all the time until the program is loaded and the ESP32 is running it).

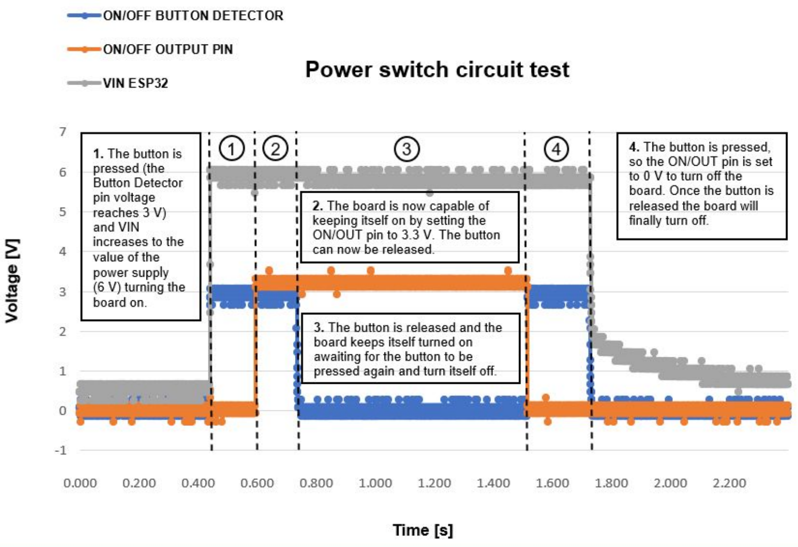
The purpose of this circuit is to control the gate of the Q8 PMOS transistor that will connect or disconnect the battery from the rest of the circuit (except for the H bridge, that is connected directly to the battery).

The PMOS will let the current flow whenever Vsg > Vth
(approx. 1V). If Vg equals the current value of the battery voltage (approx. 6 V) then Q8 will conduct when the Power switch pin's voltage is lower than 5V.

The Power switch pin's voltage can be controlled with a button. If the button is not pressed, the ON/OFF button detector pin voltage will be 0 V so the Q10 NMOS won't conduct, setting the Power switch voltage to 6 V.
If the button is pressed, the ON/OFF button detector pin voltage will be 3 V so the Q10 NMOS will conduct, setting the Power switch voltage to 0 V.
Also, by setting to 3.3 V or 0 V the ON/OFF pin we can control Q11 and control the value of the Power Switch pin.

Taking the above into consideration, we can turn on the ESP32 with the button and keep it ON with the ON/OFF pin from the ESP32. In addition, we can turn off the microprocessor simply by setting ON/OFF at low.

Power switch operation explained



1. The button is pressed (the Button Detector pin voltage reaches 3 V) and VIN increases to the value of the power supply (6 V) turning the board on.

2. The board is now capable of keeping itself on by setting the ON/OUT pin to 3.3 V. The button can now be released.

3. The button is released and the board keeps itself turned on awaiting for the button to be pressed again and turn itself off.

4. The button is pressed, so the ON/OUT pin is set to 0 V to turn off the board. Once the button is released the board will finally turn off.

This test was done by assembling this circuit in a protoboard without the LEDs and using a NodeMCU-32 board.

The board was programmed so it could be turned off by pressing the button again .

Designer's signature

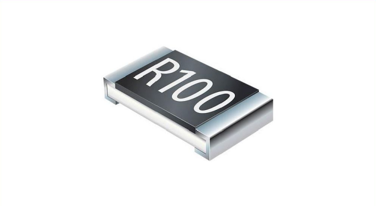
Supervisor's signature

Project title: PCB_Boat_V02.PrjPcb
Sheet title:Power switch V02.SchDoc
First design: Luis García Gámez
Final design: Carlos Albacete Fuentes
Date: 24/08/2023
Revision:V05
Sheet 12 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda

INA219 V04.SchDoc
Sheet 13

HoGT2512-2W-100mR



HoGT2512-2W-100mR
(High power resistor)

R = 0.1 Ω ± 1%
Pmax = 2 W

All the I2C devices include 10 kΩ pull-up resistors, so it is necesary to remove some of them from the main board. Otherwise its actual value would be too low since they are in paralel.

The pull-up resistors should only be left in the component that is further from the I2C pins of the ESP32, which is the Adafruit 10-DOF.

Capacitors are not necessary (all of them already have capacitor between VCC and GND integrated in main board)

A level shifter might be necesary to change the voltage of all the SCL and SDA signals to 3.3 V (some might be 5V). However, as we supply all the devices with 3.3 V, it won't be necessary.

INA219 Module

INA219

INA219
(V and I sensor)

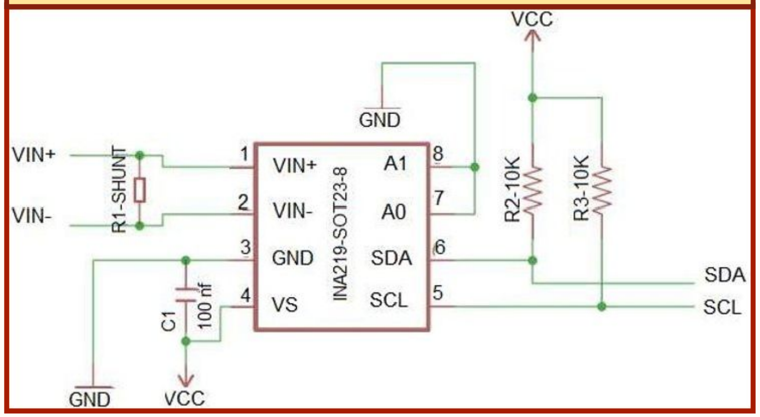
Vin = 3 to 5.5 V
I= 1 mA
P = 3.3 mW

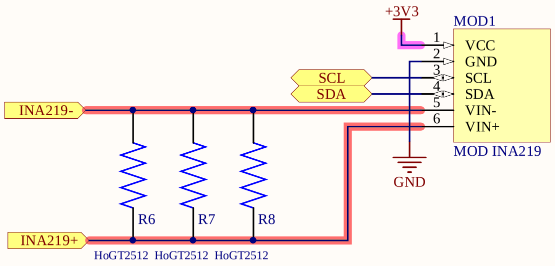
The INA219 is a current shunt and power monitor with an I2C compatible interface. The device monitors both shunt voltage drop and bus supply voltage, with programmable conversion times and filtering.

It includes a 0.1 Ω shunt resistor.

Vshunt max = ±320 mV
Ishunt max = 3.2 A

INA219 Module's schematic



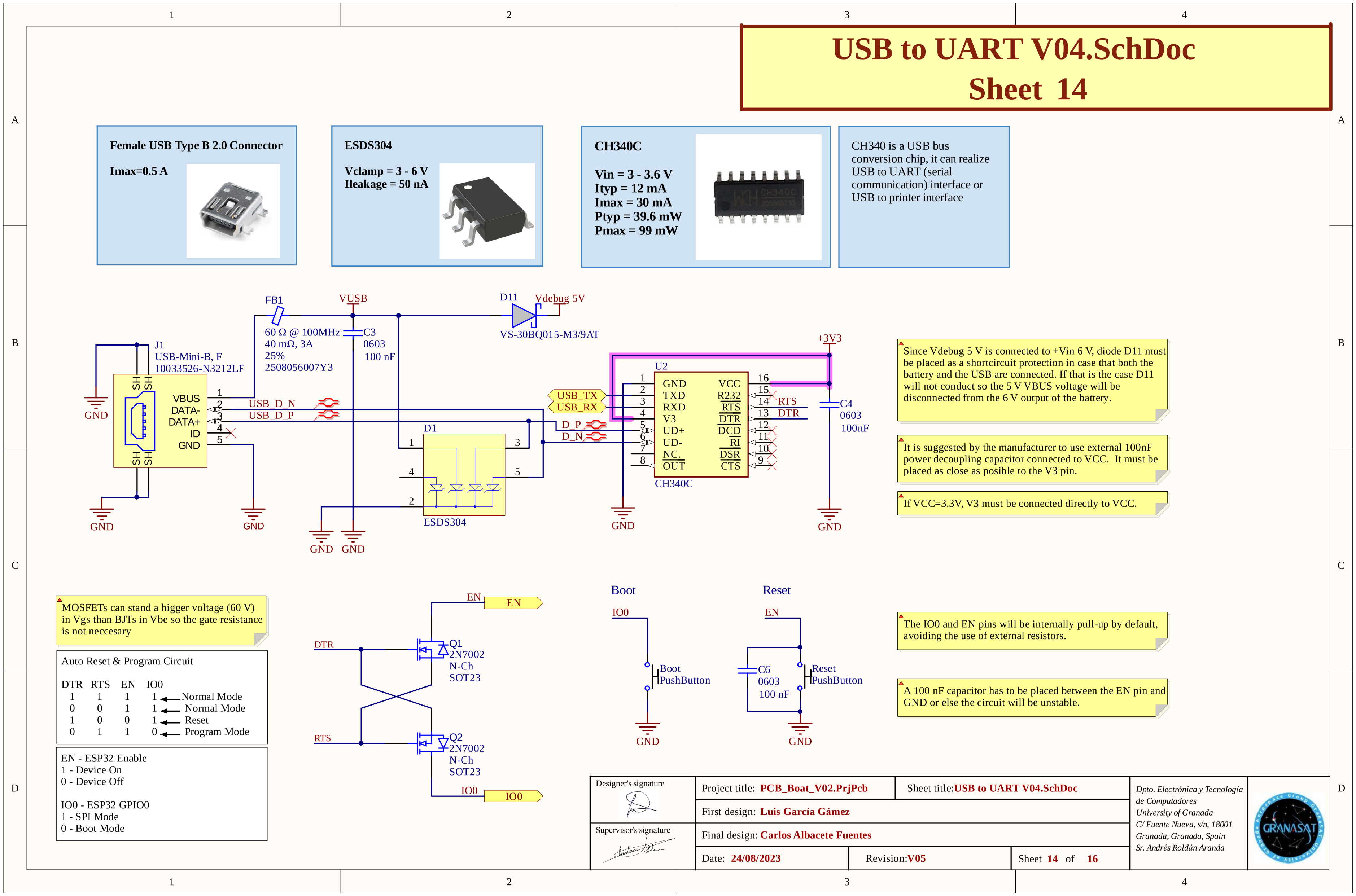


In order to measure currents higher than 3.2 A, the 0.1 Ω shunt resistor must be lower. This resitor can be lowered by adding resistors in parallel.

HoGT2512-2W-100mR is a 0.1 Ω power resistor capable of dissipating 2W. Placing four of these reduces the resistance to 0.025 Ω.

This modification allow us to measure currents up to 12.8 A with the INA219 module.
Ishunt max = 320 mV/0.025 Ω = 12.8 A

Designer's signature	Project title: PCB_Boat_V02.PrjPcb		Sheet title:INA219 V04.SchDoc	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	First design: Luis García Gámez			
Supervisor's signature	Final design: Carlos Albacete Fuentes			
	Date: 24/08/2023	Revision:V05	Sheet 13 of 16	



Since Vdebug 5 V is connected to +Vin 6 V, diode D11 must be placed as a shortcircuit protection in case that both the battery and the USB are connected. If that is the case D11 will not conduct so the 5 V VBUS voltage will be disconnected from the 6 V output of the battery.

It is suggested by the manufacturer to use external 100nF power decoupling capacitor connected to VCC. It must be placed as close as posible to the V3 pin.

If VCC=3.3V, V3 must be connected directly to VCC.

The IO0 and EN pins will be internally pull-up by default, avoiding the use of external resistors.

A 100 nF capacitor has to be placed between the EN pin and GND or else the circuit will be unstable.

MOSFETs can stand a higger voltage (60 V) in Vgs than BJTs in Vbe so the gate resistance is not neccesary

Auto Reset & Program Circuit

DTR	RTS	EN	IO0	
1	1	1	1	← Normal Mode
0	0	1	1	← Normal Mode
1	0	0	1	← Reset
0	1	1	0	← Program Mode

EN - ESP32 Enable
1 - Device On
0 - Device Off

IO0 - ESP32 GPIO0
1 - SPI Mode
0 - Boot Mode

Designer's signature

Supervisor's signature

Project title: PCB_Boat_V02.PrjPcb

Sheet title:USB to UART V04.SchDoc

First design: Luis García Gámez

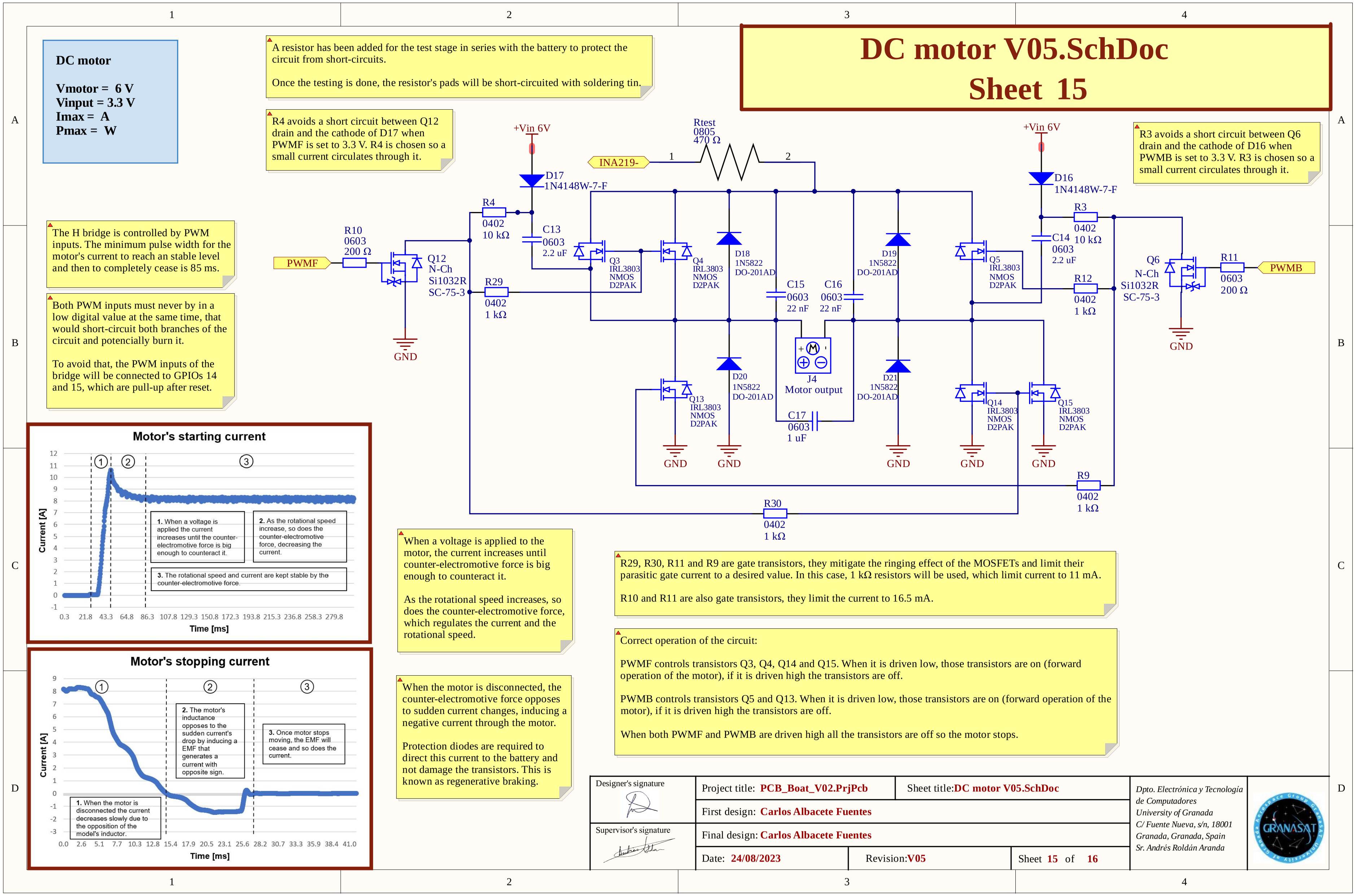
Final design: Carlos Albacete Fuentes

Date: 24/08/2023

Revision:V05

Sheet 14 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda



DC motor V05.SchDoc

Sheet 15

DC motor

Vmotor = 6 V
Vinput = 3.3 V
Imax = A
Pmax = W

A resistor has been added for the test stage in series with the battery to protect the circuit from short-circuits.

Once the testing is done, the resistor's pads will be short-circuited with soldering tin.

R4 avoids a short circuit between Q12 drain and the cathode of D17 when PWMF is set to 3.3 V. R4 is chosen so a small current circulates through it.

R3 avoids a short circuit between Q6 drain and the cathode of D16 when PWMB is set to 3.3 V. R3 is chosen so a small current circulates through it.

The H bridge is controlled by PWM inputs. The minimum pulse width for the motor's current to reach an stable level and then to completely cease is 85 ms.

Both PWM inputs must never by in a low digital value at the same time, that would short-circuit both branches of the circuit and potencially burn it.

To avoid that, the PWM inputs of the bridge will be connected to GPIOs 14 and 15, which are pull-up after reset.

When a voltage is applied to the motor, the current increases until counter-electromotive force is big enough to counteract it.

As the rotational speed increases, so does the counter-electromotive force, which regulates the current and the rotational speed.

When the motor is disconnected, the counter-electromotive force opposes to sudden current changes, inducing a negative current through the motor.

Protection diodes are required to direct this current to the battery and not damage the transistors. This is known as regenerative braking.

R29, R30, R11 and R9 are gate transistors, they mitigate the ringing effect of the MOSFETs and limit their parasitic gate current to a desired value. In this case, 1 kΩ resistors will be used, which limit current to 11 mA.

R10 and R11 are also gate transistors, they limit the current to 16.5 mA.

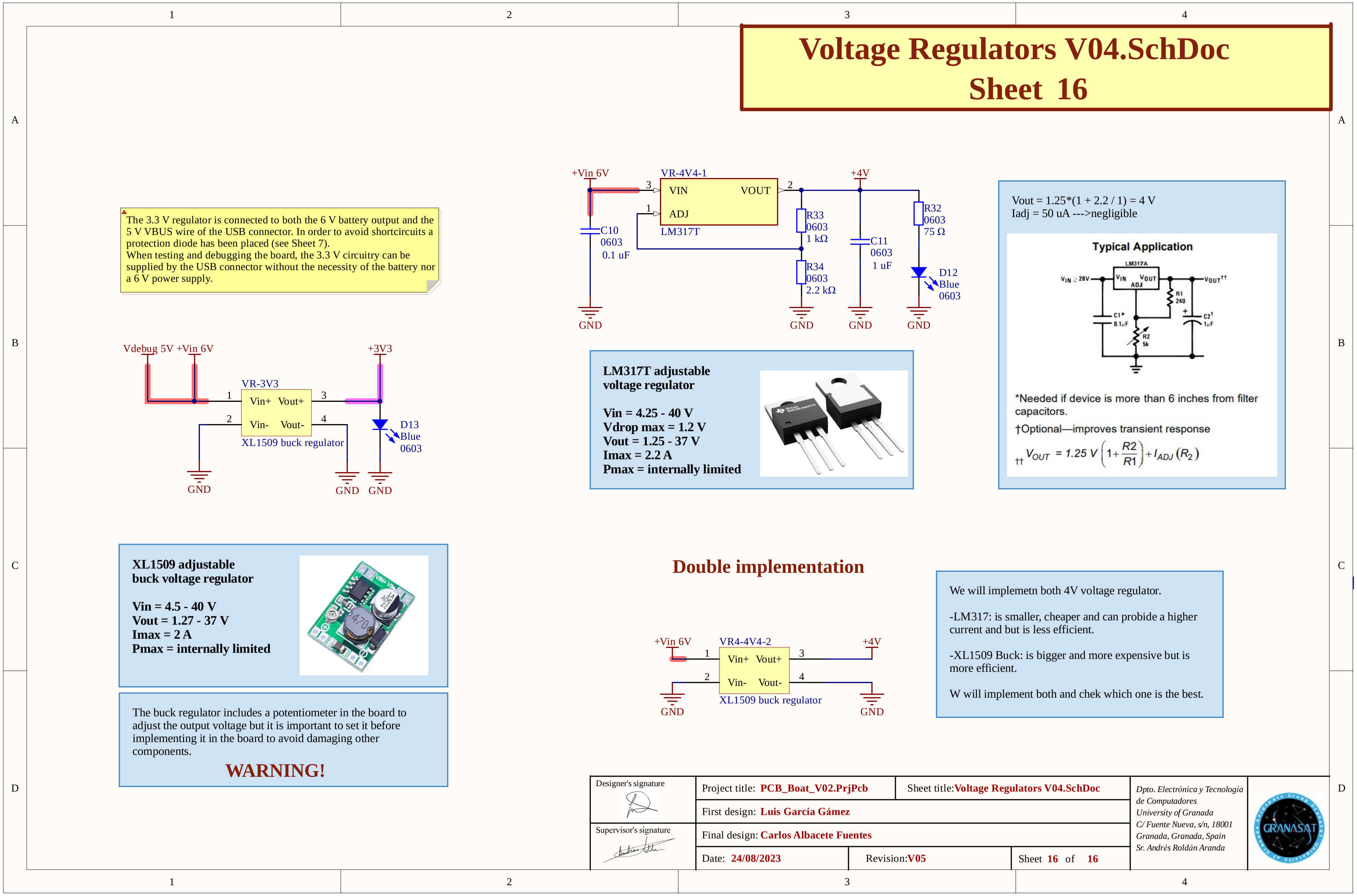
Correct operation of the circuit:

PWMF controls transistors Q3, Q4, Q14 and Q15. When it is driven low, those transistors are on (forward operation of the motor), if it is driven high the transistors are off.

PWMB controls transistors Q5 and Q13. When it is driven low, those transistors are on (forward operation of the motor), if it is driven high the transistors are off.

When both PWMF and PWMB are driven high all the transistors are off so the motor stops.

Designer's signature	Project title: **PCB_Boat_V02.PrjPcb**	Sheet title: **DC motor V05.SchDoc**	*Dpto. Electrónica y Tecnología de Computadores* *University of Granada* *C/ Fuente Nueva, s/n, 18001* *Granada, Granada, Spain* *Sr. Andrés Roldán Aranda*
	First design: **Carlos Albacete Fuentes**		
Supervisor's signature	Final design: **Carlos Albacete Fuentes**		
	Date: **24/08/2023** Revision: **V05**	Sheet **15** of **16**	



Voltage Regulators V04.SchDoc
Sheet 16

The 3.3 V regulator is connected to both the 6 V battery output and the 5 V VBUS wire of the USB connector. In order to avoid shortcircuits a protection diode has been placed (see Sheet 7).
When testing and debugging the board, the 3.3 V circuitry can be supplied by the USB connector without the necessity of the battery nor a 6 V power supply.

Vout = 1.25*(1 + 2.2 / 1) = 4 V
Iadj = 50 uA --->negligible

*Needed if device is more than 6 inches from filter capacitors.

†Optional—improves transient response

$$††\ V_{OUT} = 1.25\,V\left(1+\frac{R2}{R1}\right)+I_{ADJ}\,(R_2)$$

LM317T adjustable voltage regulator

Vin = 4.25 - 40 V
Vdrop max = 1.2 V
Vout = 1.25 - 37 V
Imax = 2.2 A
Pmax = internally limited

XL1509 adjustable buck voltage regulator

Vin = 4.5 - 40 V
Vout = 1.27 - 37 V
Imax = 2 A
Pmax = internally limited

The buck regulator includes a potentiometer in the board to adjust the output voltage but it is important to set it before implementing it in the board to avoid damaging other components.

WARNING!

Double implementation

We will implemetn both 4V voltage regulator.

-LM317: is smaller, cheaper and can probide a higher current and but is less efficient.

-XL1509 Buck: is bigger and more expensive but is more efficient.

W will implement both and chek which one is the best.

Designer's signature

Supervisor's signature

Project title: PCB_Boat_V02.PrjPcb

Sheet title:Voltage Regulators V04.SchDoc

First design: Luis García Gámez

Final design: Carlos Albacete Fuentes

Date: 24/08/2023

Revision:V05

Sheet 16 of 16

Dpto. Electrónica y Tecnología de Computadores
University of Granada
C/ Fuente Nueva, s/n, 18001
Granada, Granada, Spain
Sr. Andrés Roldán Aranda

Appendix E

PCB drawings

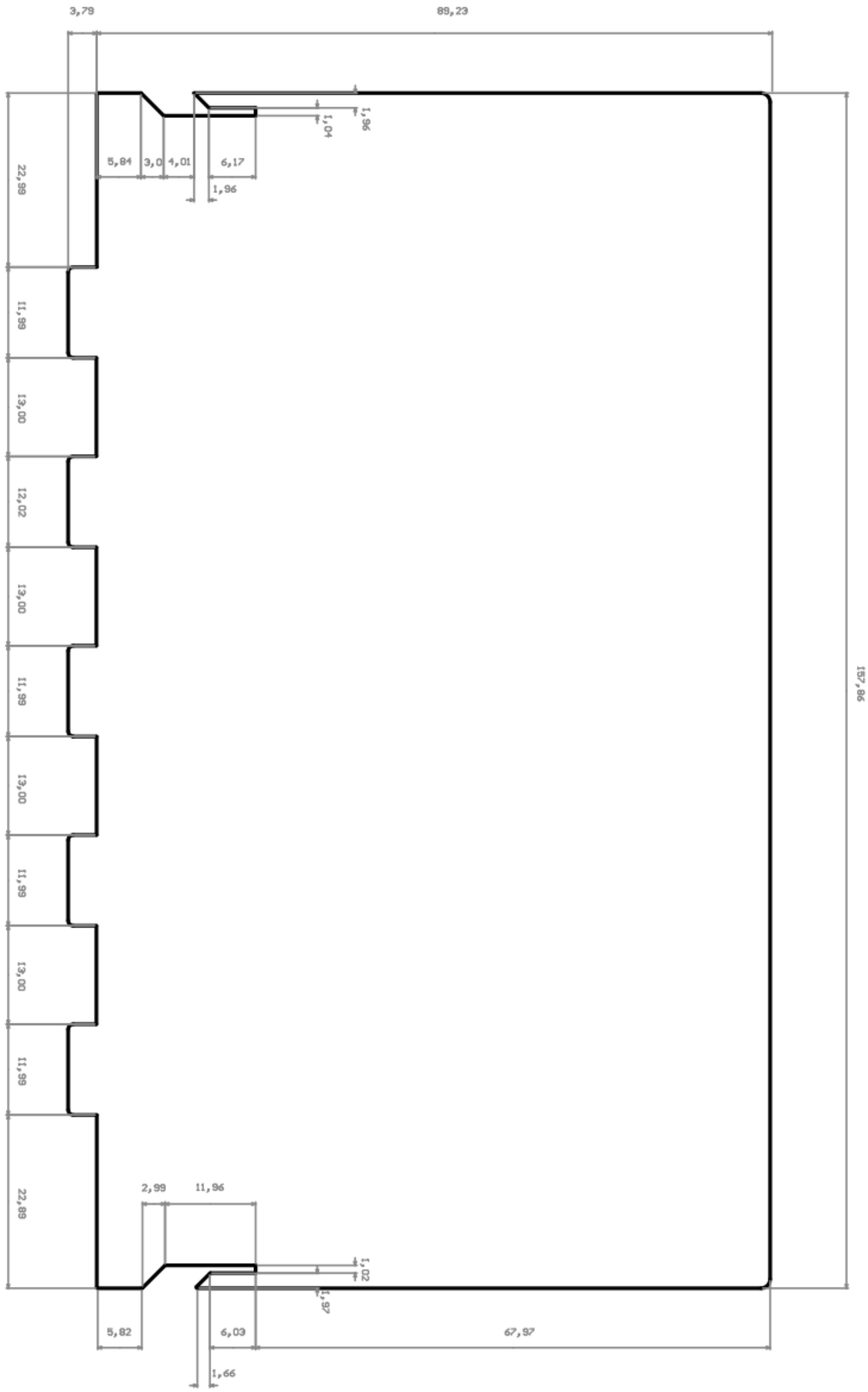


Figure E.1 – *Main board dimensions*

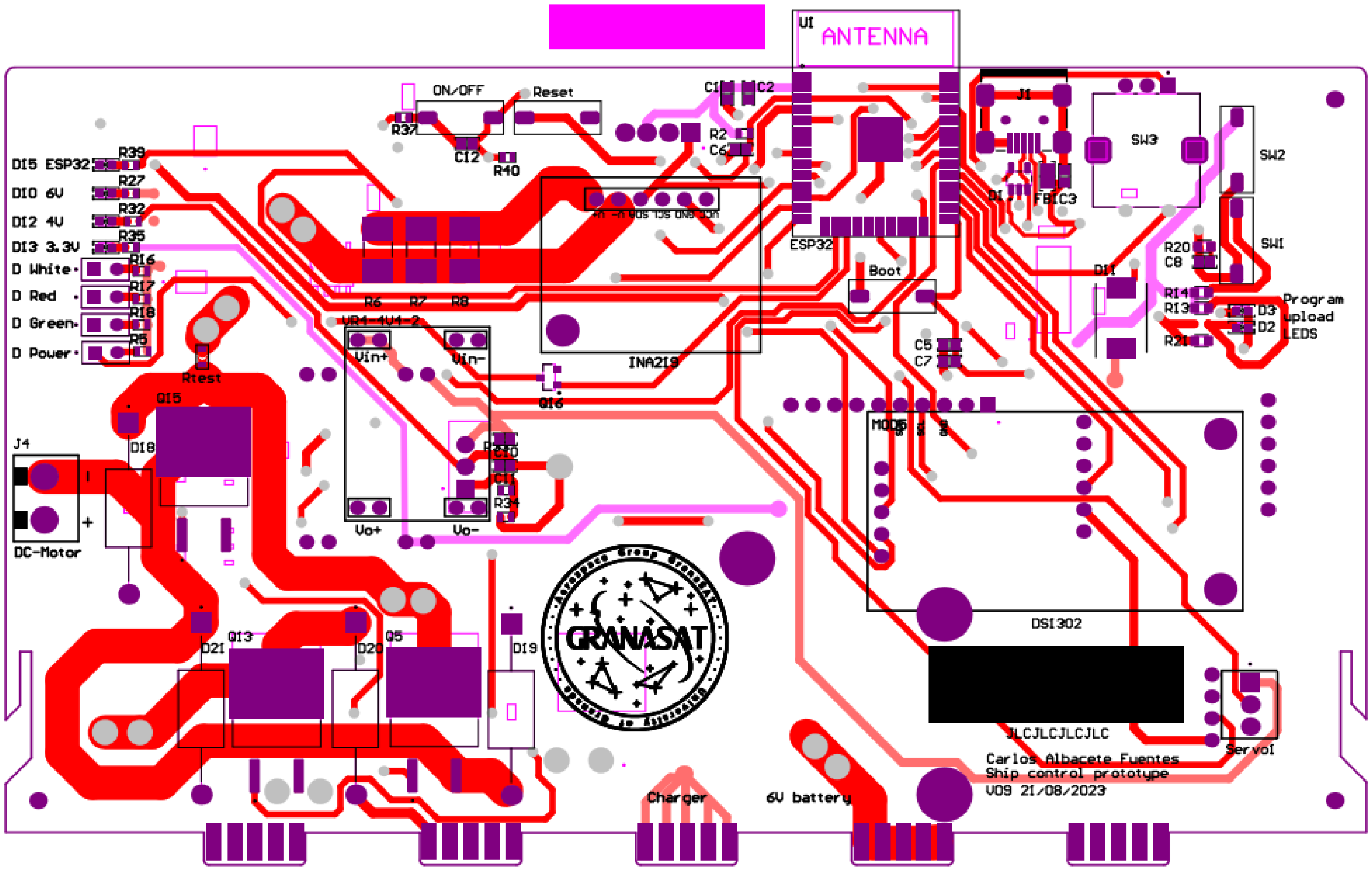


Figure E.2 – *Main board top view without ground plane*

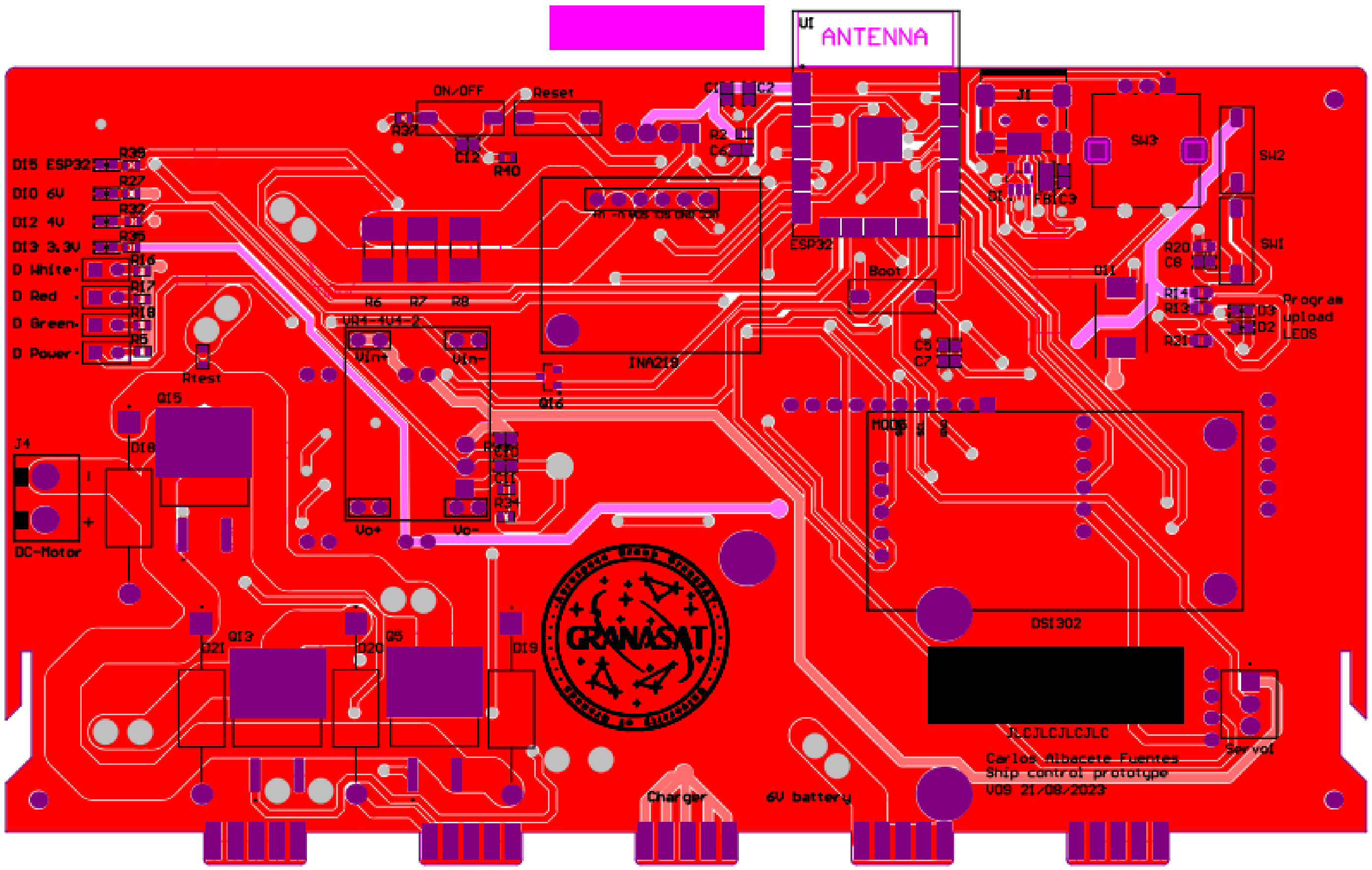


Figure E.3 – *Main board top view with ground plane*

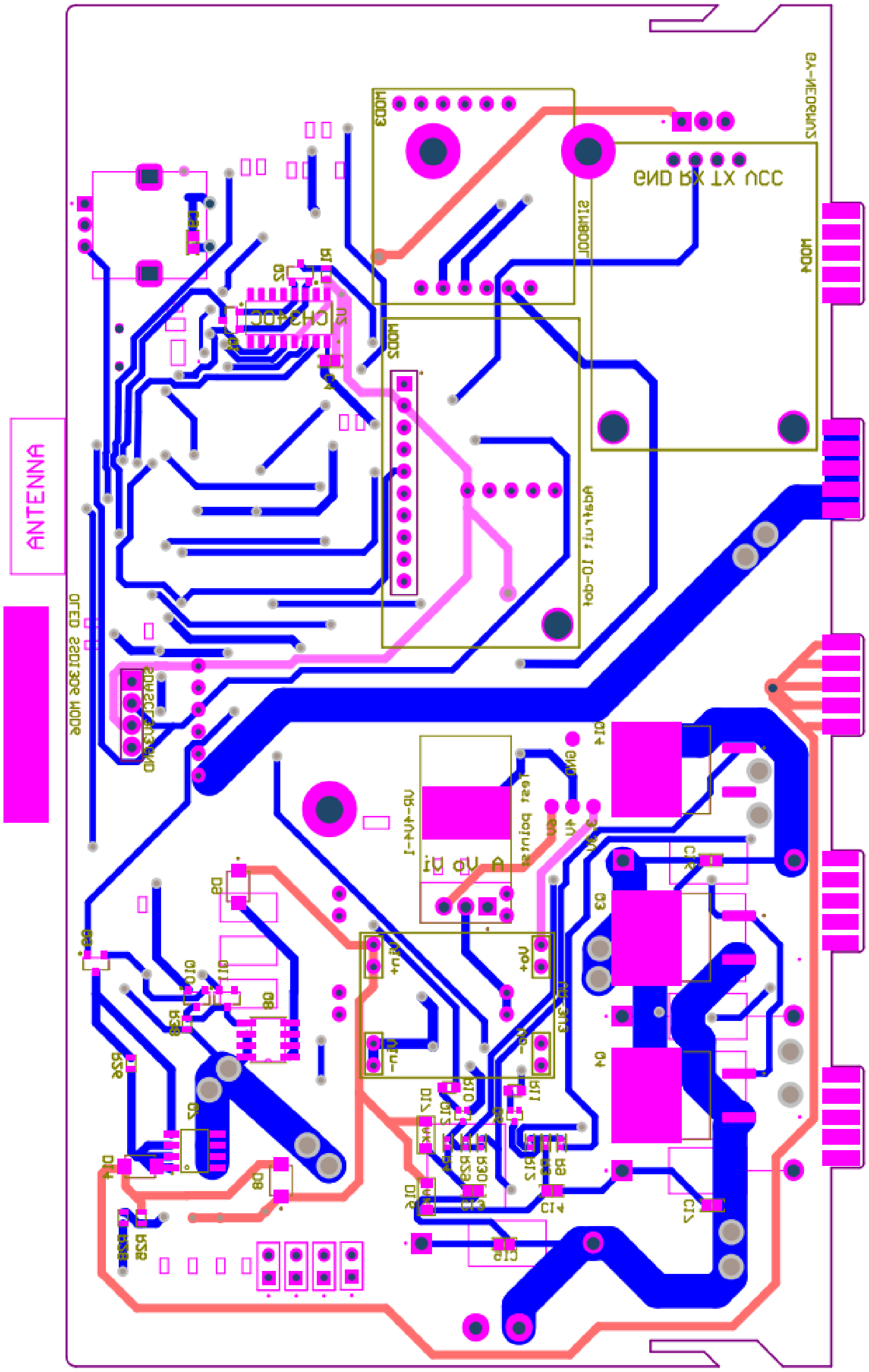


Figure E.4 – *Main board bottom view without ground plane*

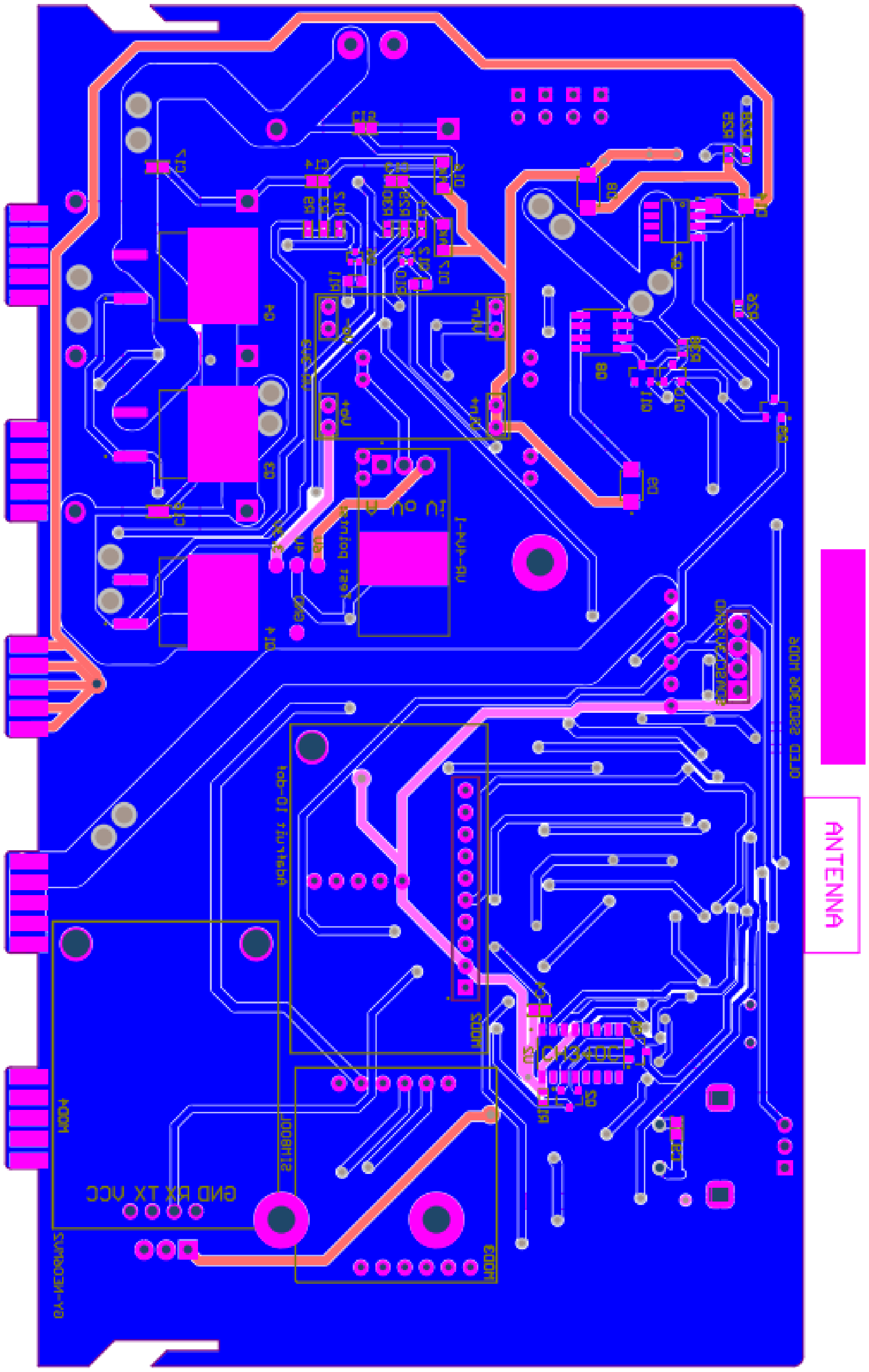


Figure E.5 – *Main board bottom view with ground plane*

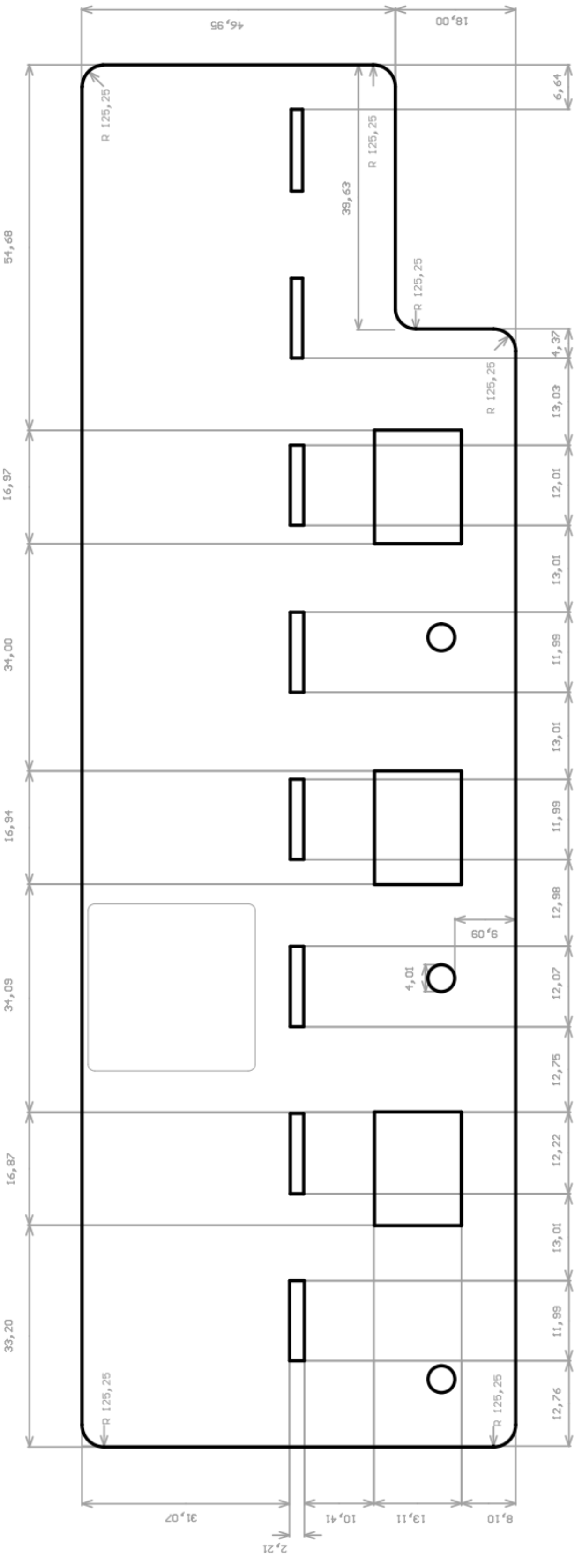


Figure E.6 – *Mount board dimensions*

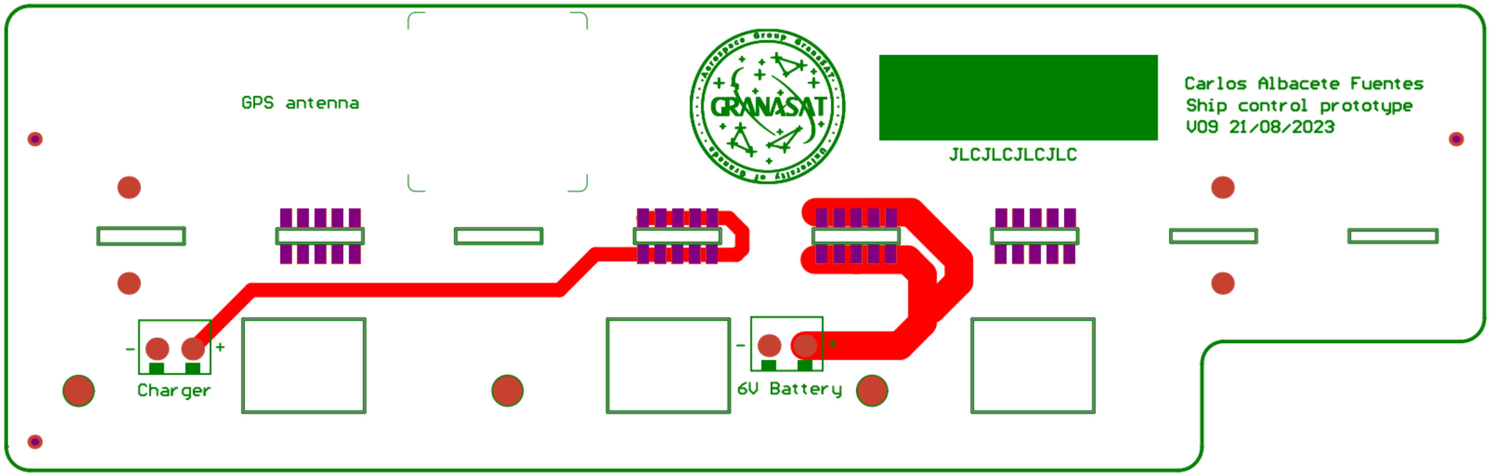


Figure E.7 – *Mount board top view without ground plane*

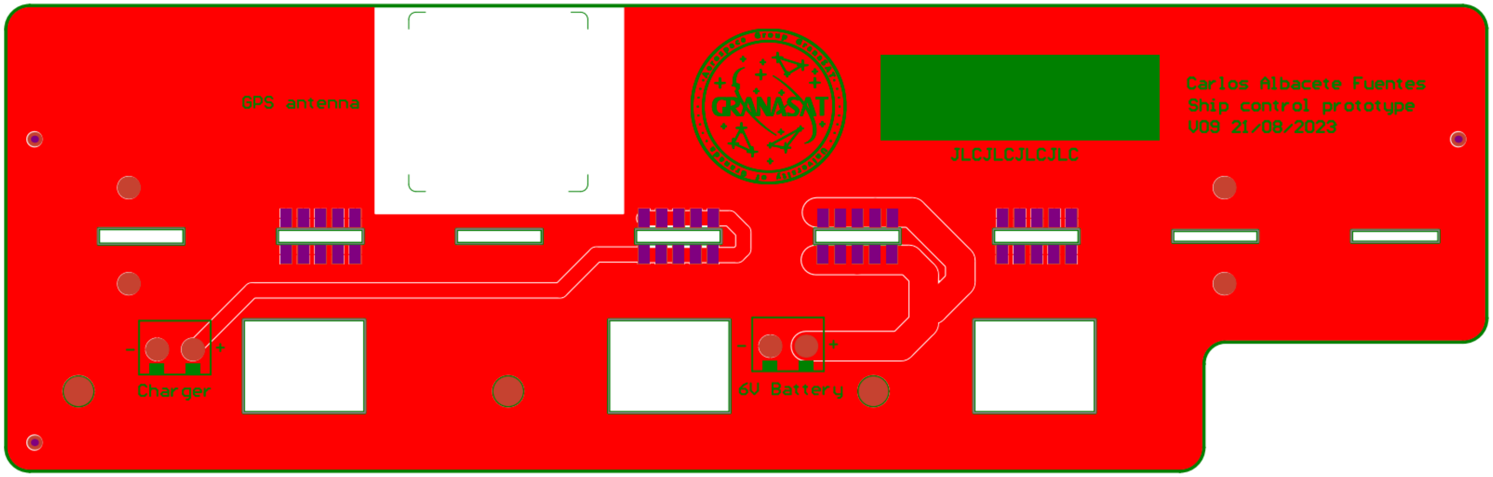


Figure E.8 – *Mount board top view with the ground plane*

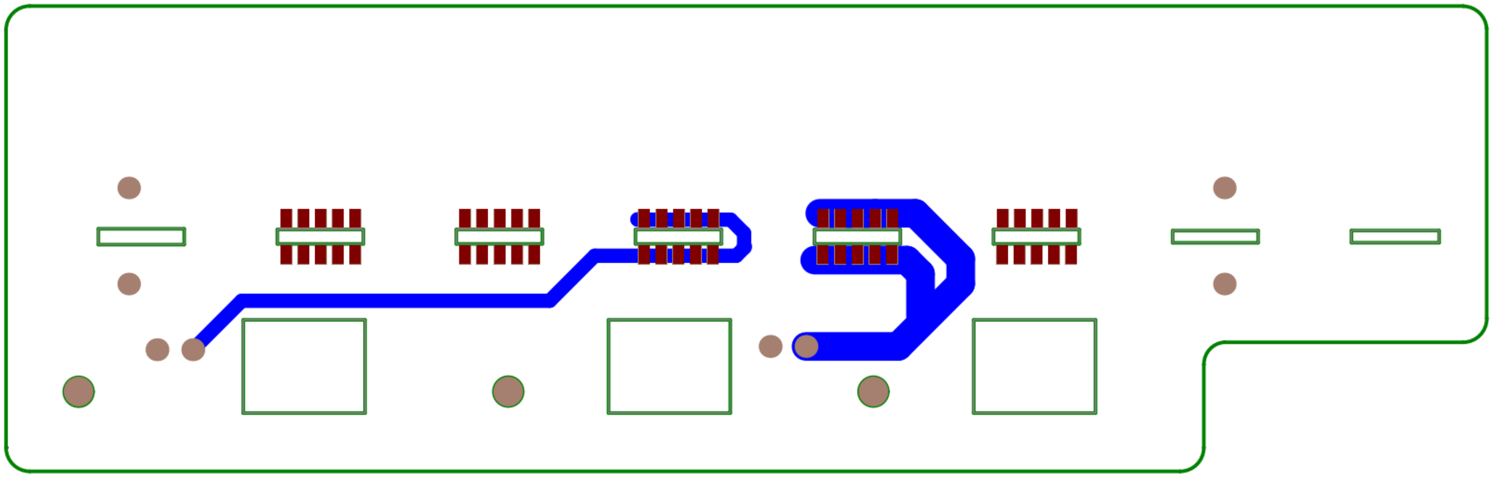


Figure E.9 – *Mount board bottom view without ground plane*

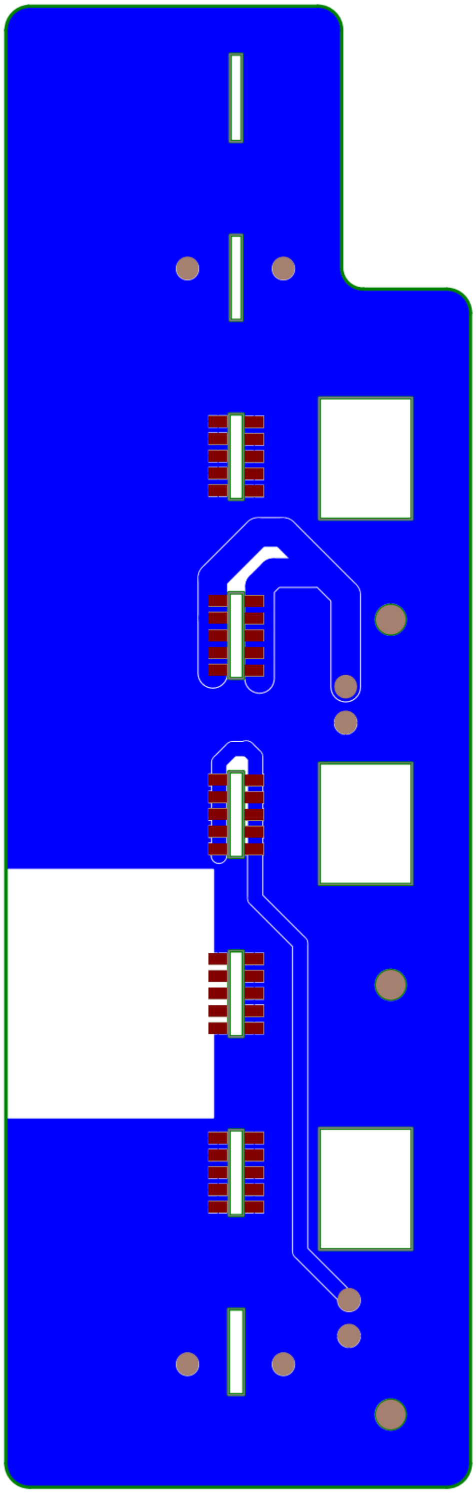


Figure E.10 – *Mount board bottom view with the ground plane*

Appendix F

PCB 3D model

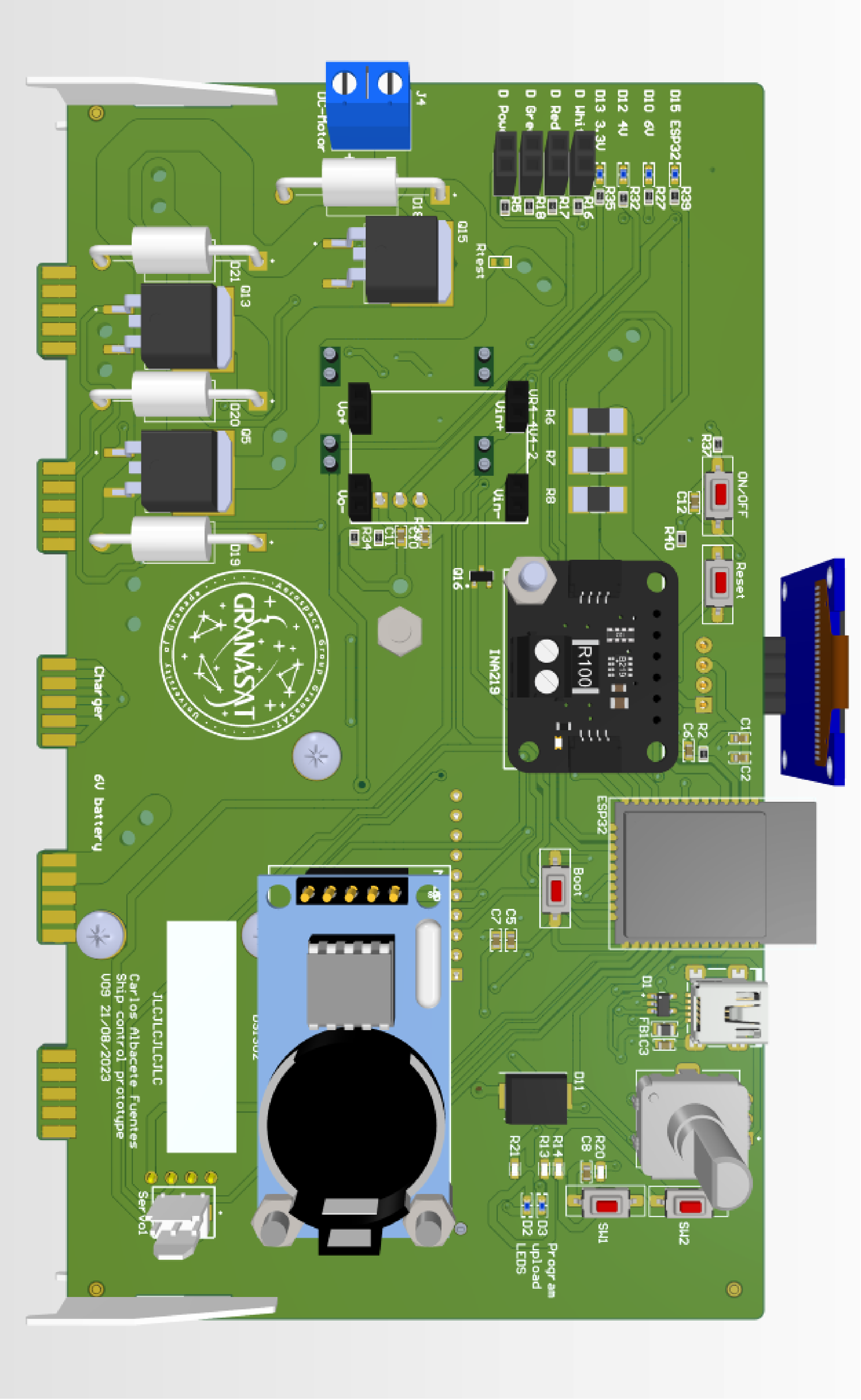


Figure F.1 – *Top view of main board 3D model*

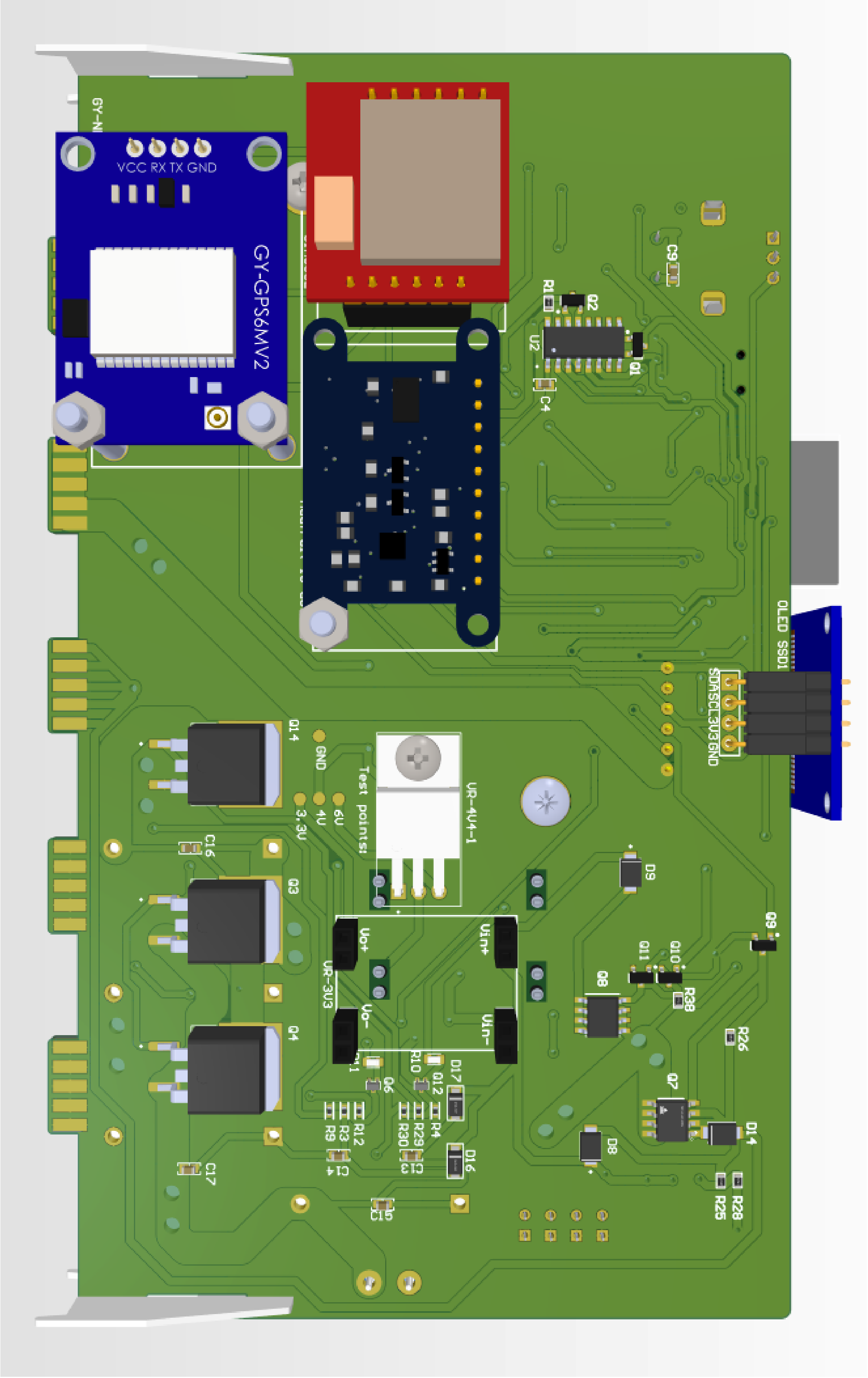


Figure F.2 – *Bottom view of main board 3D model*

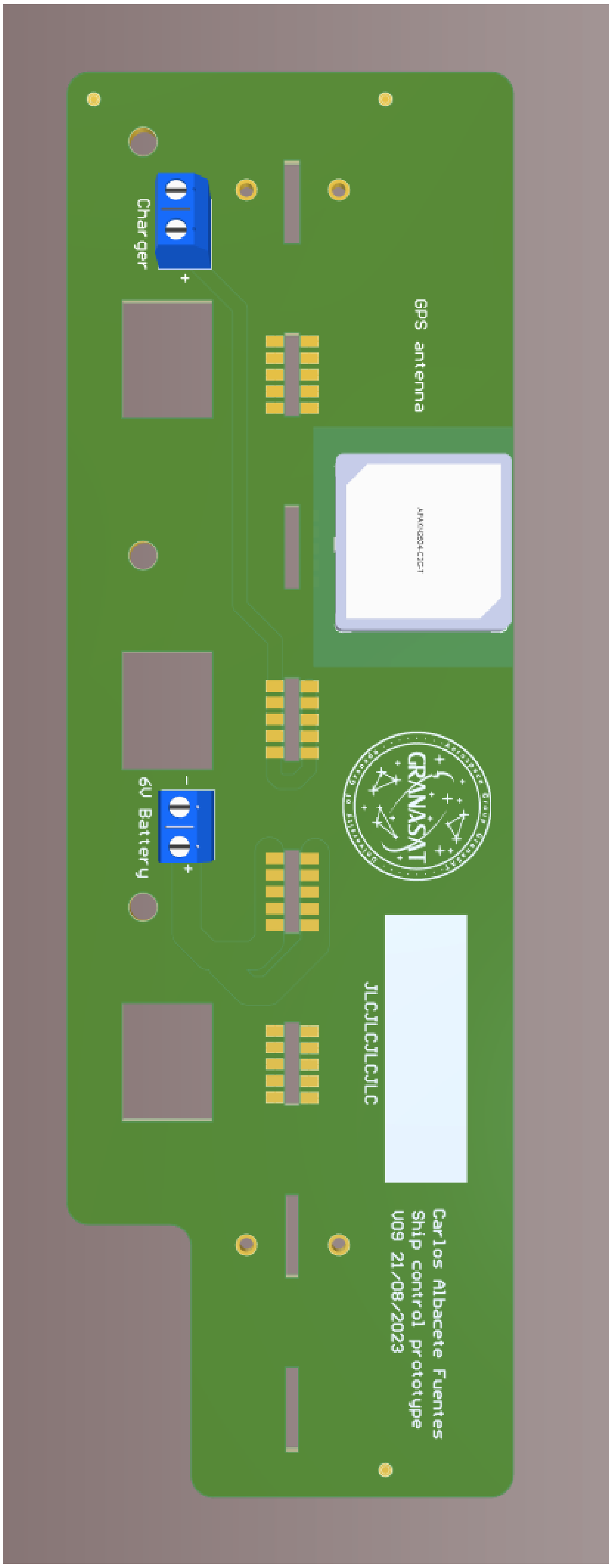


Figure F.3 – *Bottom view of mount board 3D model*

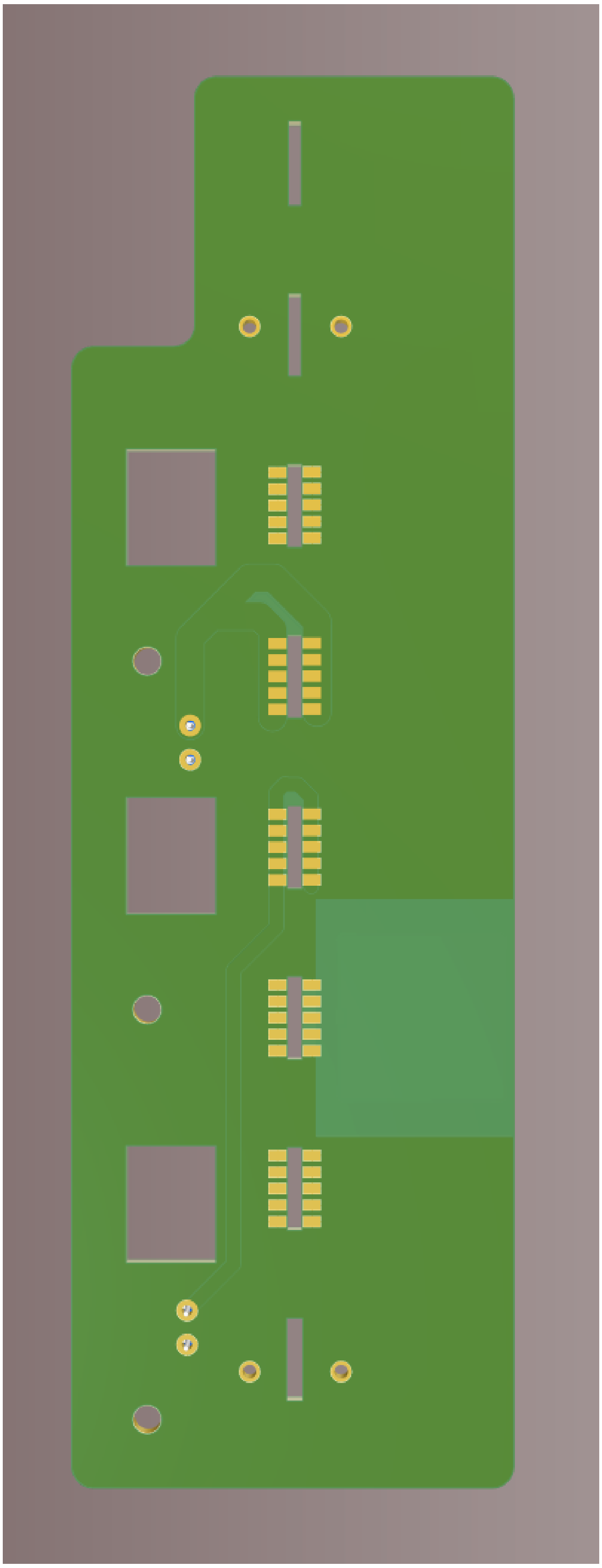


Figure F.4 – *Bottom view of mount board 3D model*

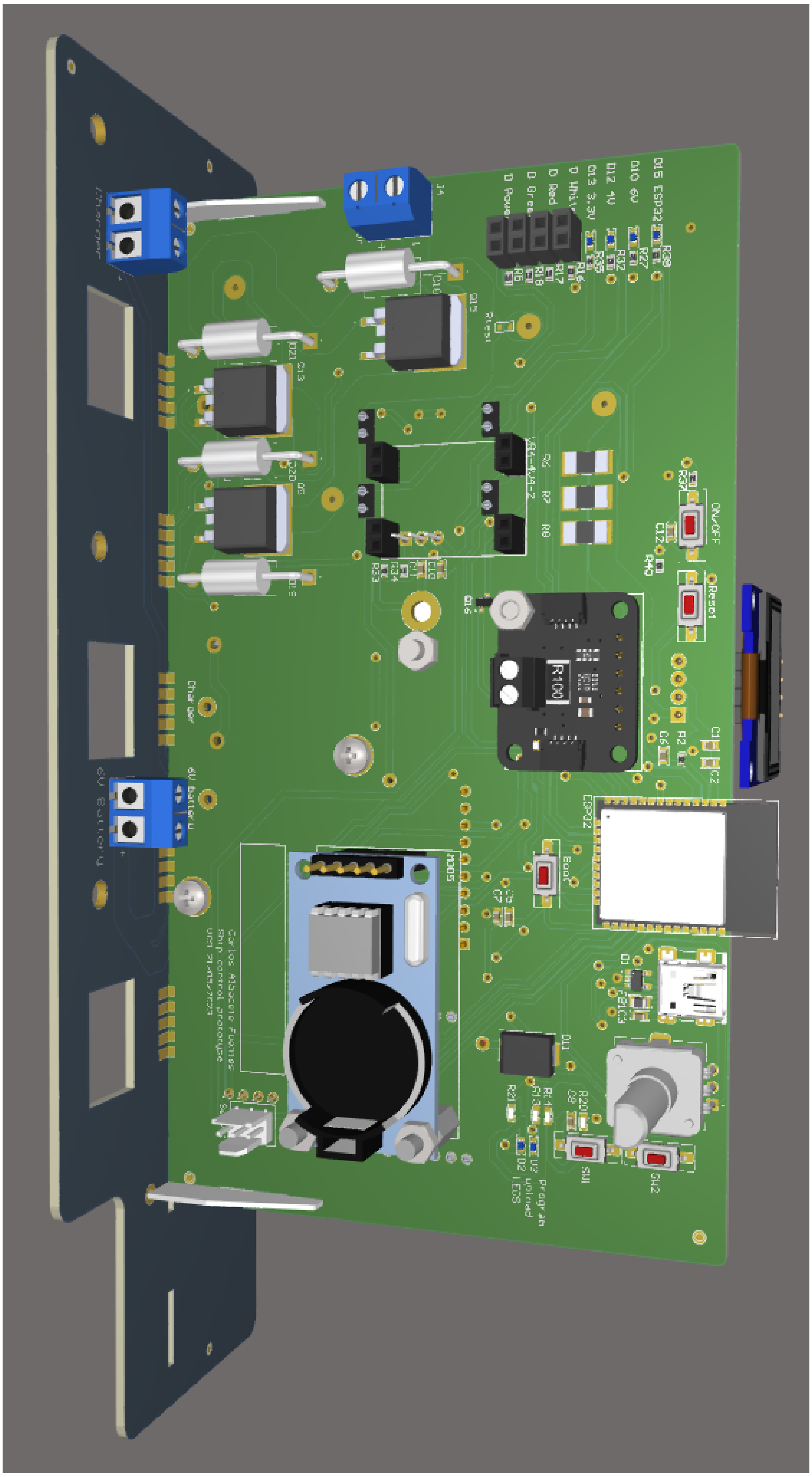


Figure F.5 – *Front view of full PCB 3D model*

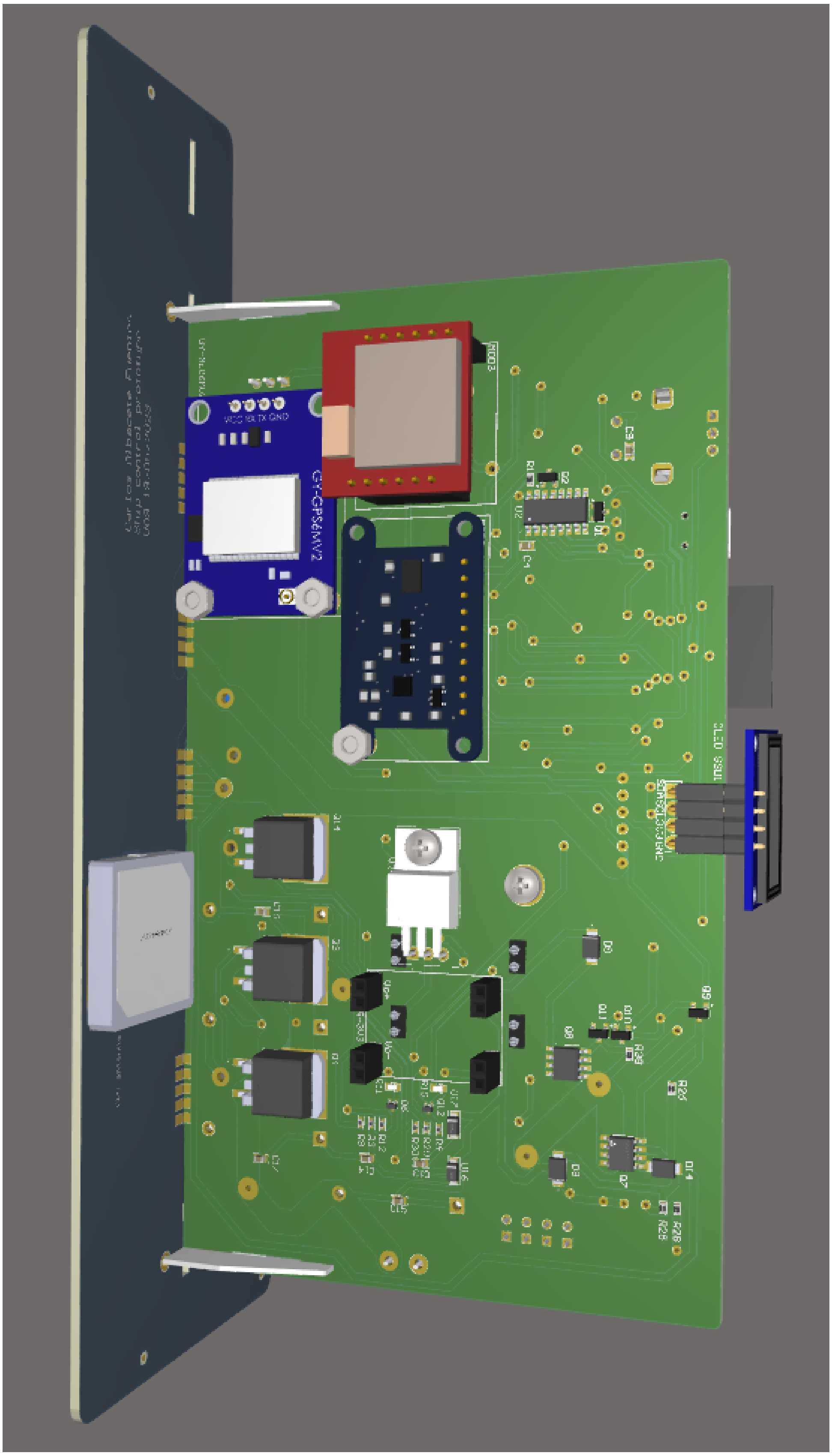


Figure F.6 – *Rear view of full PCB 3D model*