
GliderThrow*Meter*

Release V1.2.0

Nov 04, 2022

Contents

1	Get Started	3
1.1	Introduction	3
1.2	Functional requirement	4
1.3	What do you need	4
1.4	First stage of prototyping : Basic connection diagram	5
1.5	Second stage of prototyping	6
2	Software Design	7
2.1	Logical design	7
2.2	Files organization	10
2.3	Server software architecture	10
2.4	Client software architecture	13
2.5	UX Design	14
3	Hardware Design	19
3.1	Power supply and filtering	21
3.2	Lipo charging	22
3.3	USB to serial converter + ESD protection	22
3.4	MPU6050	23
3.5	ESP-WROOM-32D & Autoreset	24
3.6	Reset circuit	25
3.7	Boot circuit	26
3.8	Adressable LED	27
3.9	PCB routing	28
3.10	Bill Of Material, Eagle Files & Gerber	29
4	System Build	31
4.1	What do you need	31
4.2	How to build assemble one board	32
4.3	How to use the devices	40
5	Indices and tables	41

This is the documentation for the GliderThrow_Meter's project ([GliderThrow_Meter](#)).

GliderThrow is primary design for setting the control surface of a RC glider but you will find that it can be used on most every airplane and for a variety of applications as Measuring a dihedral angle of a wing, Measuring Model Airplane Incidence angle , etc.

GliderThrow meter is made up of two device, each using one [ESP32](#) SOC and one [MPU 6050](#) 6 doffs component.

Each device can measure the deflections in degrees / millimeters with a resolution of 0.1 degrees and can measure the differential when working together with a second unit since GliderThrow is a system that comprises two sensors, one for each wing or surface control pair of your airplane.

Using a dual system simplifies a lot the throw setting of your model by having a direct view of “equivalent” control surfaces at the same time (left and right aileron, or flap).

As the first device embedded a small http server, the data can be viewed through any web browser on a smartphone, PC or MAC, using Windows, Linux, Android or iOS.

UI is built using bootstrap and jquery, and all the files needed are embedded in the .rodata segment of the first device.

The project is made up of two parts, the server (Esp_mad_Server directory) and the client (Esp_mad_Client directory).

Two extras libraries are used in the project : i2cplibdev and MPU6050 from [jrowberg](#).

These libraries are in the extra_components directory of the project.

This project is build using the ESP-IDF 4.0.3 CMake Build System. Please refer to the [espressif documentation](#) for more information to setup an ESP-IDF environnement.





Note: Take care to build the project with an ESP-IDF framework < v4.1 (tcp_adaptor & event_loop APIs have changed since v4.1).

I highly recommend you to use the [Vscode](#) IDE and the [espressif ESP-IDF Vscode Extension](#) to build the project.

All my thanks to the members of the [Electrolab](#) of Nanterre who assisted me during the realization of this project and for the access to the manufacturing and tests tools of this miraculous Hackerspace.

This project is published under the [MIT license](#).

Enjoy !

			
Get Started	Software Design	Hardware Design	System Build

The objective of this project is to design a board based on an ESP-WROOM-32 module from the company Espressif and an MPU6050 component from the company Invensense.

The design process is based on 3 steps:

1. Prototyping of the concept from demoboards
2. Software development based on the prototype
3. Board design (schematic and routing).

The Get started section presents the first part of the process, i.e. the prototyping phase.

1.1 Introduction

The principle of the project is based on the use of two (or more) completely identical boards that communicate using Wifi, on an ad'hoc network (define as ESP_MAD network).

One of the board (named “Server”) is initialized in Access Point (AP) mode. The other boards initialize in Station mode (STA) (named “Client(s)”).

The “Server” board and the “Client” board(s) use similar software, but slightly different because of this specificity of Wifi configuration.

The “Server” board integrates a web server, which allows any web browser to connect to the “Server” board and to navigate on the HTML page (so, connection are possible with all the devices equipped with a a web browser like a PC, a Mac, a Tablet or a Smartphone under Windows, Linux, Android or IOS).

The HTML pages are developed using the Bootstrap framework for the layout and JQuery for the Javascript calls embedded in the pages.

1.2 Functional requirement

The main features requirement from the system are the following:

- Measurement of control surface deflection in either Angle or mm,
- At least two LEDs :
- One to indicate that the circuit is in operation “during calibration of the MPU6050” or in measurement mode after calibration,
- the other to indicate that the battery is charging.
- Recording and display of maximum and minimum deflection (in mm),
- Minimum management of a “Client” board in order to adjust two control surfaces at the same time and ensure “identical” deflection on both control surfaces (aileron, flap, elevator, etc),
- Possibility to change the control surface chord (due to the use of the MPU6050 which allows to recover angles, this feature is mandatory),
- Possibility to easily change the UI of the web interface,
- Battery powered with a battery life time of 3h in normal operation,
- In order to be able to be installed on a small control surface, a casing with a maximum form factor of 41 mm x 41 mm x 31 mm is targeted,
- The design of a storage box for both devices
- The design of the casing must incorporate :
 - A USB interface for reprogramming the ESP-WROOM-32 module and recharging the battery,
 - An on-off button allowing to recharge the battery without powering the rest of the circuit.

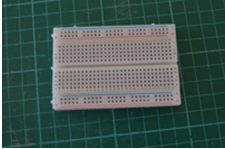
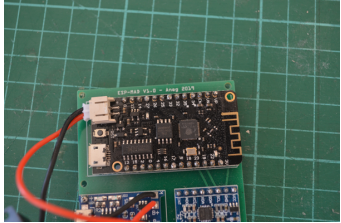

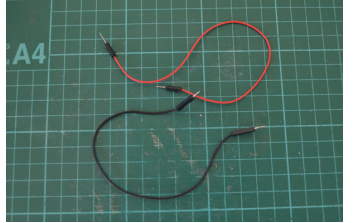
In must-have, it will eventually be possible to manage up to 5 “Client” boards at the same time in order to be able to adjust the 6 control surfaces of an F3F glider.

Note: To date, the software only takes into account one client board.

1.3 What do you need

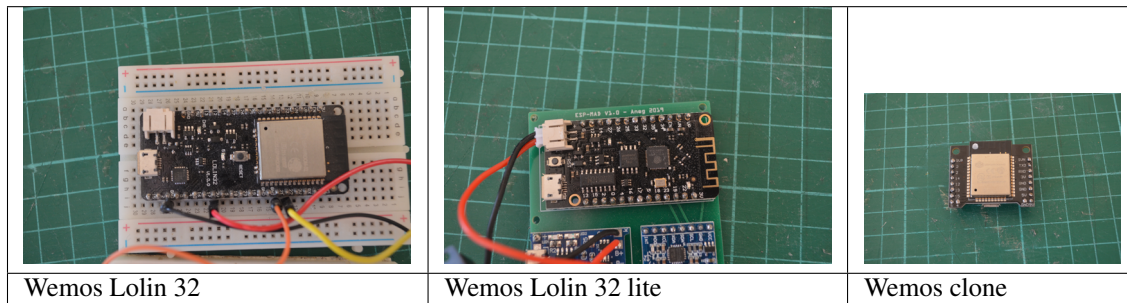
During the prototyping phase, we will use some breadboards, a development board integrating an ESP-WROOM-32 module, a development board integrating an MPU6050, as well as some prototyping wires.

These components can be easily found on the internet (Bandgood, Aliexpress, etc.), for a few euros.

			
Breadboard	Lolin32 demoboard	MPU6050 de-moboard	Prototyping Wires

Several demonstration boards based on ESP-WROOM-32 can be used

I personally used the following boards without any problem.



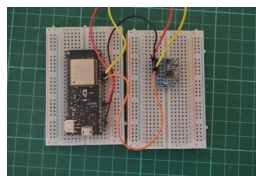
The last board in this table has a Wemos marking, but does not appear to come from the company of the same name. However, it works very well and is a little smaller than the two Lolin 32 Wemos (But this board, unlike the Lolin cards, has no battery connector and therefore no charging circuit). You can easily find this board by doing a search on the net of the type “compact ESP32 board”.

Warning: On some ESP32 boards, in particular the Wemos clone, it is essential to switch the card to “boot” mode during the software loading operation. These boards have generally two buttons, an “EN” button which is actually the Reset button, and a “Boot” button which allows you to switch the board to “Boot” mode when you launch an “idf.py flash” command.

1.4 First stage of prototyping : Basic connection diagram

I used this basic setup to develop the first version of the code, starting with the code for the “Server” board, then the code for the “Client” board. If you want to test it, you’ll have to use two “big” breadboards (one for the “Serveur”, the other for the “Client) or 4 small ones (two for the “Serveur” and two for the “Client”).

The ESP32 demoboard is easily connected to the MPU6050 board using I2C.



So,

Lolin 32	MPU6050
pin 21->	pin SDA
pin 22->	pin SCL
pin 5V->	pin VCC
pin GND->	pin GND

Note: The ESP32 allows you to change the assignments of the I2C pins. To test this feature, I used different assignments on my three demoboards. These mappings are integrated in the `esp_mad.h` file. The figure above shows the connection diagram used with the Lolin 32 demoboard.

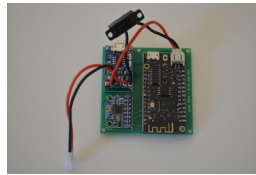
After completing this assembly, you must download the file `esp-mad-server.bin` obtained after compilation into your ESP32 demo card.

The procedure for using the UI is as follows:

1. Leave the breadboard flat,
2. Connect the “Server” card via its USB port,
3. If your board is equipped with an embedded led, this led will blink very quickly to indicate that the system is in calibration phase of the MPU6050 (leave the breadboard flat during this phase). After a few seconds, the LED will switch to a slower flashing mode to indicate that the MPU6050 has completed its calibration,
4. Connect your PC or mobile phone to the Wifi network of SSID “ESP_MAD”,
5. Launch your internet browser,
6. Type in the URL bar of your browser “<http://192.168.1.1>”.
7. The main page of the UI of the GliderThrow_Meter project will appear,
8. If you move the breadboard on which the MPU6050 is connected, the deflection values (up and down) will be displayed.

1.5 Second stage of prototyping

After this first step, I integrated on a small PCB, a Lolin32 Lite demoboard, an MPU6050 demoboard and a 1A 5V Micro USB Module Charger Module Board with Protection (ref. TP4056).



The result is encouraging and works perfectly, but the board has a form factor of 55 mm x 55 mm (i.e. a casing close to 60 mm side) which is off target compared to the initial requirement.

Finally, I tried to integrate the “clone Wemos” demoboard which is more compact by stacking a PCB that integrates the MPU6050 board and the TP4056 board. We arrive at a form factor (excluding the casing) of 38mm x 38mm x 45mm, which is close to the target, but not totally satisfactory in terms of integration and usability.

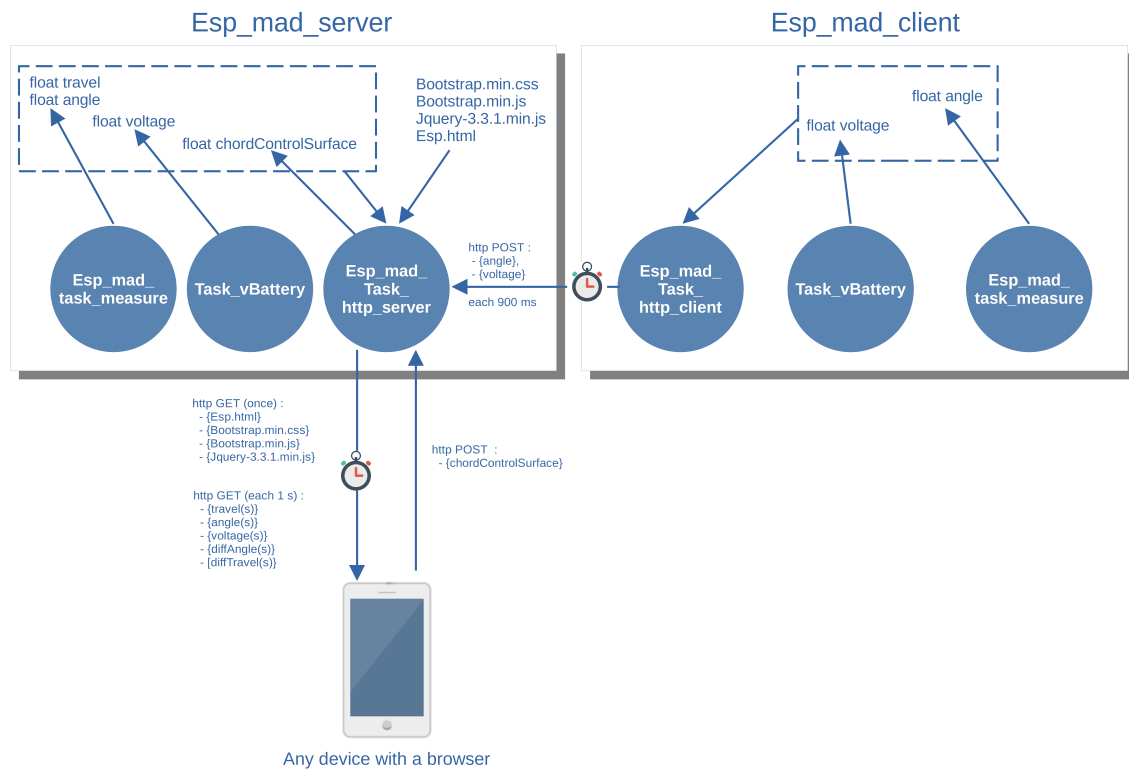


Following these two tests, I came to the conclusion that the integration of off-the-shelf boards would inevitably lead to a form factor that not stick the requirement and a low comfort of implementation.

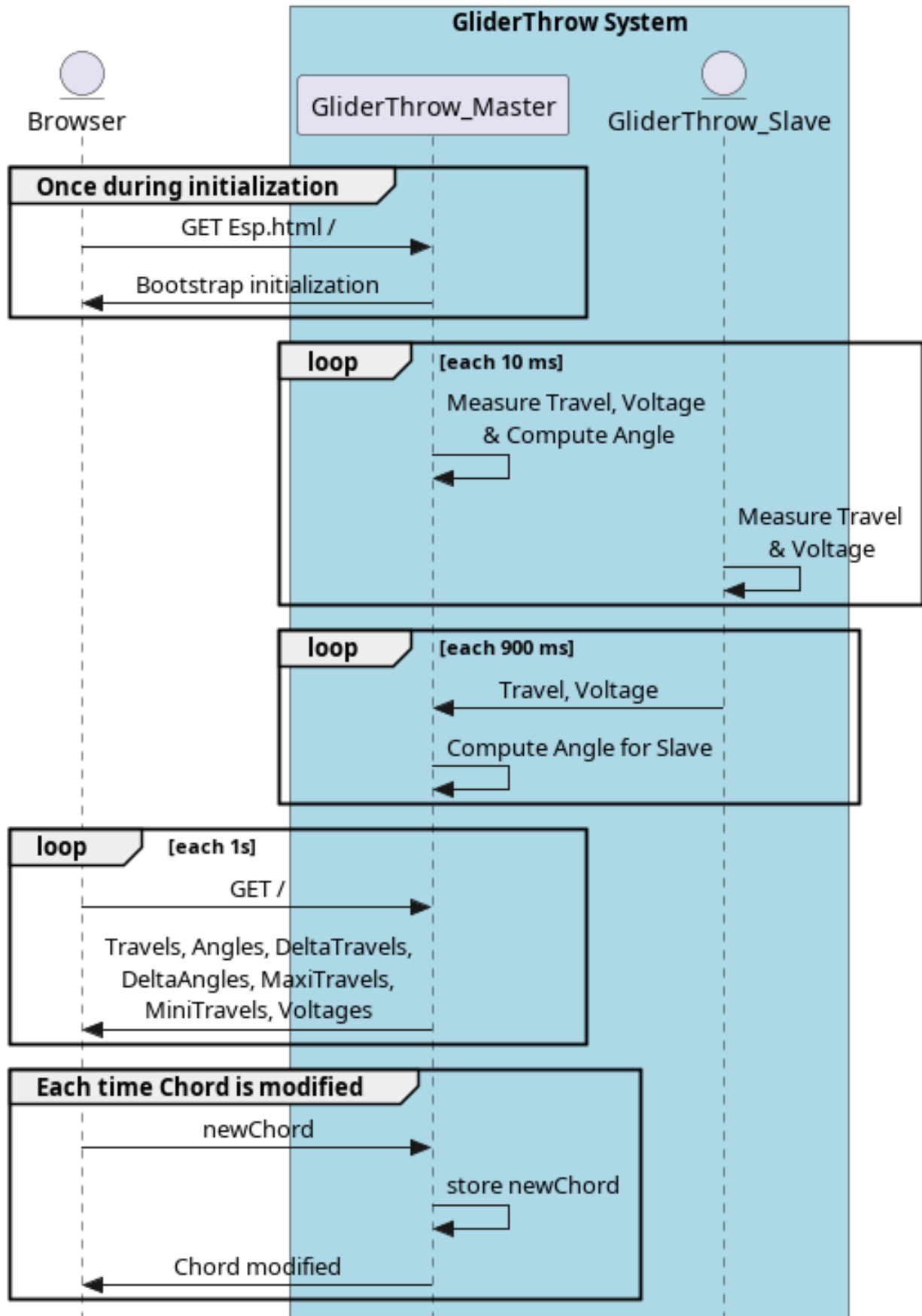
However, this second prototyping stage enabled me to target the components needed for the integration of the boards, And I decided to design a new PCB integrating all of the components. So go first to the next chapter for a description of the software.

2.1 Logical design

The following figure shows the overall logical architecture of the system composed of a server and a client.



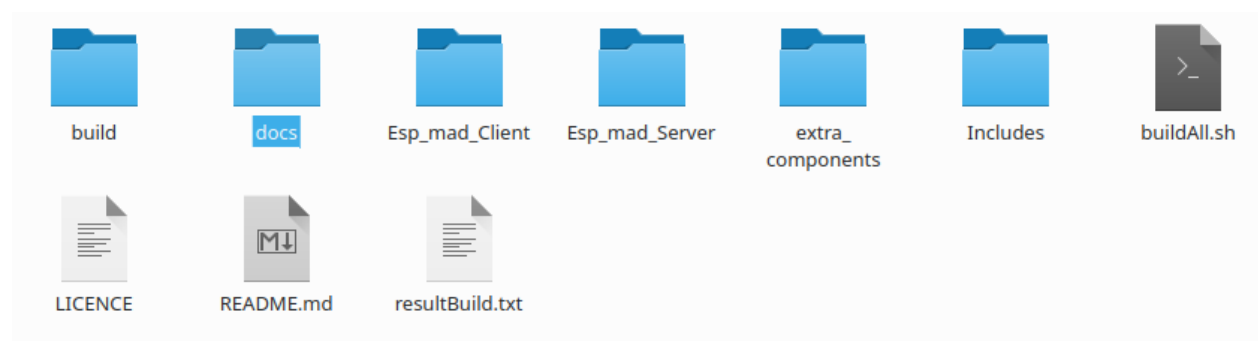
The following sequence diagram shows the global exchanges between the various components of the system.



2.2 Files organization

The project files are organized as follows:

- docs : contains all the documentation related with the project. Documentation of the project is generated using Sphinx (a python documentation generator)
- Esp-esp_mad_client : contains all the code for the Client part,
- Esp_mad_Server : contains all the code for the Server part,
- extra_components : contains the libraries used in the project and two components share between the Server and the Client (esp_mad_task_measure and esp_mad_task_vBattery),
- Includes : contains Esp_mad.h for the globals define used in the code and Esp_mad_Globals_Variables.h for the declaration of the globals variables used,
- buildAll.sh : is a little script used to clean or build all the project. The result is stored in the resultBuild.txt,
- README.md is the presentation of the project used by github and LICENCE is a MIT licence.



The docs directory contains also the bom, the datasheets for the main chips used in the project, the stl files to build the casing and the box, the eagle files and the gerber files.

2.3 Server software architecture

The “Server” software code is made up of three files:

- esp_mad.cpp: this is the launch file which will create three FreeRtos tasks
- the “measure” task
- the “http-server” task
- the “task_vBattery”
- esp_mad_task_http_server.c: it is the file which contains the code of the task “http-server”
- esp_mad_task_measure.cpp: it is the file which contains the code of the task “measure”.
- esp_mad_task_vBattery.c : it is the file with contains the code of the task “task_vBattery”

2.3.1 The http-server task

A web server is a software component that listens for incoming HTTP requests from web browsers. Upon receiving a request, the web server sends a response. This may be the return of an HTML document to be displayed in a browser

or data that forms a response to a service call. An HTTP request can also include data to be sent to the ESP32 for processing. There are many implementations of Web servers that can run in an ESP32 environment.

The Espressif (ESP-IDF) framework provides an API, the HTTP Server component, for implementing a lightweight Web server on ESP32. The two basic API calls are:

- `httpd_start()`: creates an HTTP server instance, allocates resources to it based on the specified configuration, and generates a handle to the server instance. The server will have a listening socket (TCP) for HTTP traffic. The task priority and stack size are configurable when creating the server instance by passing the `httpd_config_t` structure to `httpd_start()`. TCP traffic is parsed as HTTP requests and, depending on the requested URI, registered handlers will be called to return HTTP response packets.
- `httpd_stop()`: stops the server with the provided handle and releases the associated resources. This is a blocking function that first signals a stop to the server task, and then waits for the task to finish. Upon termination, the task closes all open connections, deletes registered URI handlers, and resets all session context data to empty.

To process HTTP requests sent to the server, we will need to register URI handlers with :

- `httpd_register_uri_handler()`: registers a URI handler by passing an `httpd_uri_t` structure object that has members including the IR name, method type (e.g. `HTTPD_GET` / `HTTPD_POST` / `HTTPD_PUT` etc ...), a function pointer of type `esp_err_t * handler(httpd_req_t * req)` and `user_ctx` pointer to the context data.

The `http-server` task starts by launching a DHCP server, then initializes the board in AP mode by associating the SSID defined in the `esp_map.h` file.

The address 198.168.1.1 is assigned to the Wifi AP as defined when the DHCP server is launched.

During the initialization of the Wifi in AP mode, an `event_group` is created to receive the various events that can be received by

- When the `SYSTEM_EVENT_AP_START` event is received, the web server is launched using the `httpd` function library. When the server is launched, the various URLs on which the server is likely to react are recorded and for each URL a callback function is associated.
- When the event `SYSTEM_EVENT_AP_STACONNECTED` is received, the corresponding bit is recorded in the `event_group`.
- On receipt of the `SYSTEM_EVENT_AP_STADISCONNECTED` event, the corresponding bit is recorded in the `event_group`.
- On receipt of the `SYSTEM_EVENT_AP_STOP` event, the web server is stopped and the associated resources are released.

Note: The URL “/”, which corresponds to the reception of an HTTP GET at the address 192.168.1.1, i.e. the main page of the site (`esp.html`), triggers the loading of the elements embedded in the page, which are `bootstrap.min.css`, `bootstrap.min.js` and `jquery-3.1.1.min.js`, by as many HTTP GET requests as required by the client browser.

All these elements, as well as the `esp.html` page, are embedded in the `.rodata` segment of the ESP32 memory (using the `MBED_FILES` directive in the project's `CMakeList.txt` file).

Each element contains in the `.rodata` segment is then referenced in the code using the following two directives :

```
extern const uint8_t esp_html_start[] asm("_binary_esp_html_start");
extern const uint8_t esp_html_end[] asm("_binary_esp_html_end");
```

Note: It is also possible to perform the same operations using a SPI Flash File System (SPIFFS), but I have not tested this solution. For a Web server using more than one HTML page, this method is probably more interesting than the method consisting in embedding the pages in the `.rodata` segment.

Data calculated by the “Measure” task (angle and travel) are retrieved by the http_server task from memory, these two variables being defined as global variables. These two values are updated by the “Measure” task every 10 ms.

The deflection angle information measured by the “Client” board is received at a frequency of 900 ms by an HTTP POST request. On receipt of the request, the deflection value in mm is calculated according to the control surface chord.

When the chord is changed from a web browser, an HTTP POST request is received and the chordControlSurface global variable is changed.

When the user wants to reset the Maximum(s) up and down on the travel tab, an HTTP POST request is received and the values are set to 0.

2.3.2 The “measure” task

The task “measure” performs the following functions :

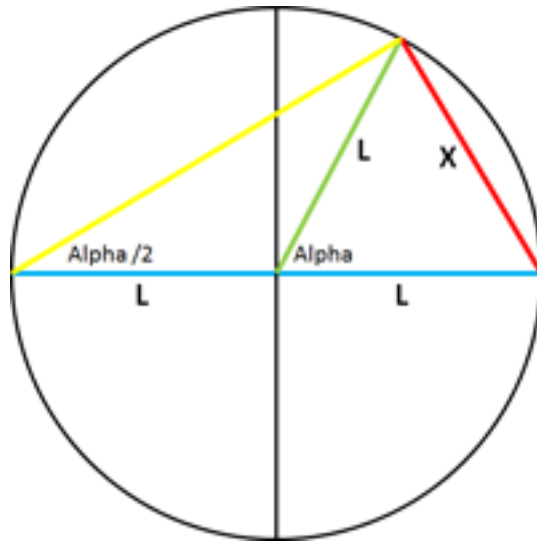
- initialization of the I2C bus,
- calibration of the MPU6050 component,
- Then periodically:
- Reading of the accelerometer and gyroscope values on the axes (x, y, z),
- Calculation of the angle in degrees based on the previous values.

Note: the task “measure” is identical for the “Server” board and the “Client” board. The only difference is that in the case of the “Server” board, the deflection value in mm is calculated periodically by the “measure” task, whereas for the “Client” board, the value of the angle is transmitted to the “Server” board using an HTTP POST request and it is the “Server” board that performs the calculation of the deflection in mm.

Complementary filter is used to combine accelero and gyro data. see [complementary filter](#) for more information.

Basically complementary filter avoid used of kallman filter, quiet difficult to implement in small platform as an ESP32. Gyro are used for fast motion as accelero are used for slow motion.

Note: The deflection value in mm is calculated as a function of the angle alpha by the following formula : $X = 2 * \sin(\alpha/2) * L$.



2.3.3 The “task_vBattery” task

The task “vBattery” compute periodically (each 30s per default) the measurement of the voltage of the battery.

The battery voltage is connected to the IO35 pin of the ESP-WROOM-32. This pin is the channel 7 of the ADC1.

A bridge resistor divider with two resistors of 100 KOhm is used to decreased the voltage from 4.2 V to 2.1 V. So the attenuation of the ADC is set to 11 dB.

2.4 Client software architecture

The “Client” software code is made up of three files:

- esp_mad_client.cpp: this is the launch file which will create three FreeRtos tasks
- the “measure” task
- the “http-client” task
- the “task_vBattery”
- esp_mad_task_http_client.c: it is the file which contains the code of the task “http-server”
- esp_mad_task_measure.cpp: it is the file which contains the code of the task “measure”.
- esp_mad_task_vBattery.c : it is the file which contains the code of the task “task_vBattery”.

2.4.1 The measure task

The measure task is totally the same code than the measure task of the “Server”. No more words to add to this section :-)

2.4.2 The esp_map_task_http_client

The “http-client” task start to initialize the board on wifi station.

Then, the task checks periodically if the board is connected to the “Server” Board, and if the MPU6050 calibration is finish testing the global Binit variable.

If these conditions are true, an HTTP POST with the angle measure by the board is send to the “Server” board.

2.4.3 The task_vBattery

The task_vBattery is totally the same code than the task_vBattery of the “server”. No more words to add to this section also :-)

2.5 UX Design

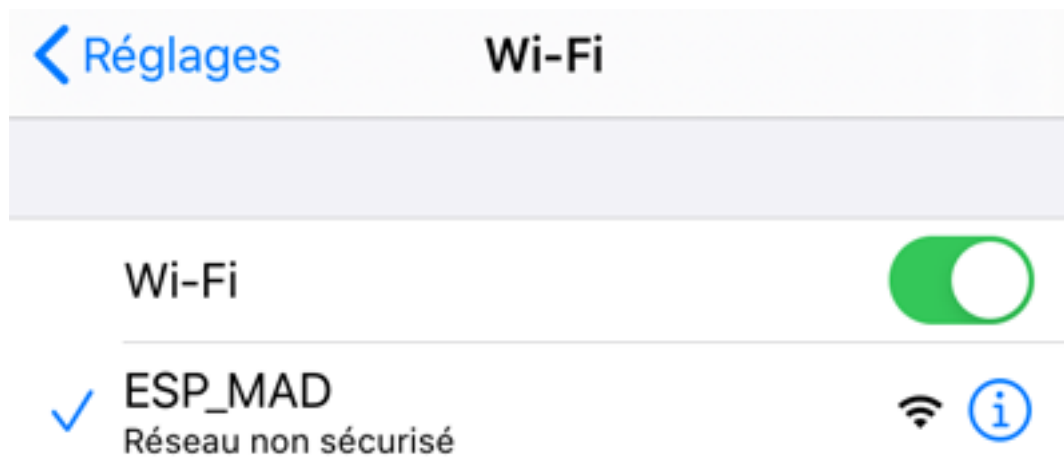
The man-machine interface (MMI) of the project consists of a single HTML page (esp.html).

This page is built using the CSS framework [bootstrap](#).

The page embeds an ajax script which periodically makes a HTTP GET request to the “Server” board which sends back the different information to be displayed in the page. A second script makes it possible to carry out the change of the chord of the control surfaces by a HTTP POST request. A third script is used to reset the Maximum(s) up and down travel on the travel tab.

All the files for MMI are located in the directory GliderThrowMeter/Esp_mad_Server/main/WebsiteFiles

To connect to the page, it is first necessary to connect to the Wifi ad’hoc network of SSID ESP_MAD.



Then, just type the address 192.168.1.1 in the URL bar of your browser to connect to the main page of the project.



The main page of the project contains 4 tabs : Travel, Angle, Setting & Info.

The travel tab displayed the current travel of each sensors, and the Maximum up and down for each sensors stored during the operation. The Reset Maximum(s) button is used to set to 0 these Maximum.

The “Angle” tab selection causes the page showing the deflection angles for both board to be displayed.

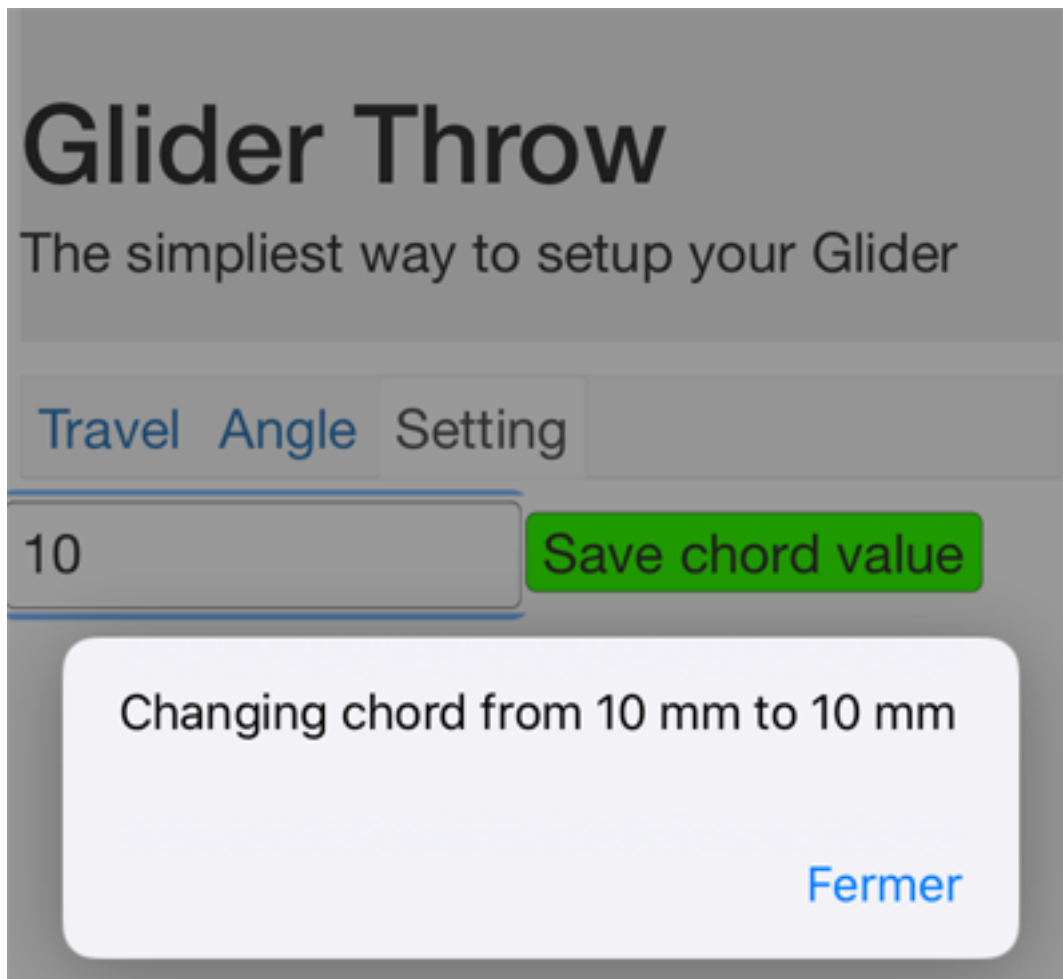
The screenshot shows the 'Glider Throw' web interface. At the top, the title 'Glider Throw' is displayed in a large, bold, black font, followed by the subtitle 'The simplest way to setup your Glider' in a smaller, regular black font. Below this, there are three tabs: 'Travel', 'Angle', and 'Setting'. The 'Setting' tab is currently selected and highlighted in blue. Under the 'Setting' tab, there are three colored bars with text: a pink bar for 'Sensor 1 angle : 0.1 degrés', a light blue bar for 'Sensor 2 angle : 0 degrés', and a green bar for 'Delta angle : 0.1 degrés'.

The “Setting” tab will display the page that allows you to change the value of the control surface chord.

This screenshot shows the 'Glider Throw' web interface with the 'Setting' tab selected. The title and subtitle are the same as in the previous screenshot. Below the tabs, there is a text input field containing the number '50'. To the right of the input field is a green button with the text 'Save chord value'.

Note: In the current version, the project allows to control only one “Client” and both boards deal with the same chord value.

To change the value of the chord, modify the value in the input field and validate with “Save change chord” button.



Finally, the “Info” tab display the voltage of the battery for both sensor.

Glider Throw

The simplest way to setup your Glider

Travel Angle Setting Info

Sensor 1 Voltage : 3.98 V

Sensor 2 Voltage : 0 V

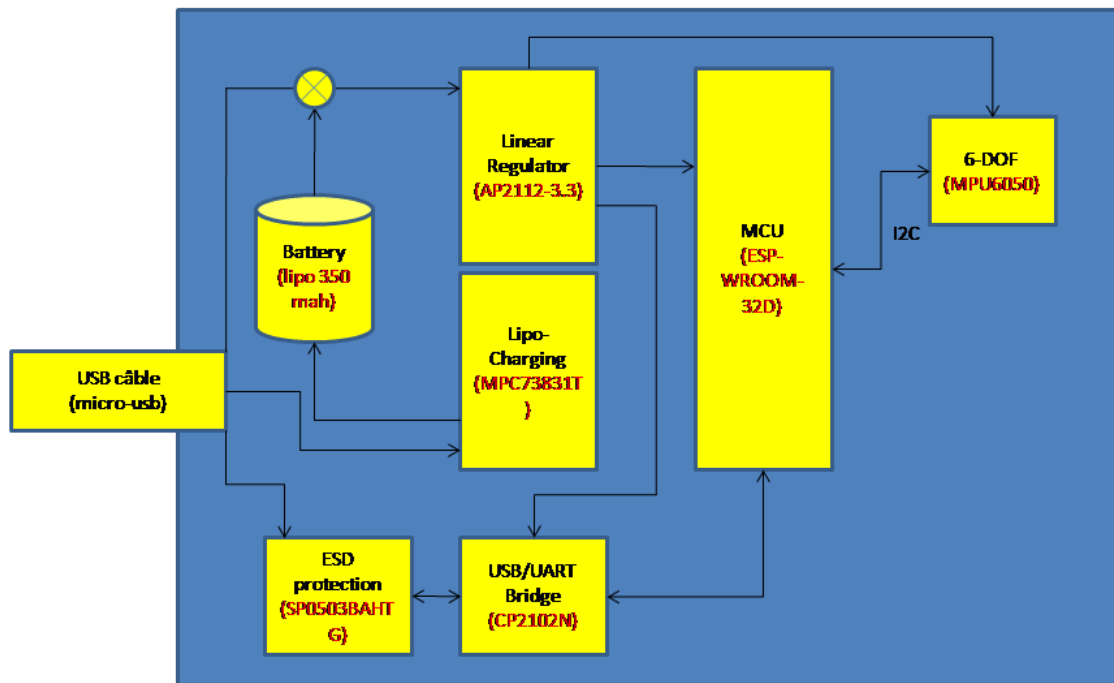
Let's move on to the next chapter for the description of the board design.

CHAPTER 3

Hardware Design

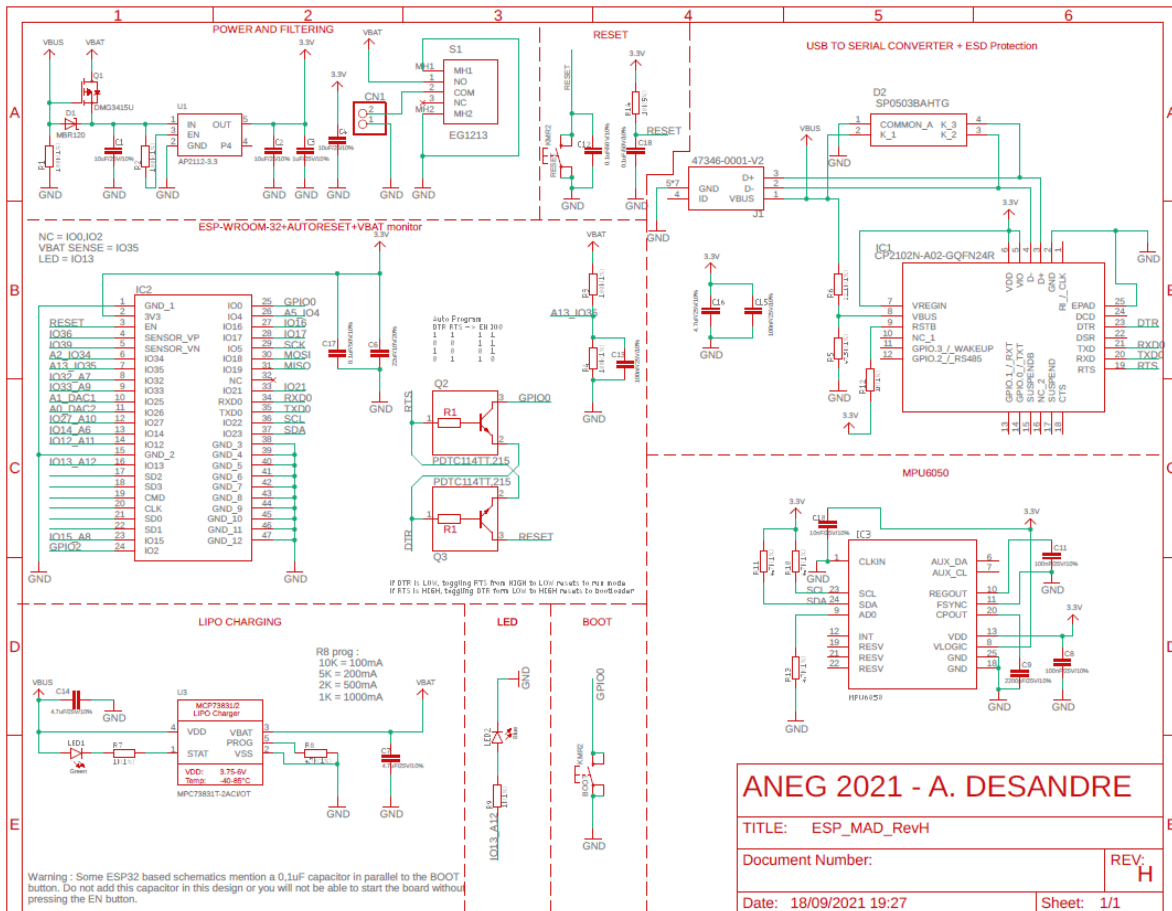
The hardware design is based on the [Adafruit HUZZAH32 ES32 feather open source board](#)

The general architecture of the system is shown below.

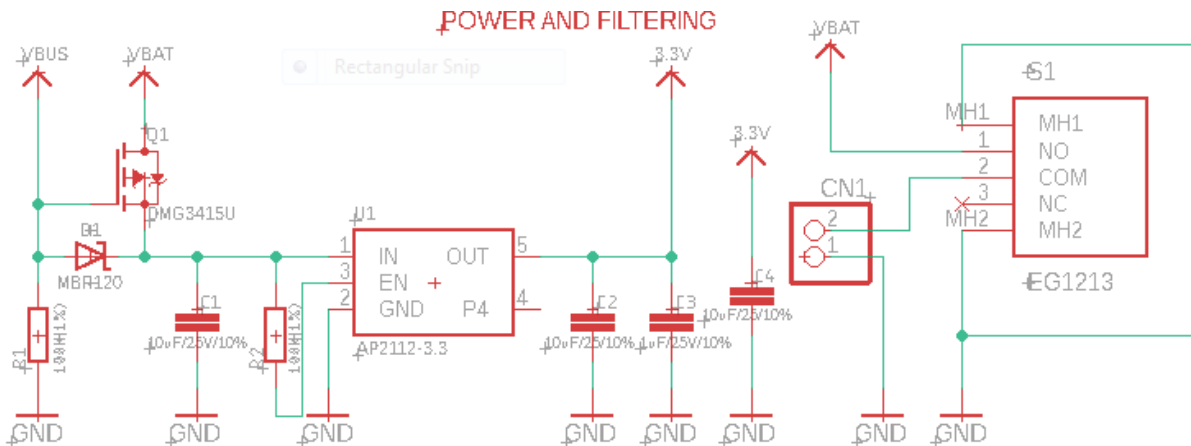


The design is broken down into seven major sections :

- Power supply and filtering,
- Lipo charging,
- USB to serial converter + ESD protection,
- MPU6050,
- ESP-WROOM-32D & Autoreset,
- Reset circuit,
- Boot circuit,
- Adressable LED.



3.1 Power supply and filtering



The board can be powered from a 5V USB port (VBUS), or from a 3,7 V LIPO (or Li-Ion) 1S battery.

The battery is connected to a switch that allows the battery supply to be turned ON/OFF.

A DMG3415U (MOSFET transistor) is used to switch between VBUS and VBAT. When VBUS is not present, the gate is pulled low, and the MOSFET shorts out the body diode, connecting VBAT directly to the LDO. When VBUS

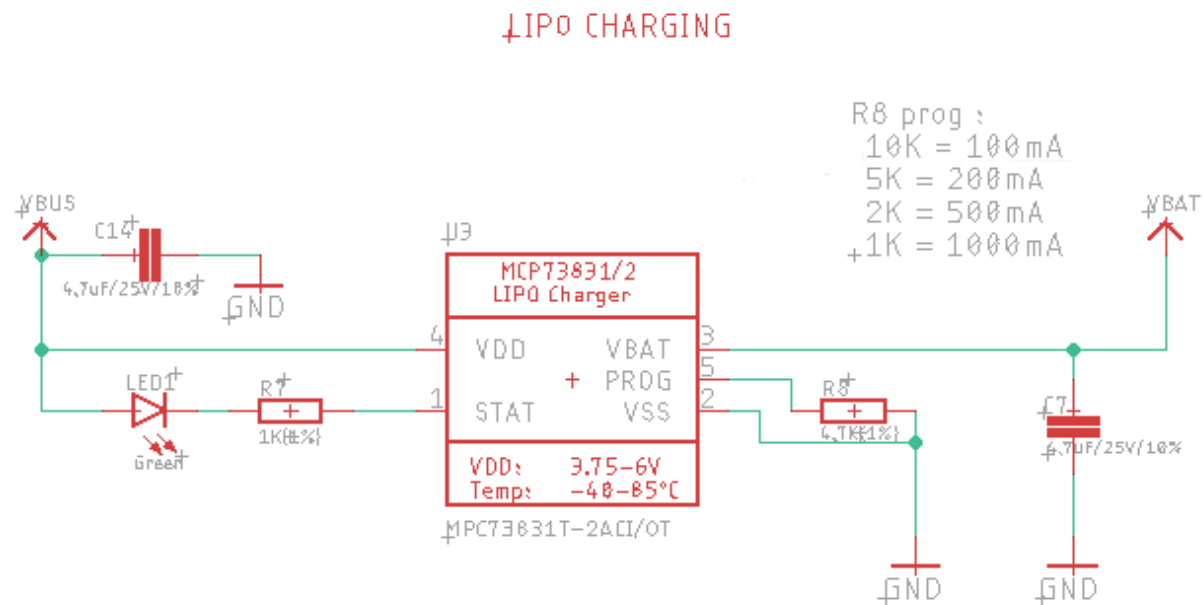
is greater than VBAT (that is our case if the board is connected by usb), the MOSFET is cut off and the body diode is blocking, disconnecting VBAT from the circuit. EN pin of the DMG3415U is pulled low to permanently enable the chip.

So with this switch, VBAT enters to the AP2112-3.3v LDO, if VBUS isn't present, otherwise VBUS enters to the AP2112-3.3.

3.2 Lipo charging

The lipo charging circuit is based on the MCP73831/2 microchip. This chip is a miniature single cell, fully integrated Li-Ion, Li-Poly charge management controllers. Typical Application schematic is used.

Note: R8 resistor is used to set the current regulation. As we will use battery around 350 mA, we fix R8 to a current regulation around 200 mA.



3.3 USB to serial converter + ESD protection

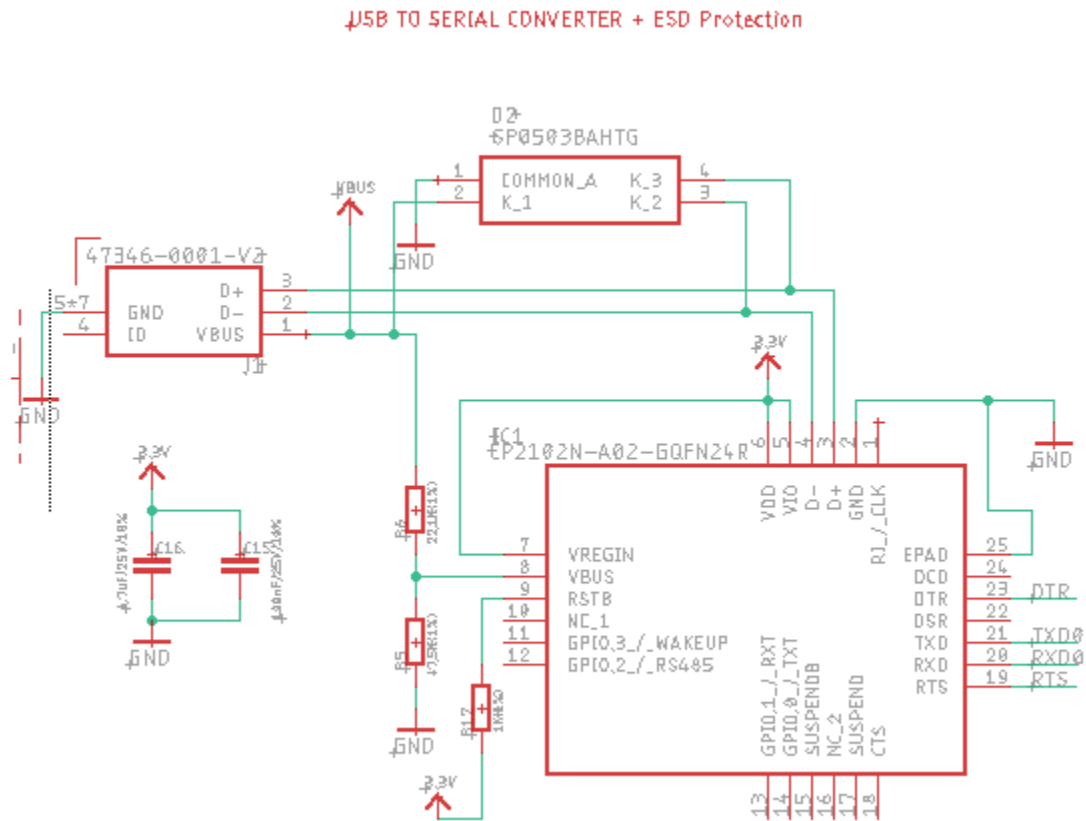
The USB serial converter is based on a CP2102N from Silicon Labs.

ESD protection is done using a SP0503BAHTG from littlefuse as recommended on the datasheet.

VEREGIN, VDD & VIO pins are tied to +3.3V, and also RSTB pin as recommended on the datasheet.

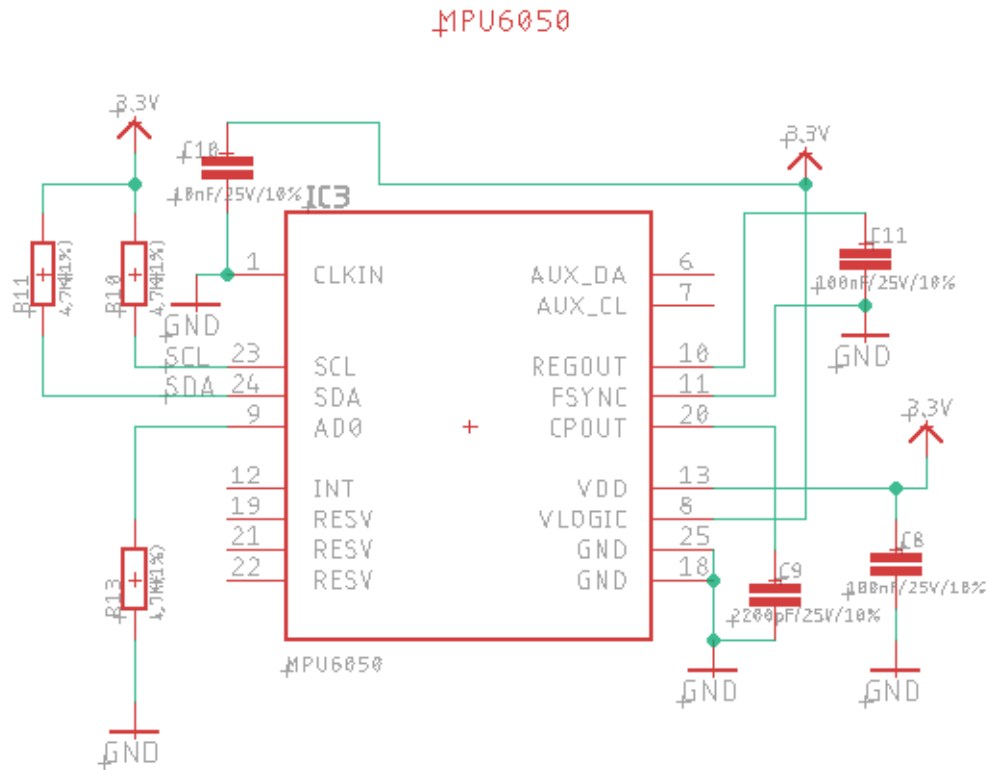
Two decoupling capacitors are also used.

To detect when the device is connected to a bus, which is defined as $V_{IO} - 0.6\text{ V}$, a resistor divider on V_{BUS} is required to meet these specifications and ensure reliable device operation. In this case, the current limitation of the resistor divider prevents high V_{BUS} pin leakage current, even though the $V_{IO} + 2.5\text{ V}$ specification is not strictly met while the device is not powered.



3.4 MPU6050

The circuit for the MPU6050 is a typical application scheme (see datasheet). SDA and SCL pins are connected to the pins 22 & 23 of the ESP-WROOM-32D with two pullup resistors.



3.5 ESP-WROOM-32D & Autoreset

ESP-WROOM-32D chip, is the last ESP-WROOM-32 update from espressif. As our board is design with built-in USB to Serial converter, we will use esptool.py to automatically reset the board into bootloader mode. esptool.py can automatically enter the bootloader by using the RTS and DTR modem status line to toggle GPIO0 and EN automatically.

EN pin forces the ESP32 chip to reset and the ESP will enter the serial bootloader when GPIO0 is held low on reset. Otherwise it will run the program in flash.

Note: GPIO0 has an internal pullup resistor, so if it is left unconnected then it will pull high.

We use two PDC114T (an NPN transistor with resistor) to control the ESP32 Boot mode Selection.

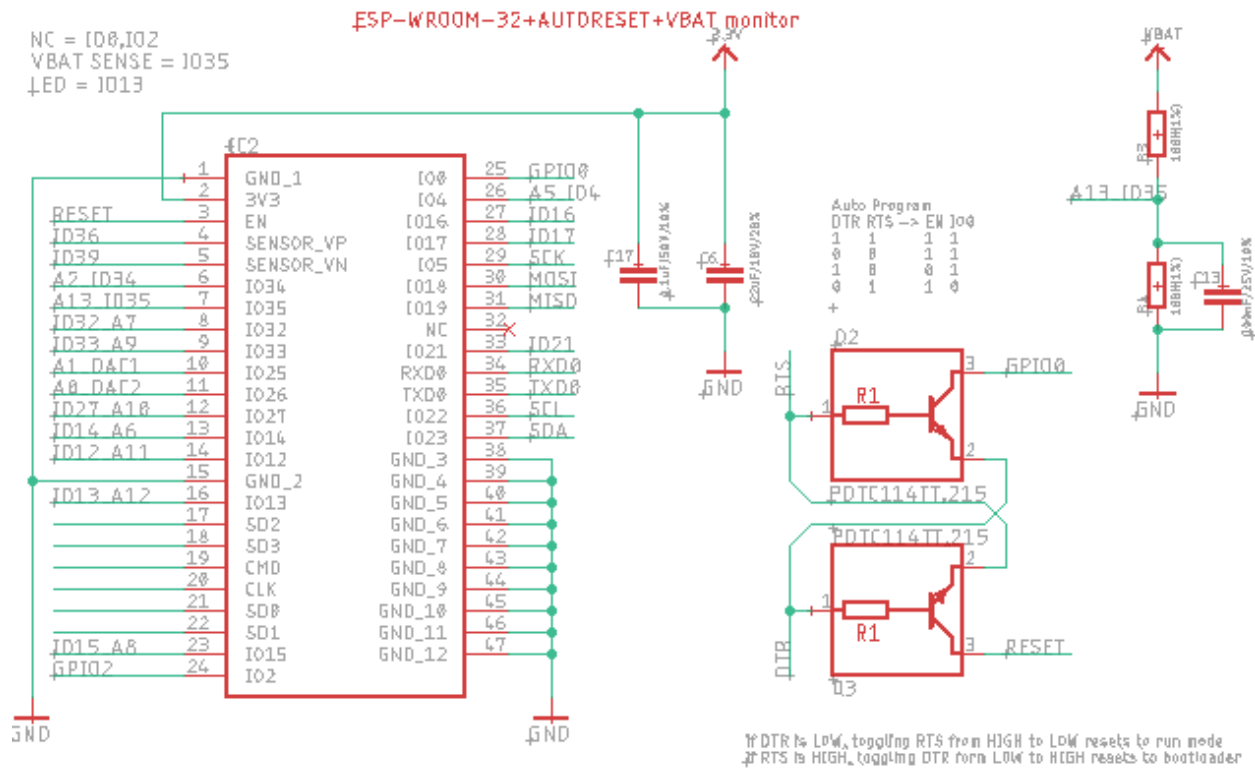
We have DTR controlling the base of a transistor whose collector is connected to RESET. We have RTS connected to the base of a transistor whose collector is connected to GPIO0. Remember, there is an external pullup resistor on RESET so default is HIGH.

When DTR is set HIGH and RTS is set LOW, this pulls RESET to LOW and GPIO0 is not controlled so it will eventually take its strapped value of HIGH. This has the same result as assuming DTR connects to GPIO0 and RTS to RESET. The processor is in the reset state.

When DTR is set LOW and RTS set HIGH, this disconnects RESET from the transistor and it gets pulled HIGH by the external pullup resistor. At the same time, GPIO0 is pulled to LOW by the transistor (with its internal pullup still engaged). This has the same result as assuming DTR connects to GPIO0 and RTS to RESET. The processor comes out of reset state and reads the GPIO0 value to be LOW to start the bootloader.

Battery voltage is measure using a a voltage divider bridge connected to the IO35 pin.

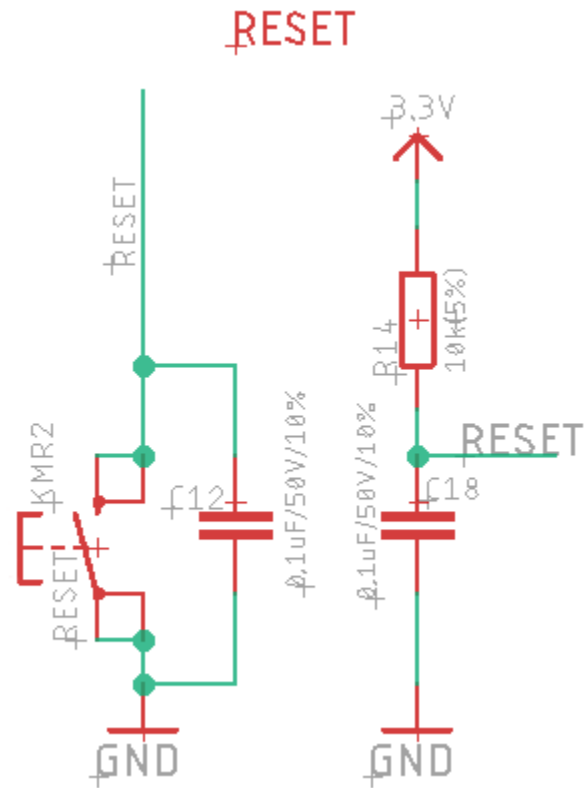
Most of the other pin are not used, unlike IO22 & IO23, set respectively on SCL and SDA I2C signal to communicate with the MPU6050.



3.6 Reset circuit

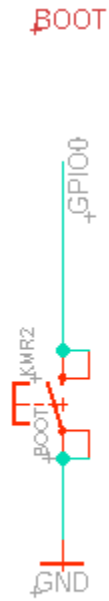
Enable (EN) is the 3.3V regulator's enable pin. It's pulled up, so connect to ground to disable the 3.3V regulator. So we connect this pin a pushbutton to restart your ESP32.

As recommended by espressif a RC circuit with a resistor of 10k and a capacitor of 0,1uF is added between EN pin and +3,3V to make automatic reset more reliable.



3.7 Boot circuit

Boot switch is connected to GPIO.



Note: Some ESP32 based schematics mention a 0,1uF capacitor in parallel to the BOOT button to debounce. Do not add this capacitor in this design or you will not be able to start the board without pressing the EN button.

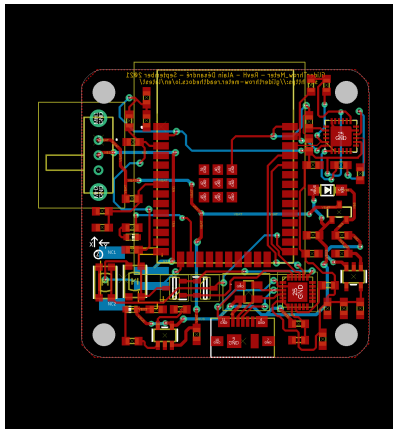
3.8 Adressable LED

The adressable led is connected to the pin IO13 of the ESP-WROOM-32D. This led is used to display the status of the MPU6050 calibration.

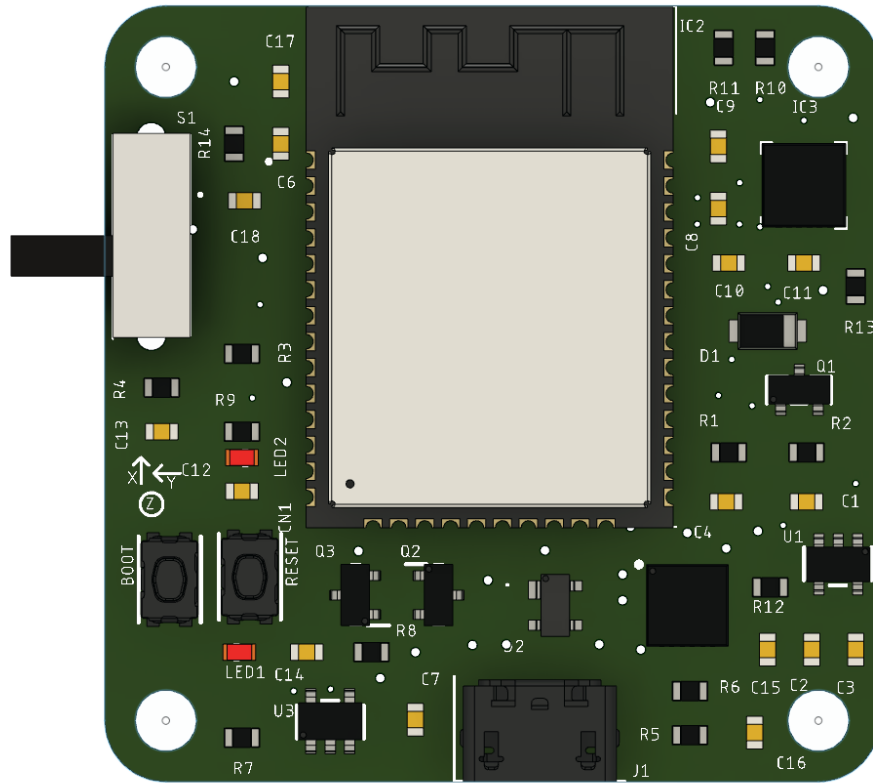


3.9 PCB routing

The routed PCB (without ground plan) is shown below. The routing was done under EAGLE.



3D made with fusion 360 is shown below.



The finish board is shown below (quite similar to the 3D model isn't it :-)



3.10 Bill Of Material, Eagle Files & Gerber

BOM can be downloaded at this link [bom.xlsx](#)

Pick & Places file can be downloaded at this link [Pick&Place file](#) The format of this file is compatible with the Smt Assistant utility from Alciom. SmtAssistant is a software designed by ALCIOM and helping to locate a part on a printed circuit board, based on a bitmap of the PCB and Pick&Place file. SmtAssistant is a useful for manual prototype assembly, inspection or board repair works. [see here](#) for more information and download.

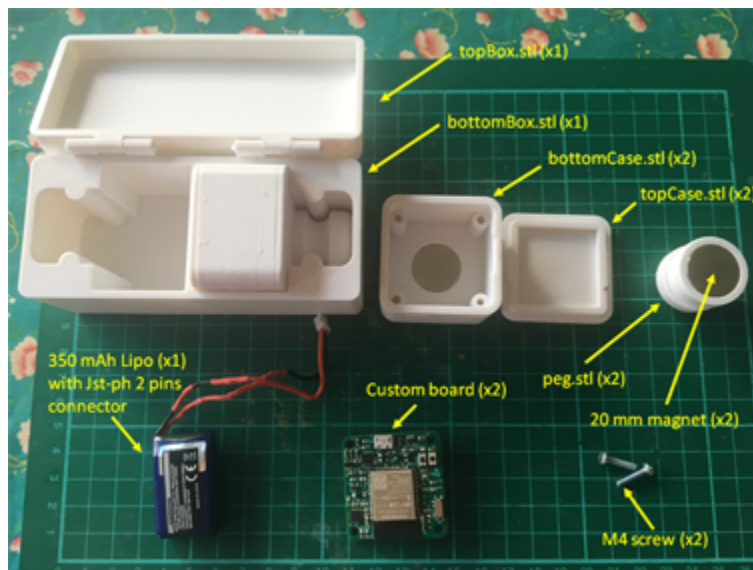
Gerber files can be downloaded at this link [ESP_MAD_Gerber.zip](#)

Eagle files can be downloaded at this link [eagle-files.zip](#)

Let's move on to the next chapter for the description of the system assembly and its use.

4.1 What do you need

Building the components for the project is extremely simple and requires only the following parts.



4.1.1 Stl files

All the stl files can be downloaded at this link [zip file](#)

4.1.2 Custom board

The custom board can be ordered from [seed fusion](#) for example (or other PCB services like Eurocircuit, jlcpcb, etc). Gerber files can be downloaded on the hardware section. I have already made several PCBs at seed Fusion and the service is fast and good. The price for 10 PCBs is around 15€ + the shipping costs. When ordering the PCBs, don't forget to order the stencil to apply the solder paste.

Then you have to get the necessary components to assemble the PCB (the BOM is given in the hardware section, and Mouser or Digikey is your friend), assemble the components either with a brussel clamp or using a Pick & Place machine, then use a reflow oven to solder the components. If you assemble the board manually, first pass the board with all the small components through the reflow oven and solder the ESP32 and the JST-PH connector on the bottom of the board by hand.

Another solution if you don't want to waste time manually assembling the board is to use the PCB Assembly service from seed Fusion. Under this service, seed Fusion will source your components based on the BOM, manufacture your PCB and do the assembly and soldering of the PCB. The disadvantage of this service is that you will not be able to order less than 10 boards and it will cost you about 140€ for these 10 boards + the shipping costs.

4.1.3 20 mm magnet

20 mm neodym magnet can be ordered at [supermagnet.fr](#)

4.1.4 Lipo 1S battery

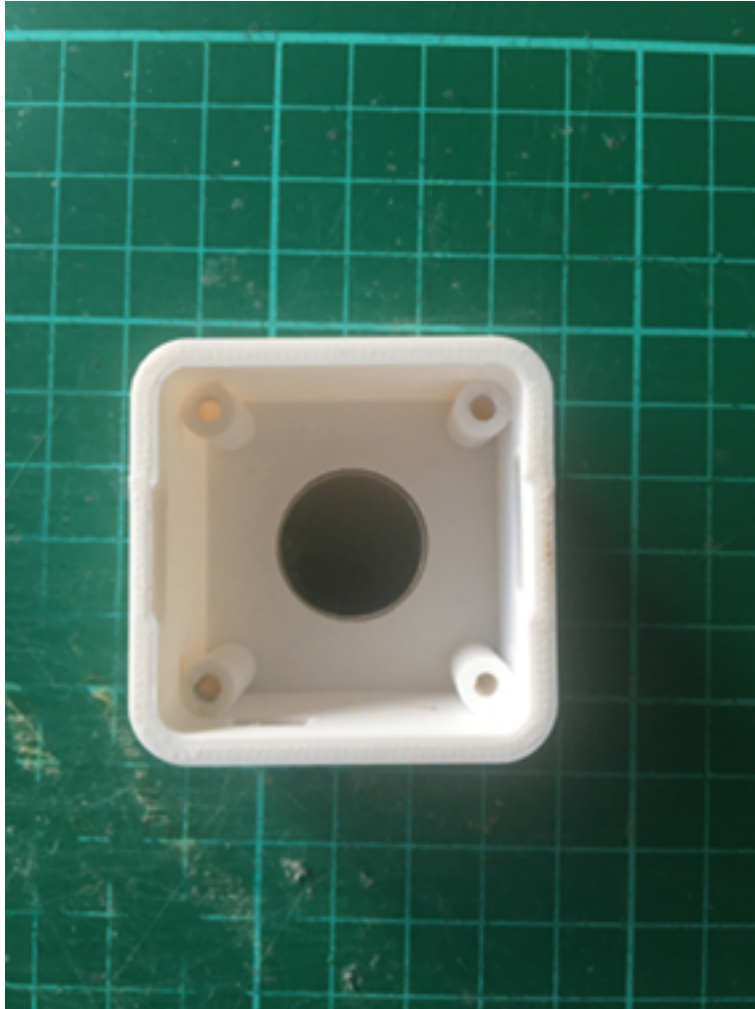
A 350 mAh Lipo battery is recommended. 500 mAh Turnigy lipo from hobbyking is also a good choice.

With the turnigy 500 mAh battery, the autonomy of one device is around 3h of use.

4.2 How to build assemble one board

4.2.1 Step 1 : Mounting the magnet in the lower part of the case

Use some epoxy glue to secure the magnet.

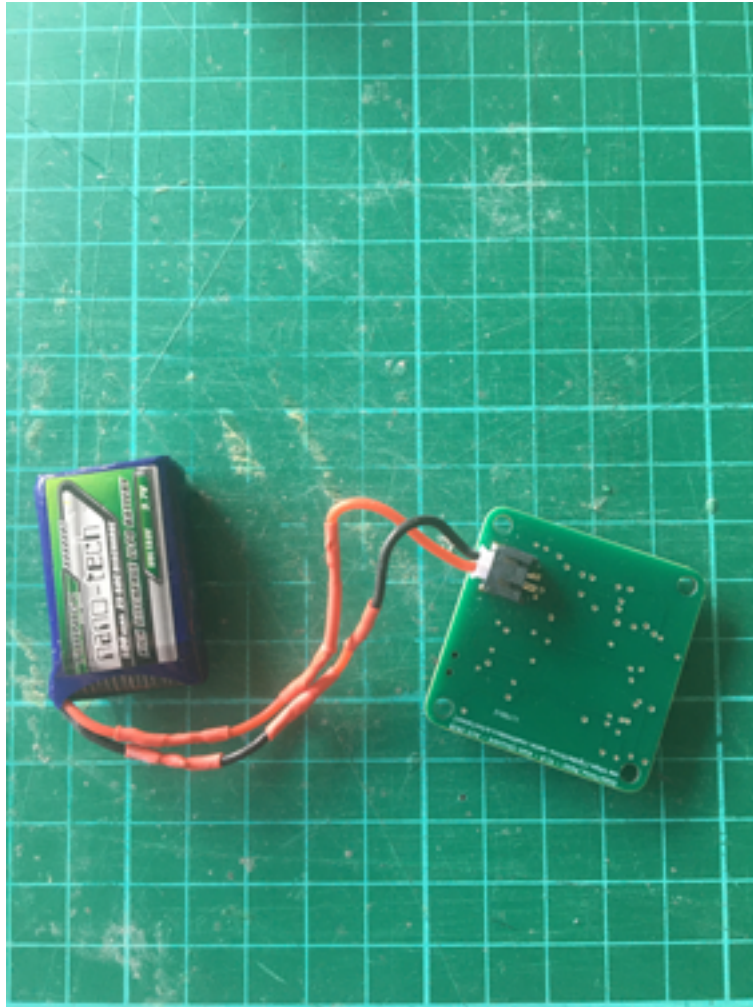


4.2.2 Step 2 : Mounting of the magnet in the base part of the peg

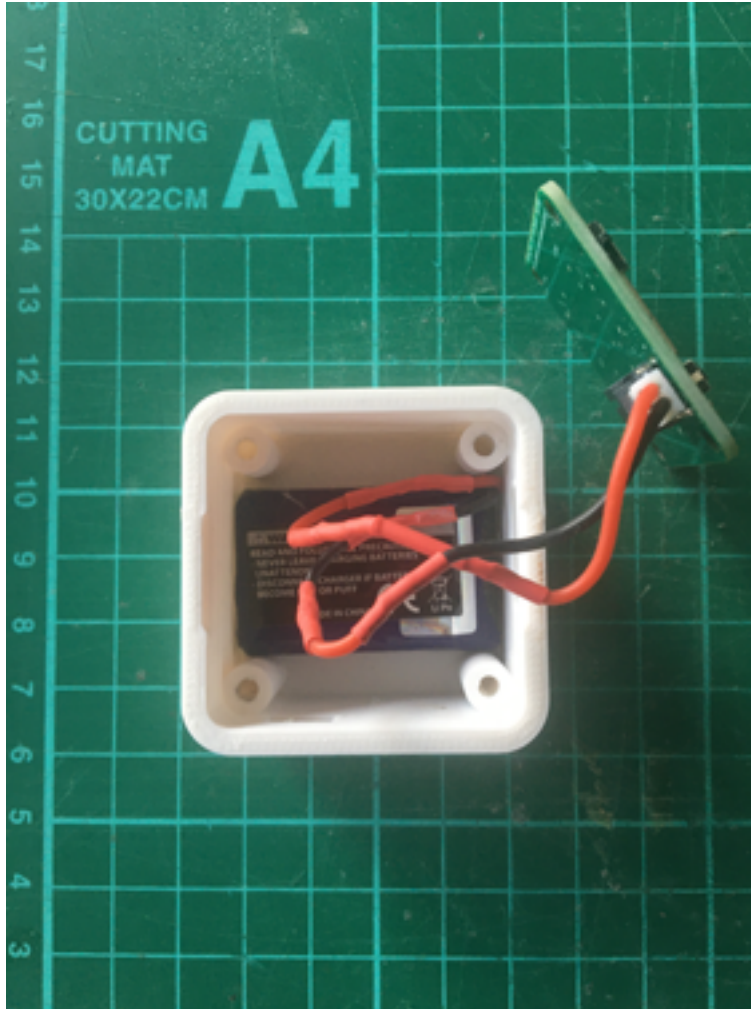
Use some glue also to secure the magnet.



4.2.3 Step 3 : Connect the Lipo battery on the custom board



4.2.4 Step 4 : Install the lipo battery on the bottom case



4.2.5 Step 5 : Screw the custom board on the bottom case

Screw the board using only two screw.



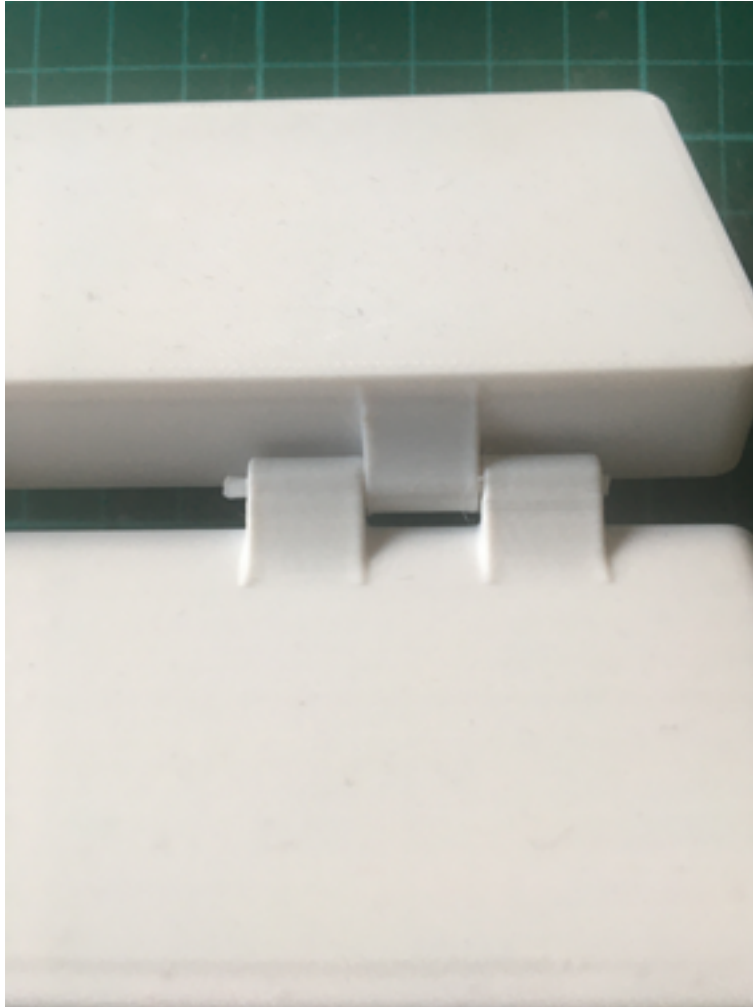
4.2.6 Step 6 : Put the top case on the bottom case

Two snap fit are provided to secure the top case and facilitate the disassembly



4.2.7 Step 7 : Assembly of the storage box

The bottom and the top of the box are assembled with two hinges through which a 1.75 mm PETG wire is passed.



The box is designed to receive a set of two devices.



4.3 How to use the devices

TBD.

CHAPTER 5

Indices and tables

- `genindex`
- `modindex`
- `search`