Remote control for CPG-based robot

Laura Balthazard

Professor: Auke Jan Ijspeert

Assistant: Alessandro Crespi

Semester project at the Biorobotics laboratory

Abstract

The goal of this semester project is to continue the development of a wireless remote control that has been started by Matthieu Tardivon three years ago. The remote must be able to control in real time the locomotion of the Amphibot III robot and to upload a new firmware from a SD card to the robot. The first step was to analyze the previous remote in order to find where the problems were and what was necessary to improve. The second step was the implementation of a working software.

Content

Introduction

This project concerns the Amphibous robots and mainly the Amphibot and Salamandra robotica robots.

At the very beginning, these robots were controlled via a computer connected to a radio interface and a joystick. However, during presentations outside of the laboratory, having all this equipment was not practical. This is the reason why a student, Gabriel Cuendet, made an autonomous remote which can communicate with the robot through the radio interface without any computer, see Figure 1.

The remote he made is working and is often used for demonstrations of the Amphibot III. The interface contains a LCD screen, four buttons and one joystick. However, it is limited in functionalities.

First, the design is not optimal. Indeed, the remote held narrowly in one hand and the other one needs to be used to move the joystick. With this configuration, it can take time for the user to become familiar with the use of this remote.

Then, a new need was identified: being able to flash to the robot a new firmware. For example, on a SD card, we could put a working firmware of the robot, and during a presentation even if someone has changed the robot firmware, we could always flash the right one to the robot without a computer. In the future this memory could also be used for saving data from the robot, as for example the actual trajectories of the motors.

In order to answer to these objectives, another student, Matthieu Tardivon, started to make a new remote, see Figure 2. To be more ergonomic, this one has the geometry of a PlayStation remote. It also has two joysticks so that we can have one to control the speed and the other one for the direction. Having a separate control of the speed and of the direction can be, for some users, more intuitive. Finally a hardware part for the SD card has been added.

Unfortunately, the remote was not finished and not working. First the shell needs to be improved, some parts are too thin and some holes are not well designed, for example a hole for the screen has been cut after the printing of the mechanical part. Then the remote was not able to control the robots.

Figure 1 - G. Cuendet's remote[1]

Figure 2 - M. Tardivon's remote

[1] Source: G. Cuendet's project presentation

1. State of art

This project started with a box filled of all the components of the remote to put together, and the code and the Solidworks parts given on Biorob's website.

The first problem was to find why the remote was not working, whether the problem came from the sofware or if the PCB had some problems.

Then, the next task was to go step by step into the software in order to understand what was going wrong.

1.1. Hardware part

After a review of Matthieu's board, the only changes made were to replace the cable connection for the battery and for the LCD. The connection for the battery has burned thus a new cable has been welded and the welds on the LCD pins were not well done so new ones have been made with a heat-shrinkable polymeric tube to avoid short-circuits.

Otherwise, the board is working. For further details of the choices made for the components of the board, Matthieu's report is available on the Biorob's website.

This part will quickly explain the interface for the user made by Matthieu.

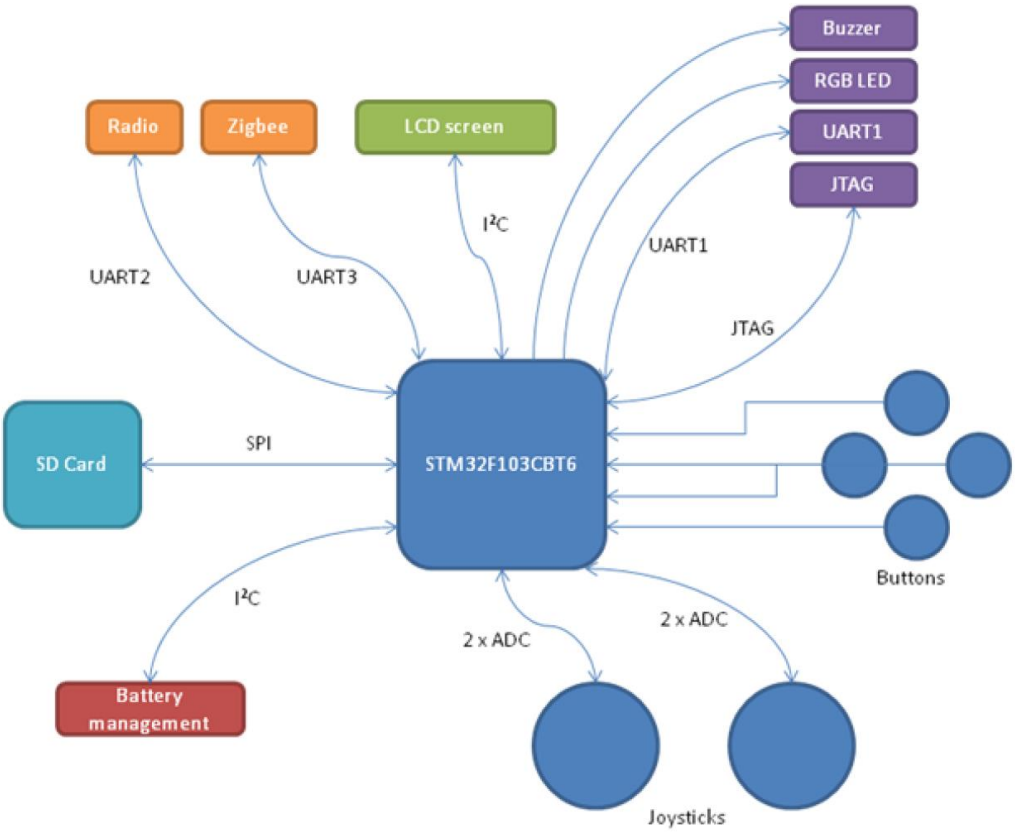

Figure 3 - Overview of the connections of the board[2]

This remote is composed, among others things, of a STM32 microcontroller, the STM32F103CBT6. It can communicate with the user through a LCD screen and four buttons. It can also communicate, thanks to the radio interface, with robots like Amphibot III or others based on the same hardware, for example Salamandra robotica or ENVIROBOT. This communication allows us to move the robot or to flash it.

1.1.1. Communication with the user

To interact with the user, informations are given through a screen. The LCD chosen is the Batron BTHQ21605V-FSRE-I2C-COG. It communicates with I2C bus. To navigate through the menu displayed on the screen, four buttons can be used. The up and down ones are for scrolling the menu, the left one for going to the previous menu and the right one for validating the choice. We can also find LEDs on the board that can help the user to know what is happening. For example, one led is turn on when the radio communication is activated or when there is a problem. There is also one buzzer, for the moment one bip is made when the remote is turned on and there is a menu to play music but we could think of more applications.

[2] Source : M. Tardivon's project report

1.1.2. Control of the robots

In this remote, there are two joysticks instead of one like the previous remote. This way the user can control the speed of the robot with the y axis of one joystick and the direction with the x axis of the other one. This way, the users should be able to learn more quickly to control the robots.

1.1.3. Communication with the robots

To communicate with the AmphiBot and other robots based on the same hardware, the remote control uses a nRF905 radio transceiver chip controlled by a PIC microcontroller, which implements the custom communication protocol used by the robot.

Moreover, to make the remote control more generic, allowing it to potentially control more types of robots, Matthieu also started to add a module to communicate through zigbee. However, this project won't go further in the development of this part.

1.1.4. Other

To help debugging the firmware of the remote control, a serial port connector has been added to allow, for example, displaying messages in a terminal.

1.2. Software part

The code made by Matthieu used a real time operating system named ChibiOS/RT. It allows doing several tasks, named a thread, at a time, like controlling the robot and navigating in the menu. As the main part of the software had already been started, the fastest solution to make the remote works seemed to be to continue with this operating system.

This part will explain how the code has been implemented.

1.2.1. Template of the code

For the structure of the code, the one made by Matthieu was kept. Namely, each physical peripheral has its own file with its driver in it. For example, in the file buzzer.c we can find the configuration of the PWM and the function to play Mario's music. Thus, there is a file for the menu, the joystick, the radio communication, the zigbee communication, the buttons, the LCD, the reading of the SD card, the flashing of the robot and one file for each robot.

The important part of the code is in the file menu. Here, the sequence of events is chosen.

1.2.2. Menu

In the main, the function menu_init() is called. In this function, the tread menuThread is activated. This thread controls the configuration of the menu and actions to do if one menu is selected. Further details about the structure of this thread are available on Matthieu's report.

To control the sequence of events, there are different states. In brief, each state has its own table of structure MENU_ENTRY:

```
typedef struct
{
  char menu_buffer[BUFFER_DEPTH];
  MENU next_menu;
} MENU_ENTRY;
```

For example, the menu Connection, where we can choose how we want to connect to the robot, is defined as following:

```
  //CONNECTION
  menu_entry_define(&menu_connection[0], "Radio connection", RADIO);
  menu_entry_define(&menu_connection[1], "Zigbee connection", RADIO);
  menu_entry_define(&menu_connection[2], "Back to main menu", MAIN);
```

1.2.3. Radio communication and flash option

To understand how the radio communication works, with the writing and reading registers, it is suggested to read Gabriel's report since Matthieu took his code. For the reading of the SD card, the flashing part and the use of the corresponding ChibiOS driver called MMC over SPI, suggestion is made to look in Matthieu's report. Even if there were some errors with the code, the structure was correct.

2. From the study to the working state

The first step was to have a proper working environment. A step up of the remote is now available in Annexe II, if you want to continue this project.

Then, the longest part of this project was to understand how ChibiOS works and to debug the code. Indeed, as Matthieu did not give all the necessary files, time was lost in order to have a proper working environment.

Once everything was in order, some improvements have been done.

2.1. ChibiOS RT

The code given by Matthieu used an old version of ChibiOS, 2.4.2, which is not available anymore. Unfortunately in the last version, the name of some functions and defines were changed. Finally, ChibiOS_3.0.2 was used but everything had to be checked and updated so that the software works with the last version. A lot of time was spent updating the code and making it works. In order to avoid this lost time, the files of the ChibiOS_3.0.2 version will be available on the Biorob's website.

For example, the syntax to create a thread used to be:

```
static WORKING_AREA(lcdLine0Thread, 256);
static msg_t lcd_line0_thread(void *arg)
{ … }
```

With the 3.0.2 version, the right syntax is:

```
static THD_WORKING_AREA(lcdLine0Thread, 256);
static THD_FUNCTION (lcd_line0_thread, arg)
{ … }
```

In order to bring the remote in a working state, the code had to be reviewed file after file with the help of the documentation. That part was really time consuming.

2.1.1. Configuration of ChibiOS

With ChibiOS, one can create multiple threads, corresponding to a task, that can run at the same time without having to manage how the controller handles it.

Each project based on ChibiOS must contain four files, which need to be updating depending on the pins of the microcontroller used:

- chconf.h: containing configuration related to kernel. For example, here it is possible to configure system timing, enable/disable kernel features and enable/disable debug options;
- halconf.h: containing configuration related to HAL. Here it is possible to enable/disable whole HAL modules or configure them: ADC, I2C, UART etc;
  If one wants to use the I2C connection:

```
/**
 * @brief   Enables the I2C subsystem.
 */
#if !defined(HAL_USE_I2C) || defined(__DOXYGEN__)
#define HAL_USE_I2C                 TRUE
#endif
```

- mcuconf.h: containing specific configuration of the microcontroller used. For example, if one enables the UART module in halconf.h and that the microcontroller has three UART ports, the choice of how many drivers assign to/enable for our purpose will be done here.
  If one only wants to use the UART 1:

```
/*
 * UART driver system settings.
 */
#define STM32_UART_USE_USART1               TRUE
#define STM32_UART_USE_USART2               FALSE
#define STM32_UART_USE_USART3               FALSE
```

These files need to be added to the project directory.

The last important file is located in the ChibiOS directories. As it wasn't given by Matthieu, the creation of this file has to be done.

- board.h: contains all the configurations of the input and output of the board used. The right configuration of this file is very important for a correct working of the remote control. For example, if one wants to use UART bus, the UART TX pin, which transmits the informations, needs to be set as an Alternate output and the UART RX pin, which receives it, as a Normal input. The I2C_SCL and I2C_SDA need to be in an Alternate output mode. In the used version of ChibiOS, this file is in the directory os/hal/boards/ OLIMEX_STM32_P103. Other files were available, but the choice of

this board was made to have a base that could be modified according to the current board.

Another possibility to configure the pins, instead of configure it directly in the board.h file, is to use, in the code, the function `palSetPadMode(port, pad, mode).` For example, in the main, the mode for the red LED is configured as following:

```
/*
* Configuration for the led
*/
palSetPadMode(GPIOA, LEDR, PAL_MODE_OUTPUT_PUSHPULL);
```

2.2. Template of the code

In the main, the call of the functions was reorganized since in the previous code all the initialization functions were in comments.

At the beginning, the main calls the initialization function of the buzzer, the LCD, the radio and the buttons. Then it enters in the function menu_init() that, as said before, activates the thread of the menu that controls the states.

The different states of the menu have been slightly changed according to the Figure 4 - Structure of the code.

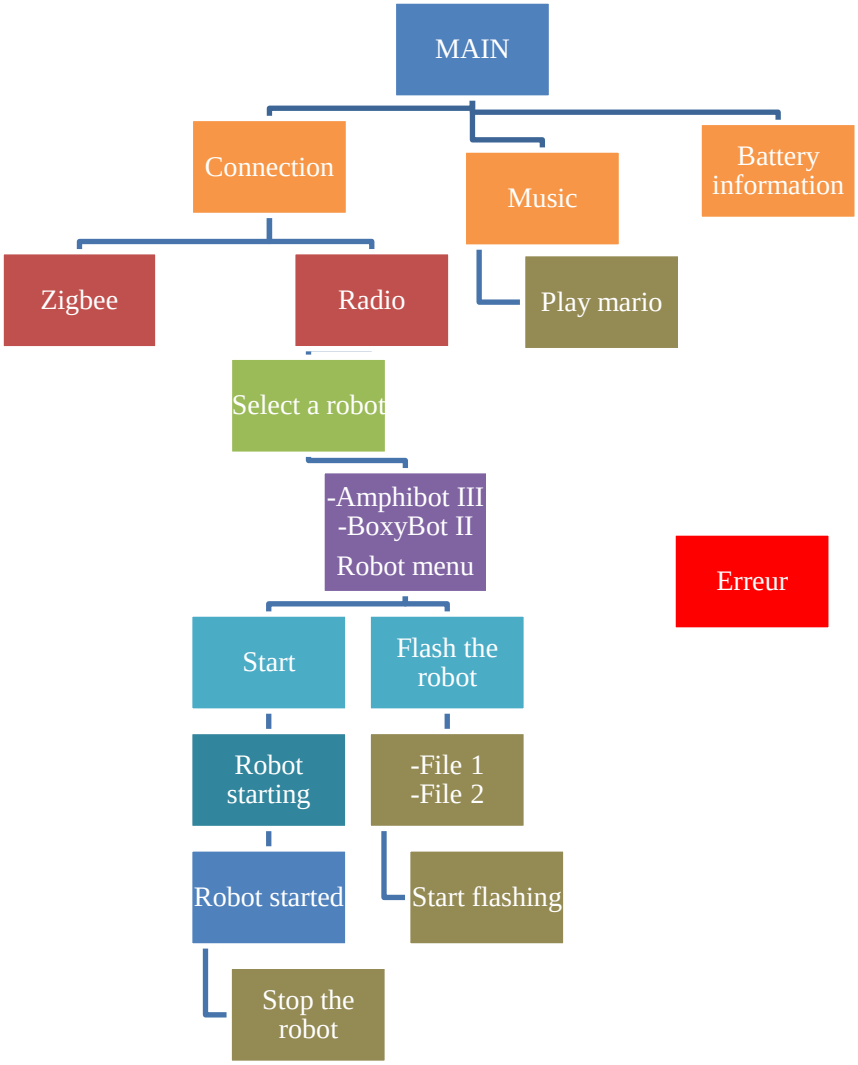

Figure 4 - Structure of the code

The real changes made are in the functions called in the menus **Music**, **Robot menu, Robot starting, Robot started** and **Error.**

In each menu there is the possibility to go back to the main menu thanks to: “Back to the main menu” entry.

First when we turn on the remote, we arrive on the **main menu**. This menu allows us to connect to the robots, play some music or have some informations about the battery.

Connection: if we validate this menu by pushing the right button, we can choose the way to connect the robot: by the radio or by the zigbee interface. However, the zigbee communication has not been developed, only the name of the menu appears but nothing can be done.

Music: This menu is not useful to control the robot but one can play melody. The Mario's song was already implemented by Matthieu, thus the code to play it if the button right is pressed has been added.

Battery: This menu is here in order to be able to know the state of the battery. However the functions need to be made.

Radio: Once the radio communication is chosen, one has the possibility to select a robot.

Select a robot: In this menu, the names of the robots implemented in the remote are displayed. For the moment, there are the Amphibot III, on which we can control the locomotion, and BoxyBot II, a small robot used in the laboratory for practical works and used in this project to test the flashing mode.

Robot menu: Once one of the different robots is chosen, one can start to move it or flash a code through the SD card. Here, a new part of code checks whether the robot is on or not. If it is not the case, the user enters into the **Error**.

When the "Start the robot" submenu is validated, the user is redirected to the menu Robot starting.

Robot starting: This is just a transition state, where the initialization of the robot is done. Then, if everything is okay, it directly goes to **Robot started**.

Robot started: Here the function to read the value of the joystick is called, then depending on which robot is selected, different functions are called to move or stop the robot. In order to stop it, the user has to validate the submenu: "Stop the robot".

Flash: When this menu is selected and if an SD card is inserted, one can explore it. Once the wanted file has been found (it has to be a .bin file), one needs to press the right button to flash it to the robot.

Error: When an error occurs, for example, no SD card or the robot is not turned on, the user arrives in this menu and the red LED is on. All the errors are implemented as a submenu of Error. To go back to the previous menu, one just needs to press the left button.

2.3. Changes of the previous code

There are two kinds of modifications. First there are the small corrections that allowed to remove the compilation, faults probably caused by the change of version.

Then, the second part will focus on the new functions that have been implemented.

2.3.1. Corrections made

Most of the errors were due to a wrong initialization of the HAL module. For example for the buzzer that uses a PWM, an initialization of a parameter was missing.

For the MMC, MultiMediaCard, which is the driver of ChibiOS for the connection of memory card, all the calls of the ChibiOS functions were wrong.

Matthieu's version:

```
uint8_t mmc_init(char error_msg[])
{ …

mmcObjectInit(&MMCD1, &SPID2, &ls_spi2cfg, &hs_spi2cfg, mmc_is_protected,
mmc_is_inserted);
mmcStart(&MMCD1, NULL);
…
f_mount(0, &MMC_FS);
…
}
```

With the help of examples from internet, his version was adapted.

```
static MMCConfig mmccfg = {&SPID2, &ls_spi2cfg, &hs_spi2cfg};

uint8_t mmc_init()
{ …
palSetPadMode(GPIOB, 12, PAL_MODE_OUTPUT_PUSHPULL);
palSetPad(GPIOB, 12);
mmcObjectInit(&MMCD1);
mmcStart(&MMCD1, &mmccfg);
…
TCHAR *Path ="0:/";
result = f_mount(&MMC_FS, Path, 0);
…
}
```

Among the others small errors, there was an inversion between the right and the left button in the thread input_button_thread in the menu file.

Concerning the radio communication, one of the function `chIOGet((BaseChannel *) &SD2)`, that does not exist, was replaced by `sdGet(&SD2)` which is the right equivalent of gets().

The part to read the SD card was not working because of a wrong path indication.

Wrong path:

```
current_path[0] = '0';
current_path[1] = ':';
current_path[2] = 0;
```

Correct path:

```
current_path[0] = '0';
current_path[1] = ':';
current_path[2] = '/';
```

2.3.2. New functions

2.3.2.1. For the LCD

In order to have a clear and friendly menu displayed on the screen with arrows to indicate that the user can scroll down the menu, a new function has been created that allows to write symbols.

```
void lcd_put_sym(char sym)
```

2.3.2.2. Amphibot III

Previously, a code to control the Salamandra robotica has been made, however as the Amphibot III was available in the laboratory to test it, a new code has been implemented in order to control this robot. It has been inspired by the one made for the first remote by Gabriel Cuendet.

The amphibot.c file contains three functions. One initializes the robot, the other makes it move and the last one stops the robot.

2.3.2.3. Reading the values of the joysticks

In order to have a software where we can easily add a new code for another robot, the former functions for the joystick were recreated.

In the previous code, there was one thread that read the values of the joysticks and directly sent them to the register of the Salamandra robotica robot. This forced to write a new thread for each robot in the joystick file. Moreover, only one joystick was used.

Now, the thread in the joystick file only reads the values of the joysticks thanks to the ADC module of ChibiOS, and saves it in two variables. Then, it is the functions in the robot file, that take these values to move or stop the robot.

There is one thread that read the value of both of the joysticks. One part converts the y-axis values, corresponding to the speed, and the other converts the x-axis, corresponding to the direction. To control the robots these values have to be between -1 and 1. Thanks to ChibiOS one function can sample the data:

```
adcStartConversion (ADCDriver *adcp,const ADCConversionGroup *grpp, adcsamp
le_t *samples, size_t depth)
```

The STM32 has a 12-bit ADC which means it returns values between 0 and 2^{12}-1=4095. If the joysticks are at their rest position we want the values to be 0.

```
int16_t offset=2046;
joystick1_y = ((float)samples1[1] – (float)offset) / (float)offset ;
joystick2_x = ((float)samples2[0] - (float)offset) / (float)offset;
```

The move function of the robot, inspired from the function made by Gabriel, is called with two arguments corresponding to `joystick1_y and joystick2_x`.

```
void amphibot_move(float speed, float direction)
```

Then in the menu **Robot started**, the user can stop the robot by validating the "Stop the robot" submenu. The following functions are called:

```
void joystick_stop(void)
void amphibot_stop(void)
```

The first one stops the thread of joystick that do the ADC conversion and the second one stops the PID controllers of the robot.

Conclusion

The main functions of the remote control started by Matthieu are now working. It is possible to navigate through the menu displayed on the LCD. Tests have been made in the water to control the Amphibot III in speed and direction. Finally, the option to flash a firmware has been tested on the BoxyBot II, with a firmware that blinks the LED of the robot. However, time was missing to completely terminate the remote. First, all the files were not provided, then at the beginning it was not even sure that the hardware was fine and finally it took time to find why it was not working. This is the reason why the mechanical part has not been improved.

The implementation of the menu Battery needs to be done. For the moment only the Amphibot III robot can be control, but the code is made so that only the functions to move, stop and initialize, depending on each robot, have to be implemented. Finally, the improvement of the case still has to be done.

ANNEXE I - PCB 3D model

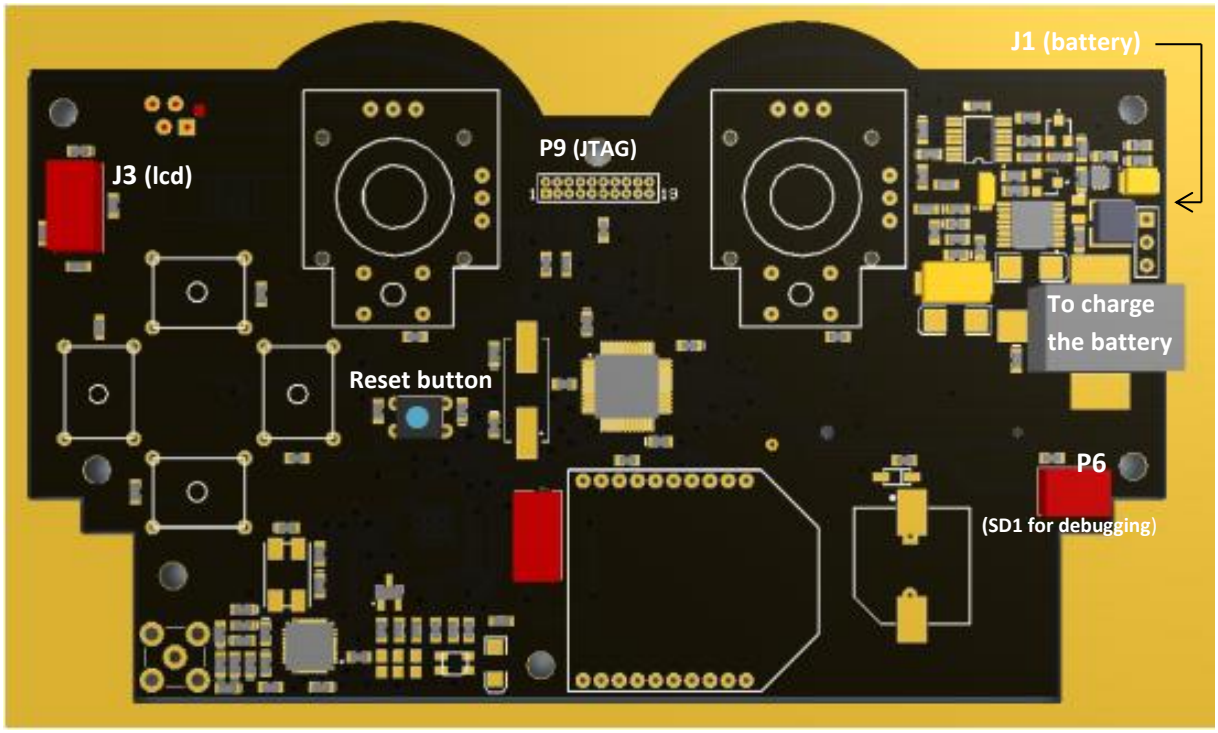

Figure 5 - Annoted 3D model of the PCB

ANNEXE II - Step up the remote

If you want to continue this project some steps have to be followed.

This project has been made on linux, with eclipse (not necessary) and the open source tool chain: arm-none-eabi.

First, you have to download this toolchain. During this project the version used was the 4.9 2015q3. https://launchpad.net/gcc-arm-embedded/+download

Then you need to download ChibiOS, http://www.chibios.org/dokuwiki/doku.php. As said previously, be careful if you want to use the latest version, it is possible that some changes need to be done on the code. However, it is very important to keep the file board.h because it contains the right configuration for the ports of the board made. It is located:

ChibiOS_3.0.2/os/hal/boards/OLIMEX_STM32_P103/board.h

You also need to download Openocd (Open On-Chip Debugger), it is an open-source software that interfaces with a hadware debugger's JTAG port. Matthieu used an OLIMEX ARM-USB-OCD-H probe but during the semester it just stopped working for no apparent reason. Thus then I used the ST-Link/v2, from STM32.

Once you downloaded Openocd, it will be located in usr/share. The version doesn't matter. Then you will need the file named openocd.cfg, located in the same directory that the makefile of the project. It contains the lines:

```
Source [find interface/stlink-v2.cfg]
Source [find target/stm32f1x_stlink.cfg]
```

If you change the probe, you will also have to adapt these two lines.

Then to flash the remote, you need to plug the ST-Link/v2, on the connector P9, between the joysticks. It is possible, like you don't have the administrator rights on the computer, that you cannot use the drivers of the probe. If it is the case, you have to create a file, in /etc/udev/rules.d, named stlink-v2.rules. In it, write:

```
SUBSYSTEM=="usb", ACTION=="add", ATTRS{idProduct}=="3744",
    ATTRS{idVendor}=="0483", MODE="664", GROUP="users"
```

Then open two terminals. In one, go to the directory where the project is, then write:

```
openocd -cf openocd.cfg
```

If it is well connected you should have:

```
Open On-Chip Debugger 0.7.0 (2013-10-22-08:31)
Licensed under GNU GPL v2
For bug reports, read
```

```
    http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : This adapter doesn't support configurable speed
Info : STLINK v2 JTAG v23 API v2 SWIM v4 VID 0x0483 PID 0x3748
Info : Target voltage: 3.235672
Info : stm32f1x.cpu: hardware has 6 breakpoints, 4 watchpoints
```

In the other terminal write:

```
telnet localhost 4444
halt
flash write_image erase build/ch.elf
```

Don’t forget to press the reset button once the code has been flashed.

As said before, Matthieu added an additional serial port (SD1) on the board, in order to be able to debug. This way, thanks to the chprintf function of chibiOS you can see on a screen what you want. Example of use of the function:

```
sdStart(&SD1, NULL);
chprintf((BaseSequentialStream*)&SD1, “Hello world\r\n”);
```

To use it, you need a particular cable that you have to plug into the connector P6, located next to the buzzer. You also need to download the application: Screen. Then open a terminal and write:

```
screen /dev/ttyS0 38400
```

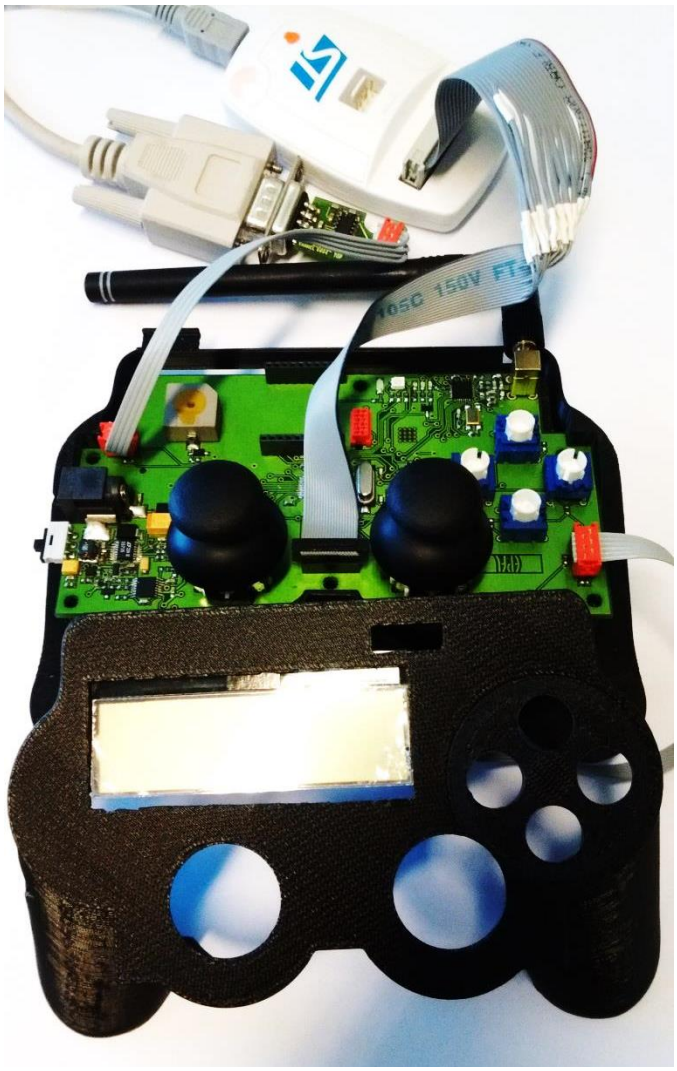

Figure 6 - Configuration of the connections