Remote control for CPG-based robots

Gabriel Cuendet

Semester project at the Biorob laboratory
Ecole polytechnique fédérale de Lausanne

Professor: Auke Jan Ijspeert
Assistant: Alessandro Crespi

Spring semester 2010

Abstract

This 3^{rd} year semester project's aims at the realization of an autonomous remote control for the CPG-based Biorob's robots. This report presents the work that was done from the beginning of the project to the final realization of the device. The first step was the realization of a first prototype (the development version) on which the software could be developed. Then a second version was designed in order to fit into a case. Some considerations were also made about the ergonomics of the device.

Contents

Chapter 1

Introduction

1.1 Current way to remote control the robots at Biorob

The communication between the remote controller and the robot is based upon a radio protocol using the ISM frequencies band. The radio interface on the robot's side will not be detailed in this project. On the controller's side, the radio interface is currently connected to a computer via USB. The user is then controlling the robot with a wireless gamepad. A software on the computer interfaces the signals from the gamepad to the radio interface which sends those signals to the robot. Thus the computer is used only as an interface between the gamepad and the radio interface but still remains essential because the protocol used by the gamepad is proprietary. The consequence is that it is very difficult to interface the gamepad directly with the radio interface, without needing the computer.

The need for a computer is not a problem when controlling the robots inside the laboratory. On the opposite, when outside the laboratory, for example in an exhibition for a demo or in the nature to make a movie, the computer becomes annoying.

Therefore the goal of this project is to make an autonomous device that permits to wirelessly remote control any of the Biorob's CPG-based robots that use the ISM frequencies band. The next page shows the task description including the initial project's description.

Remote control for CPG-based robots

Semester project, task description

Gabriel Cuendet, March 19, 2010

The goal of this project is to develop a wireless remote control that communicates with a robot, which is controlled by a CPG . The electronics for the RF part aren't part of that project, since they are already designed. The remote control is able to configure a small number of locomotion parameters on the robot. It allows a user to interactively remote control the robot without needing a PC.

Task-list :

1. Review of the existing remote control using a PC and a radio interface. This radio interface is then included in the current project.
2. Definition of the components needed to achieve interaction between the user and the robot.
3. The PIC18F2580 development kit is available for test purpose.
4. Design of the hardware (PCB).
5. Design of the software on the micro-controller.
6. Design of the housing and mechanical integration of the hardware.

Deliverables:

1. A final report.
2. An entry on the BIRG site, which summarizes the work on the project.
3. A live demo of the remote control during final presentation.

Chapter 2

Hardware design

For a better understanding, the device can be divided into three distinct parts:

- The radio interface
- The user interface
- The battery management and voltage supply

Even if the three parts are interconnected, they are analyzed separately in this report. Indeed, they were designed that way. The complete schematic of the three parts can be found at the end of this report (cf. Appendix A). Although the design of the radio interface is not strictly speaking part of this project, a small section will be dedicated to its functioning.

During the project, a total of two versions of printed circuit board of the hardware have been realized:

- the first prototype, called the *development version*, on which the software part of the project has been developed and
- the second prototype, called the *final version*, which had to fit into a case and was made more user-friendly.

2.1 The radio interface

The radio interface is composed of a microcontroller (the Microchip **PIC16LF876A**), a 868 MHz radio transceiver (the Nordic **nRF905**), an impedance matching circuit and an antenna.

The microcontroller communicates with the user interface (which previously was the computer) using an asynchronous serial transmission. The communication protocol which is implemented on that microcontroller was designed by Alessandro

Crespi and the relevant aspects of it will be seen in the « software » section of this report.

Without going too much into details, this radio interface can thus be considered as a black box with two wires for the asynchonous serial communication input and output on one side and an antenna for the 868 MHz radio communication input and output on the other side.

2.2 The user interface

As it is said in the task description (cf. 1.2), the device must be able to configure a small number of locomotion parameters on the robot and to permit *interaction* between the user and the robot.

2.2.1 Definition of the needed components

First, in order to allow the user to control the locomotion parameters of the robot, which are mainly the speed and the direction, a two-axis joystick is needed. It has to be a potentiometer joystick and not a switch joystick in order to provide a better resolution in the movements.

Furthermore, four push buttons are used in a menu navigation system to control every other parameters: an ENTER button, a CANCEL button, a PLUS (+) button and a MINUS (-) button. To allow further development of the device, a fifth button is added.

Finally, in order to achieve interaction, a visual feedback is provided to the user through LEDs and a LCD screen. As the LCD is mainly used in the menu navigation system, a simple 2x16 characters, alphanumeric LCD is sufficient. In this case, a I^2C interfaced LCD (the Batron's **BTHQ21605V-FSRE-I2C-COG**) has been chosen in order to minimize the number of pins used for the display on the microcontroller. It also minimizes the number of tracks on the printed circuit. Moreover, the LCD voltage supply is only 3.3 V [1].

The LEDs can be used instead of the LCD to display binary values or to indicate an error. In the device, two green LEDs are used to display binary values (for example if the robot is moving in a walking or swimming mode) and a red one to indicate an error. In the development version, a LED indicated if the device was turned ON or OFF. As the device is supplied by a battery, this LED has been removed in the final version; it was only discharging the battery faster by pulling a

[1] see the datasheet of the BTHQ21605V-FSRE-I2C-COG pp.7-8 , for more details

constant current out of it without providing any consistent information to the user.

2.2.2 Functioning of the user interface

The link between the above elements and the radio interface is done by a microcontroller, a Microchip **PIC18F2580**. The choice of this microcontroller was dictated by the habits of the staff of Biorob and the fact that this particular microcontroller and development tools for it were available at Biorob.

Figure 2.1 shows the user interface components, linked to the microcontroller. For a complete schematic, see the Schematics section in the appendix (cf. Appendix A).

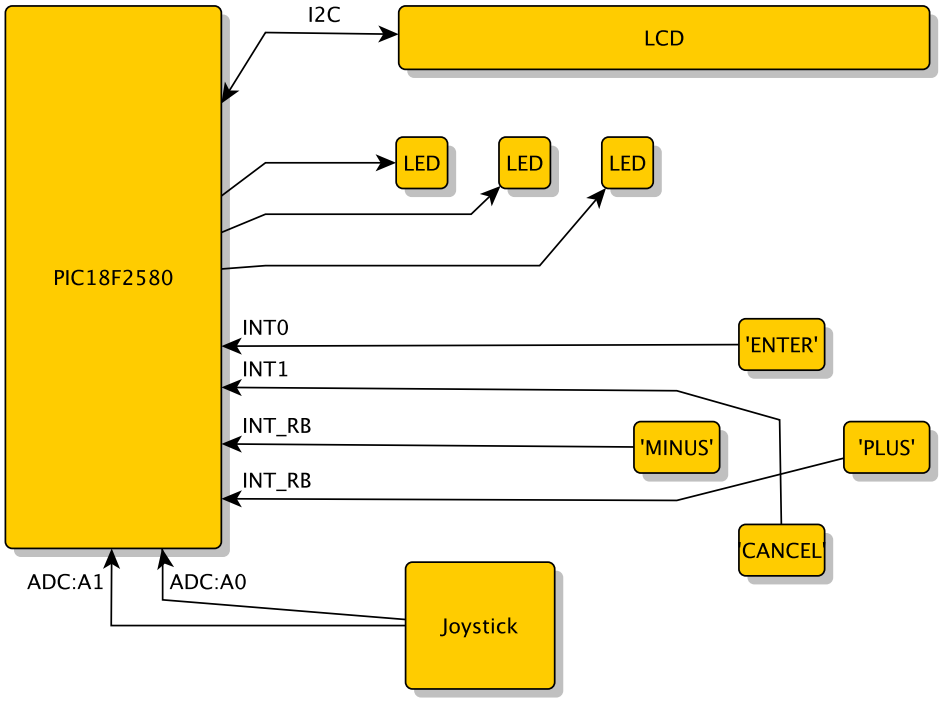


Figure 2.1: User interface components

Two of the eight analog-to-digital 10-bits converter of the microcontroller are used to convert the voltages on the two joystick potentiometers (each corresponding to an axis) to an 8-bits number. In spite of the 10-bits conversion capability of the converter, 8-bits are sufficient to encode the position of the joystick with a good resolution. Consequently, the 8-bits mode is used instead of the 10-bits one.

The ENTER and CANCEL buttons are connected to input 0 and input 1 of port B, which correspond to the external interrupts 0 and 1 (INT0 and INT1 Pin interrupts).

The additional button is connected to input 2 of port B (INT2 Pin interrupt). It is not represented in figure 2.1 as this button is not used. Those three buttons have independent interrupts, in opposition to the PLUS (+) and MINUS (-) buttons which are connected to input 4 and input 5 of port B. Those inputs are triggering the same interrupt, which is the PortB interrupt-on-change (INTRB). In spite of that fact, the buttons are easily differentiated by a simple test on these inputs at the beginning of the interrupt routine.

As the LCD uses a I^2C communication protocol, only two wires are needed to interface it with the microcontroller. Pins 3 and 4 of port C are respectively the I^2C clock and data input/output.

The LEDs are connected between outputs 0, 1 and 2 of port C and the ground through 150 Ω resistors. The forward current is then approximately [2]:

$$I_f = \frac{U_{supply} - U_f}{R} = \frac{3.3\text{V} - 1.9\text{V}}{150\Omega} = 9.3\text{mA} \tag{2.1}$$

The maximum current sourced by any I/O pins is 25 mA while the maximum current sourced by the port is 200 mA. The microcontroller is then capable of sourcing a sufficient current for those three LEDs. A lower value for the resistor could have been chosen in order to increase the current and the intensity of light, but then the battery would not have lasted as long.

Finally, the microcontroller is connected to the radio interface via a serial link. Pins 6 and 7 of port C are connected to the TX (transmit) and RX (receive) tracks. Of course, on the side of the radio interface's microcontroller (the PIC16LF876A) the tracks are cross-connected, which means that the TX track is connected to the RX pin and vice versa.

2.3 The battery management and voltage supply

As the device is powered by a rechargeable Li-Ion battery, this one needs to be protected and charged. Moreover, a monitoring function is very useful to know the charge level of the battery. The voltage from the battery is included between 4.2 V and about 3 V depending on its charge. Due to this variation, a DC/DC converter is needed to provide the device a constant voltage.

The above functions are realized using the same integrated circuits as in the salamander 3. Indeed, the schematics have been adapted from the schematics *SAL3BODYPWR* drawn by Alessandro Crespi.

[2] The forward voltage U_f is given in the datasheet of the LED in function of the forward current

2.3.1 Battery management and charge

The **DS2764** integrated circuit from Dallas Semiconductor is a high-precision Lithium ion battery monitor including a data-acquisition, information-storage, and safety-protection device. The circuit permits to disconnect the battery in case of overcharge (overvoltage), overdischarge (undervoltage), and excessive charge and discharge currents (overcurrent, short circuit) to avoid irreparable damages. This circuit also allows to monitor the temperature, voltage and current through a I^2C communication protocol. Moreover, it has a 40 bytes EEPROM memory that will not be used in this project but could be used in further developments.

The battery charging is managed by the **LT1571-5** integrated circuit from Linear Technology. The charge voltage can be include between 8.2 V and 26 V. At Biorob, the chargers are providing a 24 V constant voltage. The circuit also provide a flag output which indicate the end of the charge. This one is not used in the final version of the remote control because a software implementation of that functionality has been preferred. The advantage of the software implementation versus the LED is mainly at the mechanical integration level. Indeed, a charge-LED would have meant one more hole near the power jack connector, which would have been problematic because of the lack of space on this side of the case (cf. chapter 4).

2.3.2 Voltage supply to the device

The option to supply the whole device with a single 3.3 V voltage has been taken. It would have been perfectly possible to supply two parts of the circuit with two different voltages, but obviously, it requires less components to generate only one constant supply voltage. As the radio interface circuit (nRF905) needs a supply of 3.3V, this voltage is inevitably necessary.

Most of the LCD displays need a 5 V supply, but the chosen one only needs a 3.3 V supply. This is one of the reasons why it was chosen, beside the I^2C interface.

The only problematic circuit is the user interface's microcontroller, the PIC18F2580. According to the datasheet[3], this version of the microcontroller only accepts a supply voltage between 4.2V and 5.5V. However, the low-voltage part PIC18**L**F2580 functions over an extended V_{DD} range of 2.0V to 5.5V but with reduced clock speed. In both versions (development and final version) of the remote control, a PIC18F2580 is used. Indeed, some tests had already been made at Biorob and the only problem with a supply voltage of 3.3 V on that microcontroller was a failure with the CAN bus at a temperature exceeding 35° C. As the CAN bus is not used in this project,

[3]see the datasheet of the PIC18F2580, pp.422-423

this is not a limitation.

Taking into account all those considerations, a single DC/DC converter is needed in the device. The **LTC3240-3.3** from Linear Technologies has been used. This is a step-up/step-down charge pump DC/DC converter that produces a fixed regulated output voltage of 3.3V over a wide input voltage range (1.8V to 5.5V) including the voltage range covered by the battery at the different charge levels.

2.4 Key points about the PCB

The most sensitive routing part is the impedance matching circuit. The tracks must be as short as possible all the way between the ant1/ant2 outputs of the nRF905 integrated circuit and the antenna. Moreover, the tracks must be balanced.

For most integrated circuits, good routing advices can be found in the corresponding datasheets. For example, the routing advices about the charger[4] (LT1571-5) have been followed.

[4]see the datasheet of the LT1571-5 p.14

Chapter 3

Software

During the project, only the software on the microcontroller PIC18F2580 has been developed. Indeed, the software on the radio interface's microcontroller was already developed and thus will not be part of this report. The software has been developed on the development version of the hardware.

This chapter aims to present the global functioning of the software and not to describe every single functions in details. The reader should read the code, which is commented, in order to get those details.

3.1 Basics

3.1.1 Getters and setters

The way to control the robots consists only in writing into or reading specified registers of the robot. As a consequence, the most needed software functions are the *setter* functions and the *getter* functions.

The remote control always remains the master and the robot the slave. The robot cannot initiate the transmission. Thus, the *getter* functions first have to *ask* the robot to send the value of a register. There is no function receiving data automatically from the robot. That enables both the *getters* and the *setters* to share the same structure. For both functions, a one dimension array **req** is defined. Its structure is shown in figure 3.1.

The first two elements of that array are then sent to the radio interface via the serial link. The radio interface microcontroller answers with a confirmation byte. This byte has either the value '6' in case of success, which value corresponds to the ACK (Acknowledgment) character in the ASCII table or the value '13' (NAK (Negative Acknowledgement) character) in case of failure.

req[0]	op	op	op	op	op	op	addr	addr
req[1]	addr	addr	addr	addr	addr	addr	addr	addr
req[2-n]	data	data	data	data	data	data	data	data

Figure 3.1: Structure of the array **req** with 'op' refering to operation and 'addr' to address.

Depending on the function (either a *getter* or a *setter*), the data are then sent from the radio interface via serial link and written in the **req[2]** and following or read from the **req[2]** and following and sent to the radio interface via RS-232. The number of lines of **req** depends on the length of the register that is being accessed. That length varies from one byte to 29 bytes making the number of lines of **req** between 3 and 32. Several *getters* and *setters* of various length are then implemented, but each one is based upon the same basic function:

```
int1 reg_op(int8 op, int16 addr, int8* data, int len);
```

All the access to the robot will be provided by the specific-length *getters* and *setters*.

3.1.2 Interfacing the buttons

Every time a button is pressed, the corresponding interrupt is trigered. The figure 3.2 shows those interrupts names.

Button	*Pin*	*Interrupt*
ENTER	PORT B0	INT0
CANCEL	PORT B1	INT1
(SUPP	PORT B2	INT2)
PLUS	PORT B4	INT_RB
MINUS	PORT B5	INT_RB

Figure 3.2: The buttons and their corresponding interrupts

As the buttons PLUS and MINUS trigger the same interrupt, inputs 4 and 5 of port B are read at the beginning of the routine of interrupt INT_RB in order to decide which buttons trigger interrupt up.

This detail taking apart, all the interrupts then works the same way:

1. A *do-while* loop increments a variable from 0 to 50 each millisecond as long as the button state does not change (no bouncing).

2. If the variable is greater than 45, which means the button has been pressed for 45 millisecond without bouncing, the function

```
void press_button(char button);
```

is called. Otherwise, the function is not called. As the button is bouncing, it is very likely that the interrupt is going to be set up again at the next bounce.

The function *press_button* implements the state machine as described in section 3.3.

3.2 The state machine based model

3.2.1 Definition of the different states

In order to facilitate the navigation into the different functions, six states have been defined:

1. RC Autonomous: When no robot is controlled, this state allows the user to access the menu by pressing the ENTER button in order to select one robot or to access the remote control's functions like the display of the battery voltage.

   In this state, the charge of the remote control is displayed on a seven segments bar graph. The values of the segments are the following: 3.1 V / 3.25 V / 3.4 V / 3.55 V / 3.7 V / 3.85 V / 4 V. If the voltage is under 3 V, a text message is displayed : 'EMPTY!'.

2. Menu: In this state, the menu is displayed. The working of the menu is described in section 3.3.

3. Stopped: When a communication with a robot has been set up, this state allows the user to access the menu but not to control the robot in speed and direction (the INT_RTCC interrupt is disabled).

   The LCD displays the text "Ready" on first line and "ENTER FOR MENU" on second line. Entering the menu enables the user to start the robot and thus to control it in speed and direction.

4. In charge: When the robot is charging, the corresponding state is "In charge". On the older version of the salamander the charge can be interrupted by pressing the CANCEL button. The menu can still be accessed, but the robot cannot be started and thus cannot be controlled in speed and direction.

5. Started: In this state, the INT_RTCC interrupt is enabled. The joystick controls the speed and direction of the robot (see section 3.4 for more details). The

menu cannot be displayed and the robot is stopped by pressing the CANCEL button. The PLUS button is enabling the turbo mode while the MINUS button is disabling that turbo. A single pressure on the ENTER button switches the locomotion between walking and swimming.

6. Error: "ERROR !" is displayed on the first line of the LCD and a small description of the error on the second line. Furthermore, the red error LED is turned on.

The figure 3.3 shows the state machine.

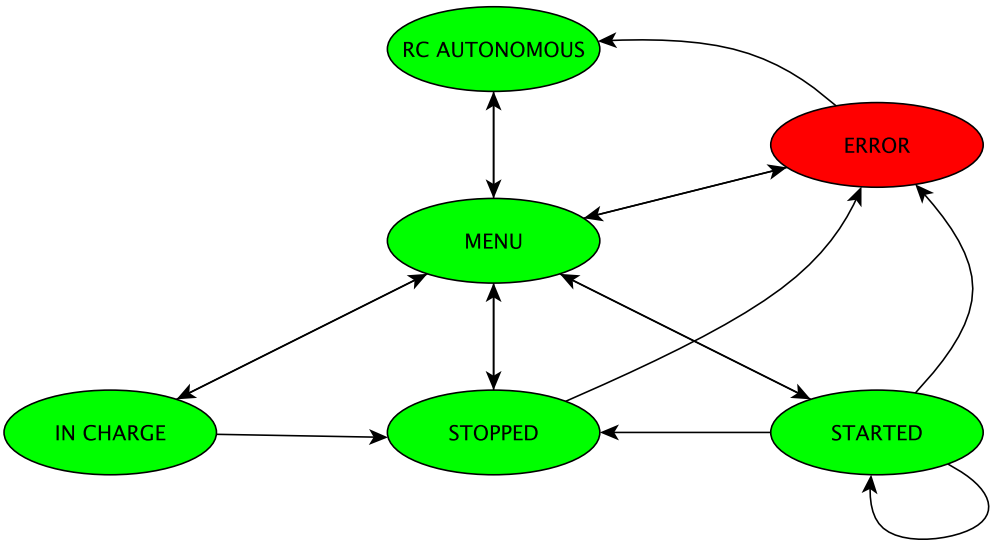


Figure 3.3: State machine

3.2.2 Operations of the state machine

Two global variables (*current_state* and *previous_state*) are defined. They are of the type *state*.

typedef enum{STARTED, IN_CHARGE, STOPPED, MENU, RC_AUTONOMOUS, ERROR=9} state;

The use of two variables enables to remember a previous state. This is useful with the MENU and ERROR modes where a return to the previous state is needed.

Basically, the transition between the states are made via the MENU state. The states IN CHARGE and STARTED can be accessed only if the previous state (stored in

the *previous_state* variable) is STOPPED. Indeed, the access to the STOPPED state depends on the valid connexion to a robot and the states IN CHARGE and STARTED need a valid connexion to this robot.

When the current state is STOPPED, the function *start robot* of the sub menu ROBOT initializes the robot, enables the INT_RTCC interrupt and set the current state to STARTED. It allows the user to control the robot in speed and direction.

Once in the states IN CHARGE or STARTED, a simple pressure applied on the CANCEL button make the current state STOPPED.

At any time, in case of error, for example when selecting a function in the menu for which a robot is needed but that is not responding, the current state switches to the ERROR state. The error description remains displayed until the user press any of the buttons. The previous state is then reloaded.

3.2.3 Infinite loop in the *main*

In the *main*, after the initialization of the device, the program enters a **while**(1) loop. Inside the loop, a test is made on the current state. Depending on the current state, the corresponding informations are displayed on the LCD. But if the current state has not changed and if the display does not need to be actualized, it is annoying to continuously update the display. Indeed, it creates a blinking effect. To avoid that disturbing effect on the LCD, the previous state is compared to the current state at the beginning of the **while**(1) loop and if they are equals, nothing is done.

As the MENU and ERROR states need to have access to the previous state, the variables *current_state* and *previous_state* will never be equal while in those states. It is not a problem for the ERROR state because the display is updated at the call of the function

```
void err(char* description)
```

and not during the **while**(1) loop. On the contrary, while in the MENU state, the function

```
void display_menus()
```

is called during the **while**(1) loop. This leads to a blinking effect while in the MENU state. To avoid this, the comparison of the current state and previous state has been reinforced with a more complete test condition:

(current state $\neq$ *previous state)* ***XOR*** $\lceil$*(current state* = MENU*)* ***AND*** *(display didn't change)*$\rfloor$

3.3 The menu like user interface's implementation

The menus, displayed on the LCD and whose give access to the different functions of the remote control, are two levels menus. For a memory saving purpose, the labels of each top menu and each function are stored in a single dimensional array of characters strings:

```
char menu_labels[13][16] =
{"INFOS RC","Batt. voltage", ..., "Stop charging"};
```

Indeed, none of the menu have the same length and it would have been a waste to allocate memory according to the longest menu (the one having the most functions) in a two-dimensional array. As a consequence, two variables are then needed to store the current first level and second level during the navigation. Those variables are:

```
typedef enum  {INFOS_RC_MAIN=0,
               SELECT_ROBOT_MAIN,
               ROBOT_MAIN} main_menu;

main_menu    current_main_menu = 0;
```

and

```
typedef enum
        {NO_SUB=0, BATT_VOLTAGE_SUB=1, VERSION_RADIO_SUB,
        FAVORITES_SUB=1, ENTER_CHANNEL_SUB, SCAN_CHANNELS_SUB,
        START_SUB=1, STOP_SUB, START_CHARGING_SUB,
                STOP_CHARGING_SUB} sub_menu;

sub_menu     current_sub_menu = NO_SUB;
```

The corresponding functions are stored in an array of pointers on functions taking no arguments and returning nothing.

```
typedef void (*tab_fct)(void);

tab_fct menu_fct[10]={show_batt_voltage, ...,
                                        menu_stop_charging};
```

In order to be able to display the label (or to select the function) corresponding to the current main menu index and/or sub menu index, the number of menu and the length of each menu have to be store. This is done in the variable:

```
int8 menu_lengths[4]={3, 2, 3, 4};
```

Where the first element is the number of main menu and the following elements are the length of each sub menu.

Finally, the function **void** *display_ menus()* is called each time a button has been pressed. This is indicated by the value of the variable:

```
int1      display_changed = 0;
```

This variable is set to one each time the function **void** *press_ button(char button)* is called and set to zero before each new display.

3.4 Controlling the robot in speed and direction

Each of the two axis of the joystick corresponds to a parameter, either the speed or the direction. When the user is controlling the robot, the timer-based interrupt INT_RTCC is enabled. At each interrupt, an *update* function updates the content of defined registers in the robot.

3.4.1 Reading the position of the joystick

The period of the interrupts has been chosen in order to have a good compromise between a slow, non responsive system (if $T \gg 0.2$s) and a too demanding period for the system ($T \ll 50\mu$s [1]). Indeed, during the interrupt, $2 \cdot 10\,\mu$s are necessary to switch the input channel plus the time of conversion which also has to be taken into account.

Afterwards, it appeared that a very similar value had been chosen in the previous software[2] which was used with the gamepad and the computer (see Current way to remote control the robots at Biorob, section 1.1). The chosen period is given by equation (3.1) for a prescaler equal to 4.

$$T_{interrupt} = \frac{2^{bits}}{\frac{f_{counter}}{prescaler}} = \frac{2^{16} \cdot 4}{2500000} \approx 104 \text{ ms} \tag{3.1}$$

At each interrupt, the voltages on the x axis (corresponding to the speed) and the y axis (corresponding to the direction) are converted to the 8-bits numbers a and b.

The *update* function (here the *update* function of the salamander[3]) is then called

[1] see the datasheet of the PIC18F2580, p.258 for the detailed calcul of the minimum required acquisition time, which is if the order of 10 - 15 µs maximum

[2] cf. file "joyctrl.cpp", part of the software developped by Alessandro Crespi.

[3] corresponding functions exist for the amphibot II, the amphibot III and for the amphibot with Mathieu Porez software

with two arguments:

```
int1 update_salamander(float spd, float dir)
```

As the *update* functions take two floating point numbers between (-1) and 1 in argument, those are calculated from the converted values a and b of the ADC:

$$spd = \frac{a - \frac{2^8-1}{2}}{\frac{2^8-1}{2}} = \frac{a - 127.5}{127.5} \tag{3.2}$$

$$dir = \frac{b - \frac{2^8-1}{2}}{\frac{2^8-1}{2}} = \frac{b - 127.5}{127.5} \tag{3.3}$$

Moreover, the speed and direction values are displayed on the LCD.

3.4.2 Updating the locomotion parameters on the robot

The *update* functions update the values of the registers of the corresponding robot in function of:

- the position of the joystick (given by the arguments *spd* and *dir*),
- the current locomotion mode stored in the global variable

  ```
  locomotion_mode current_locomotion_mode;
  ```

  which is of the type:

  ```
  typedef enum{WALKING, SWIMMING} locomotion_mode;
  ```

  (can be either *walking* or *swimming* for the salamander and only *swimming* for the amphibot) and
- the turbo option (turbo ON or OFF) which value is stored in the global variable

  ```
  int1 turbo = 0;
  ```

In order to control the different types of CPG-based robots at Biorob, which are mainly the salamanders and the amphibots, several *update* functions have been implemented. Indeed, each different type of CPG-based robot has its own registers addresses and its own locomotion parameters set. Moreover, the locomotion parameters set is different from a version to another even for the same type of robot. In each *update* function the parameters set is updated in function of the version of the current robot (which is stored in the *current_robot* global variable).

The values of the different registers and the locomotion parameters set have been copied from the code of the previous software which was used with the gamepad and the computer[4].

[4]cf. file "robots.h" and "robots.cpp", part of the software developed by Alessandro Crespi.

Chapter 4

Housing and mechanical integration

From the very beginning of the project, the goal was to come to a device integrated into a case. An effort has been made in order to find an ergonomic case and to keep the aesthetic side on mind. This chapter focus on the choices that have been made and not on all the mechanical aspects of the work.

4.1 Choice of the case

The key point in the choice of the case has been the size. Indeed, as the device was intended to be hold in the hand, the size of the housing had to be reasonable. On the other hand, some elements were of such dimensions that they would not have fitted into a small case. For example, the height of the joystick or the width of the LCD were limiting factors.

During the search for the right case, two main options appeared:

1. A bigger case in which the joystick could fit entirely but less ergonomic because of its size and shape. In particular the height of the case would not have permitted to hold it in one hand comfortably.

2. A specific case designed for hand-held comfort, with a reduced height but in which the joystick would not have fit entirely.

Figure 4.1 shows an example of a case of each category. The second option has been preferred, giving priority to the ergonomic. Moreover, the aesthetic point is not forgotten. Indeed, a case specifically designed for hand-held comfort is much more aesthetic than a big rectangular box in spite of the joystick sticking out of the case.

From a mechanical point of view, the chosen option is a bit more complex because of the shape of the joystick hole which is not just a circular hole. This leads to an

Figure 4.1: Two types of case: a) option 1; b) option 2

important comment: the device is certainly less protected from the dust or even the water splashing when choosing the second option. Even if the device does not have to be waterproof, an improvement of that aspect could be made in further development.

The case is in molded ABS plastic with a black soft plastic grip on the sides.

4.2 Ergonomics of the user interface

Figure 4.2 shows the front panel (top) of the remote control.

Figure 4.2: Front panel view (without antenna)

The orientation of the case was dictated by the fact that the antenna is an external one. As it is sticking out of the box from the end panel, the end panel had to be pointing forwards and not sideways which would have been the case with an horizontal orientation. The horizontal orientation would have permit to hold the case with both hands: controlling the joystick with one and pushing the buttons with the other one. The use of the device would then have been very similar to the use of the gamepad.

Conversely, the vertical orientation enables the user to hold the device with only one hand. A small experiment among the staff of Biorob shows that the right-hander first take the device with the left hand in order to have access to the joystick with the right hand. The four buttons, slightly on the right side are then also accessible with the right hand. While not controlling the robot in speed and direction, the device can also be held with the right hand. In this case, the buttons are accessible with the same hand. The use is then similar to the use of a television remote control.

The layout of the buttons is strongly inspired from the gamepad that is currently used at Biorob. Indeed, the gamepad is especially designed to be ergonomic.

4.2.1 End panel

Figure 4.3 shows the end panel of the remote control.

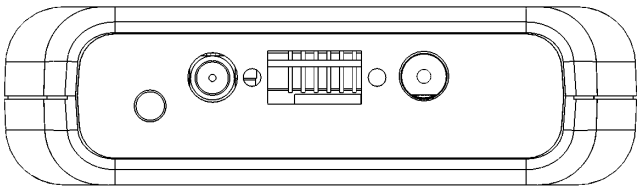


Figure 4.3: End panel view (without antenna)

From the left to the right, there is a small reset push button, the antenna connector, the main power switch and the jack connector for the charger.

The reset button[1] is intentionally not too easily accessible as its size is reduced and its position is on the side of the top panel.

[1]Note that this button resets only the user interface microcontroller (PIC18F2580) and not both microcontrollers.

4.3 Components and integration

4.3.1 Buttons

The buttons that have been used are the Multimec **3FTL6**. They are composed of a through hole mounting body covered by a cap. The important point is that this cap can be chosen in various height : 16 mm, 19 mm or 22.5 mm. In the present case, 19 mm height buttons perfectly fit for the application, sticking out from the case of about 3 mm.

4.3.2 LCD

The total height between the top surface of the PCB and the inner top side of the case is 8.9 mm with a 1.6 mm thick PCB. Thus, the LCD could not be fixed on the PCB and at the same time be near the surface of the inner top side of the case which is what is wanted.

The selected solution is to fix the LCD on another PCB which is itself fixed on the top of the case. The link between the main PCB and the LCD is then made via a flat cable.

Figure 4.4 shows the side view of the inside of the case. The second PCB is in green and is screwed in the dark grey boss.

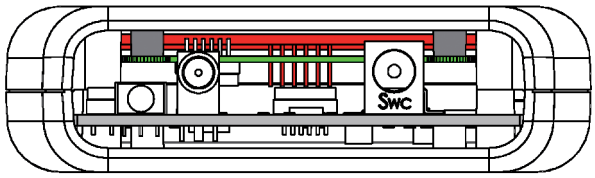


Figure 4.4: Elevation view of the inside of the case. The main PCB is in light grey, The LCD is in red and the second PCB in green.

This solution enables the battery to be fixed between the two PCBs by providing just the sufficient height. In this version, the battery is fixed with double side scotch-tape. An alternative idea is to integrate the fixating system on one of the PCB. This system could be as simple as four pins surrounding the battery, one on each side, making a cage. By assembling the two parts, the battery is then hold in position.

Figure 4.5 shows the shape of that second PCB and the position of the LEDs and buttons.

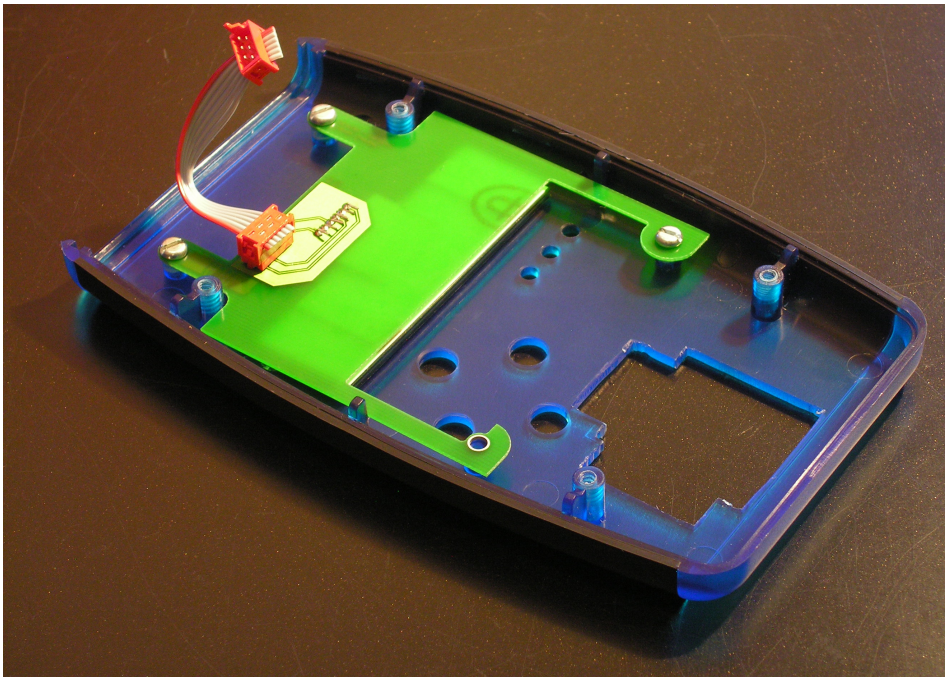


Figure 4.5: Second PCB on the inside of the top panel

4.4 Mechanical result

Figure 4.6 shows the disassembled device.

Figure 4.6: Final version of the remote control

Chapter 5

Conclusion

The designed device satisfies the main objective of the project, which was to have an autonomous device. Moreover, the designed remote control seems to be practical and user-friendly, but this has to be verified during the effective use of it. The external size of the device is not much bigger than the size of the previously used gamepad.

Keeping in mind that the final version of the device as presented in this report is only the second prototype, improvements can still be made.

5.1 Recommendations for improvements and further developments

From a mechanical point of view, the drawings could be improved. Indeed, as this is only the second version, a big margin has been taken on every hole. By reducing this margin, the device could be better protected against dust and even water splashing. Nevertheless, the case cannot be made waterproof as the joystick is sticking out.

An interesting development could be the integration of the antenna inside the case. This would considerably reduce the external size of the device. Moreover, it could enable the user to hold the device horizontally, providing new potential uses.

Most of the improvements that could be done are in the software part. For example, the voltage indicator during the RC_AUTONOMOUS state, could better fit the discharge curve of the battery, which is non-linear. A function could also provide an access to the charge level of the battery, during its charge. Indeed, the function

```
float battery_current();
```

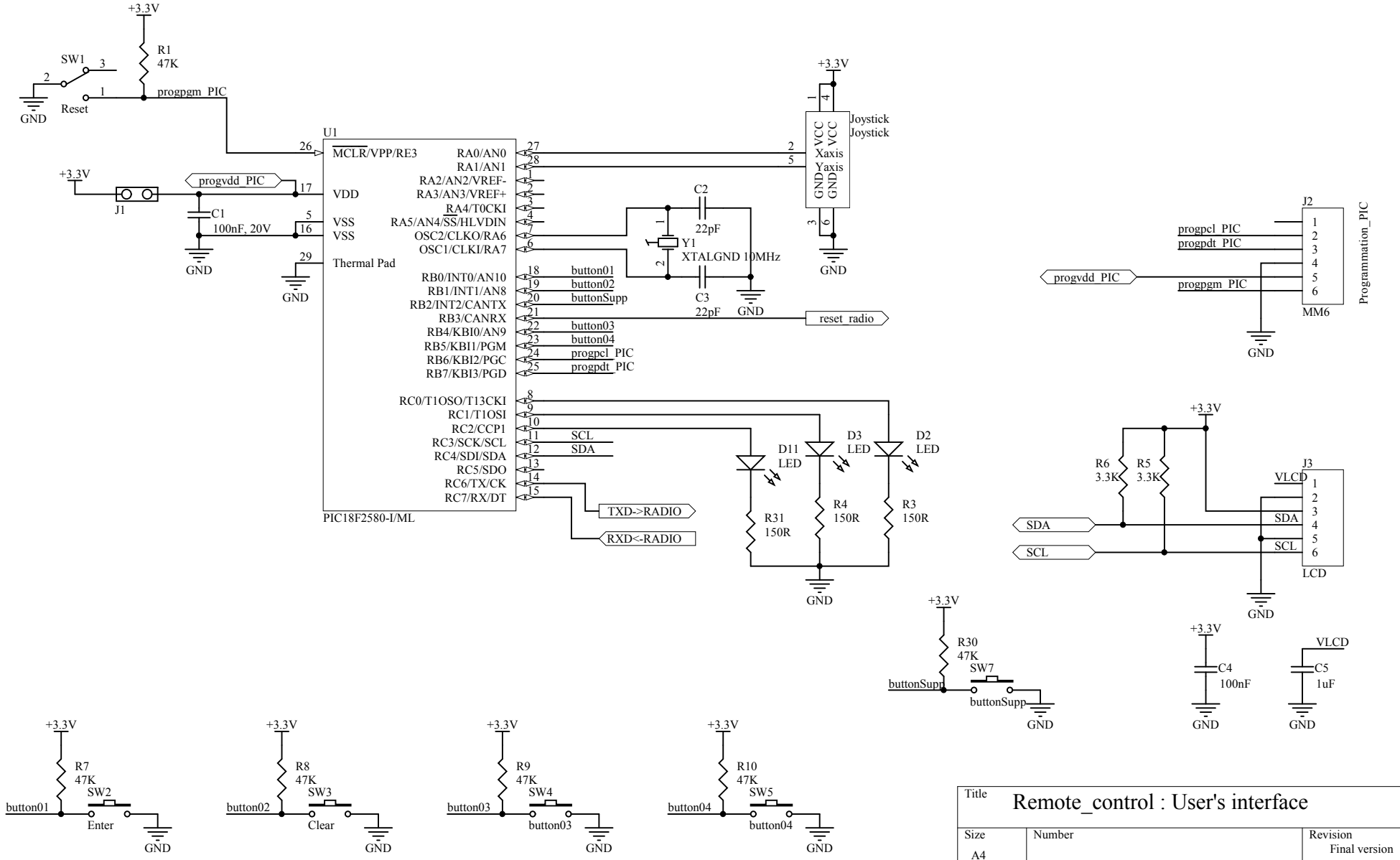
can be used in order to measure the current flowing into the battery (while in charge) or out of the battery. This provides access to the level of charge, as the battery is

charging under a constant voltage during the second part of the charging cycle[1].

The management of the different robots could also be improved, making easier to add a robot with its own parameters set without having to write several functions (the *init* function, the *update* function and possibly other specific functions).

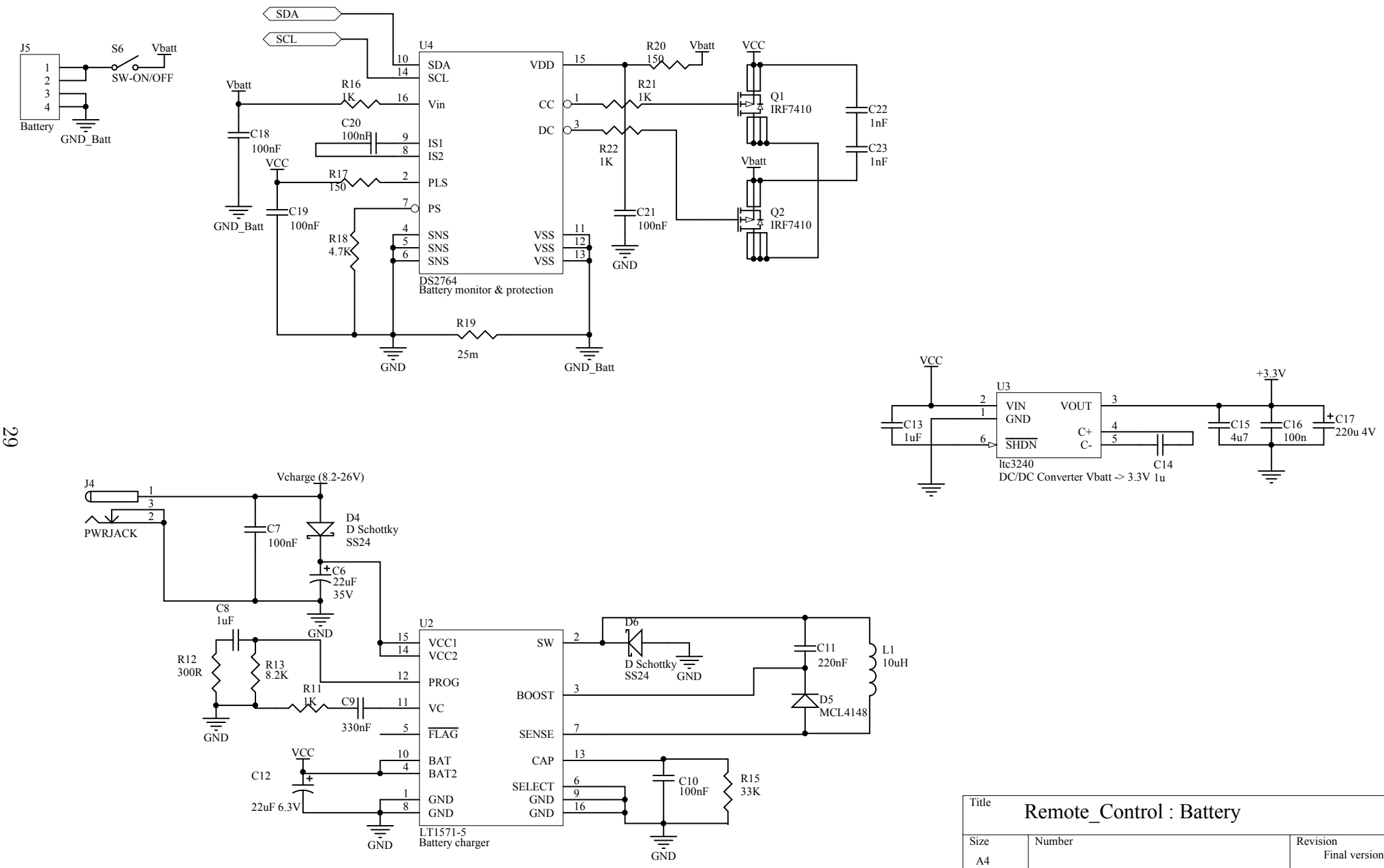
A further software development could be to enable the user to choose between different parameters sets for each robot. The locomotion could then be adapted to different conditions, like for example the type of the ground.

[1]In the first part of the charging cycle, the current flowing into the battery is constant ($I = 600\text{mA}$). In the second part, the voltage remains constant while the current is decreasing non linearly.



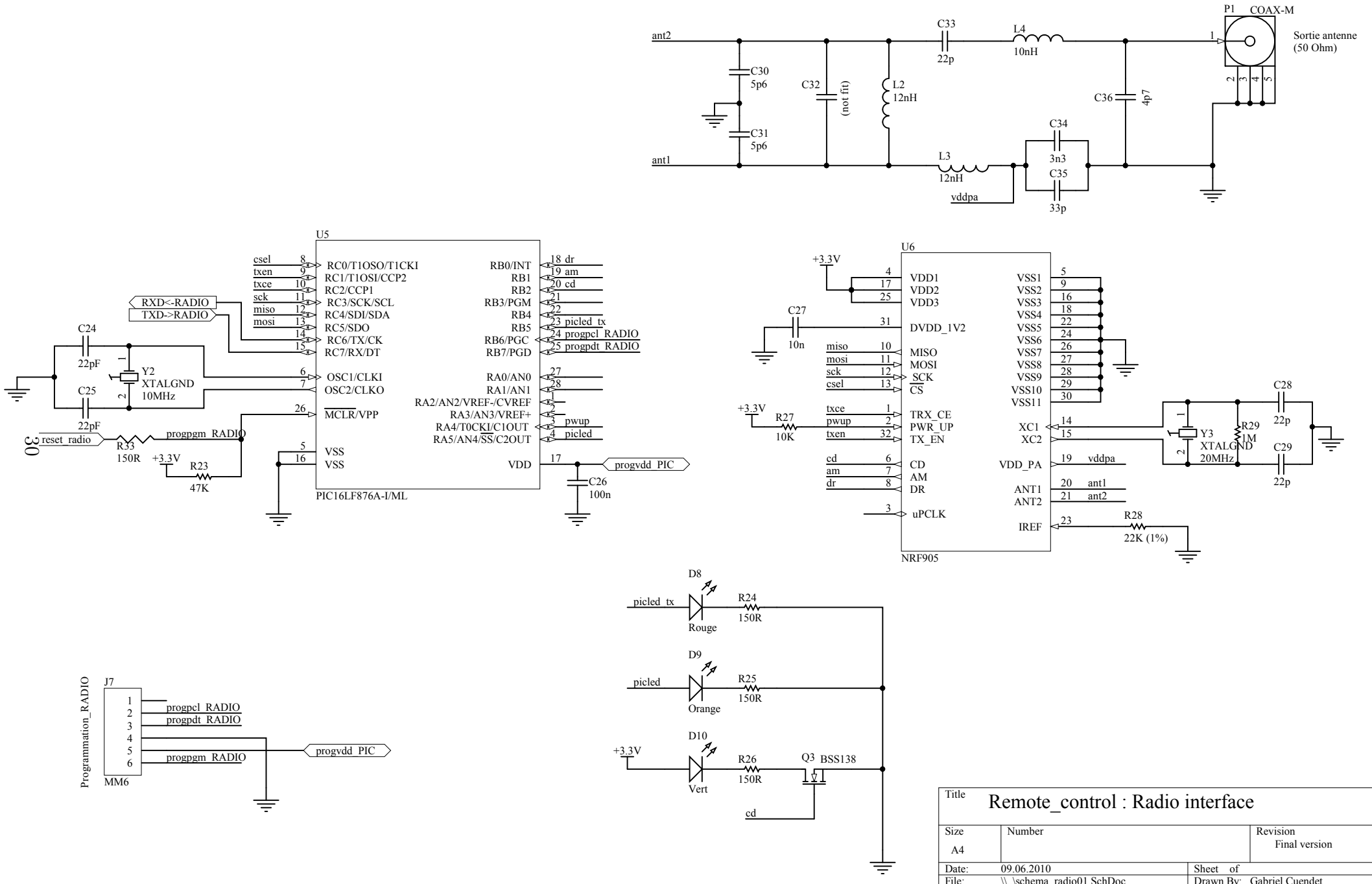
28

Title	Remote_control : User's interface	
Size A4	Number	Revision Final version
Date: 09.06.2010		Sheet of
File: \\..\schema_remoteControl01.SchDoc		Drawn By: Gabriel Cuendet



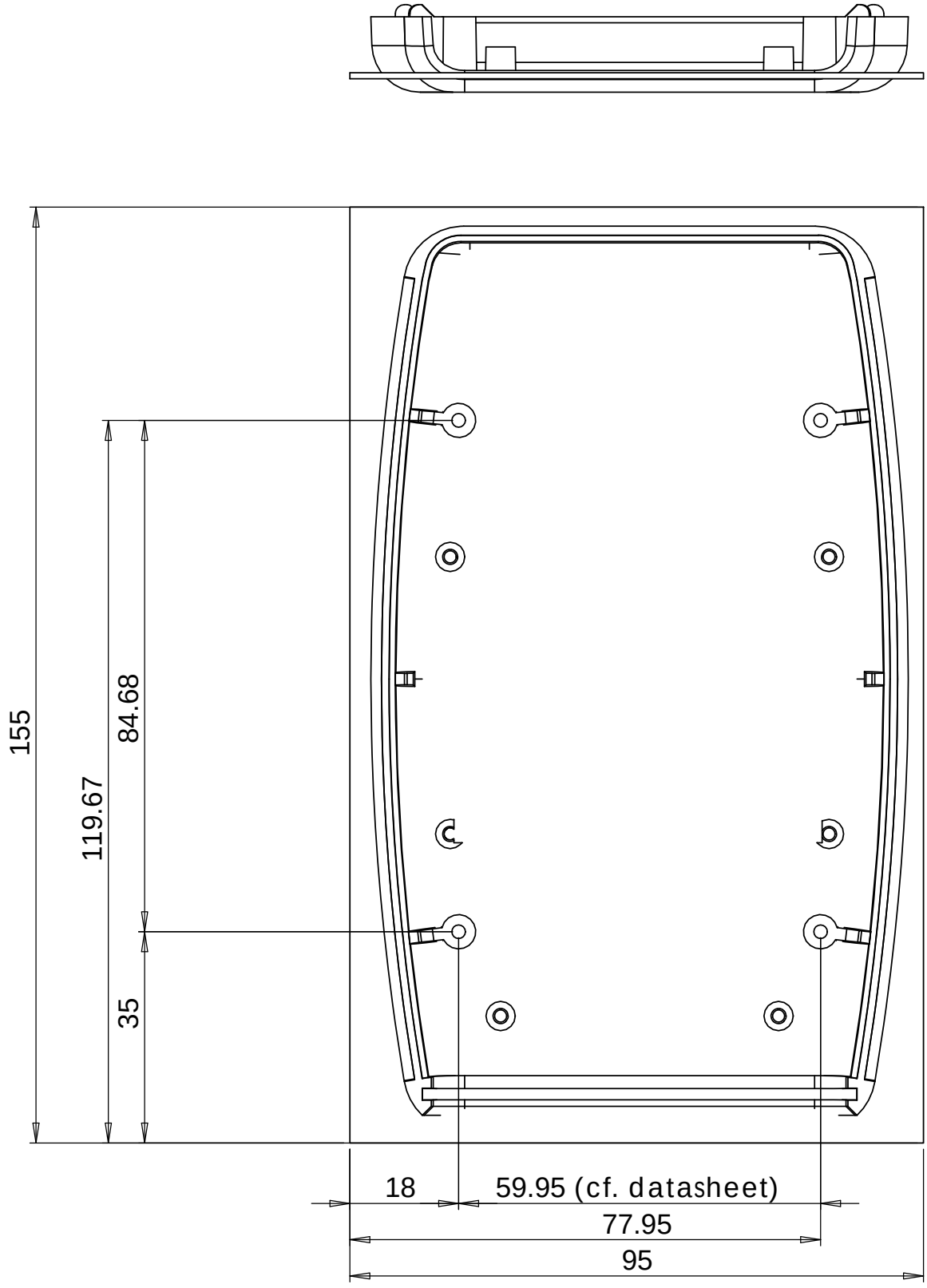
29

Title	Remote_Control : Battery
Size: A4	Number:
Revision: Final version	
Date: 09.06.2010	Sheet of
File: \\..\schema_alim01.SchDoc	Drawn By: Gabriel Cuendet



Title	Remote_control : Radio interface	
Size: A4	Number:	Revision: Final version
Date: 09.06.2010		Sheet of
File: \\..\schema_radio01.SchDoc		Drawn By: Gabriel Cuendet

Qté	Nom	Valeur	Fabriquant	N° Farnell	Footprint
1	C1	100nF, 20V			0603_G
7	C2,C3,C24,C25,C28,C29,C33	22pF			0603_G
9	C4,C7,C10,C16,C18,C19,C20,C21,C26	100nF			0603_G
4	C5,C8,C13,C14	1uF			0603_G
1	C6	22uF, 35V			CAP_SMD_D
1	C9	330nF			0603_G
1	C11	22nF			0603_G
1	C12	22uF, 6.3V			CAP_SMD_A
1	C15	4,7uF			0603_G
1	C17	220uF, 4V			CAP_SMD_B
2	C22,C23	1nF			0603_G
1	C27	10nF			0603_G
2	C30,C31	5,6pF			0603_G
0	*C32*	*not fit*			*0603_G*
1	C34	3,3nF			0603_G
1	C35	33pF			0603_G
1	C36	4,7pF			0603_G
2	**D2,D3**	**LED verte, 3mm**	**Kingbright**	***1142502**	**LED_5MM**
1	**D11**	**LED rouge, 3mm**	**Kingbright**	***1142517**	**LED_5MM**
2	D4,D6	Diode schottky			DSO-C2/X2.3
1	D5	MCL4148	Micromelf		MICROMELF_D_2
1	D8	Led smd rouge			0603_D
1	D9	Led smd orange			0603_D
1	D10	Led smd vert			0603_D
1	J1	Jumper			JUMP
4	J2,J3,J7	MM6	Micromatch		MM6
	*LCD, pour le 2e PCB	MM6	Micromatch		MM6
1	J4	RASM712	Switchkraft	*1608734	RASM712
1	J5	MM4	Micromatch		MM4
1	Joystick	STD-2603AR	Multicomp	1148306	
1	L1	10uH			WE-PD2-S
2	L2,L3	12nH			0603_G
1	L4	10nH			0603_G
1	LCD	BTHQ21605V-COG-FSRE-I2C	Batron	1220409	LCD
1	**P1**	**Connecteur RF, SMA coudé**	**Multicomp**	***1169632**	**CONN_SMA**
2	Q1,Q2	IRF7410			SOIC127P600-8AM
1	Q3	BSS138			SOT95P240-3M
7	R1,R7,R8,R9,R10,R23,R30	47K			0603_G
8	R3,R4,R17,R20,R24,R25,R26,R31	150R			0603_G
3	R5,R6,R27	10K			0603_G
4	R11,R16,R21,R22	1K			0603_G
1	R12	300R			0603_G
1	R13	8.2K			0603_G
1	R15	33K			0603_G
1	R18	4,7K			0603_G
1	R19	25m			RESC5025M
1	R28	22K (1%)			0603_G
1	R29	1M			0603_G
1	R33	150R			RESC1608L
1	SW-ON/OFF	Jumper			JUMP
1	**SW-ON/OFF, facade**	**Slider switch**	**Knitter-Switch**	***807539**	
1	**SW1**	**SPDT Subminiature switch, M6RE**	**Multicomp**	***1550267**	
5	**SW2,SW3,SW4,SW5,SW7**	**Bouton, corps 3FTL6**	**Multimec**	***1132885**	
5		**Bouton, cap. 1S09-19.0**	**Multimec**	***1132918**	
1	U1	PIC18F2580-I/ML	Microchip		QFN-ML28_M
1	U2	LT1571-5,Battery charger			
1	U3	ITC3240, DC/DC converter, 3.3V			DFN6-DC
1	U4	DS2764,Battery monitor			TSSOP16
1	U5	PIC16LF876A-I/ML	Microchip		QFN-ML28_M
1	U6	NRF905,single chip radio trans.			QFN32_M
2	Y1,Y2	Crystal Oscillator, 10MHz			XTALGND
1	Y3	Crystal Oscillator, 20MHz			XTALGND
1	**Boitier Hammond 1553D**	Noir	Hammond	*1511160	
		Bleu translucide		***1511161**	
		Jaune		*1511163	



DIMENSIONS ARE IN MM TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±		
MATERIAL: ABS		
FINISH		
DO NOT SCALE DRAWING		

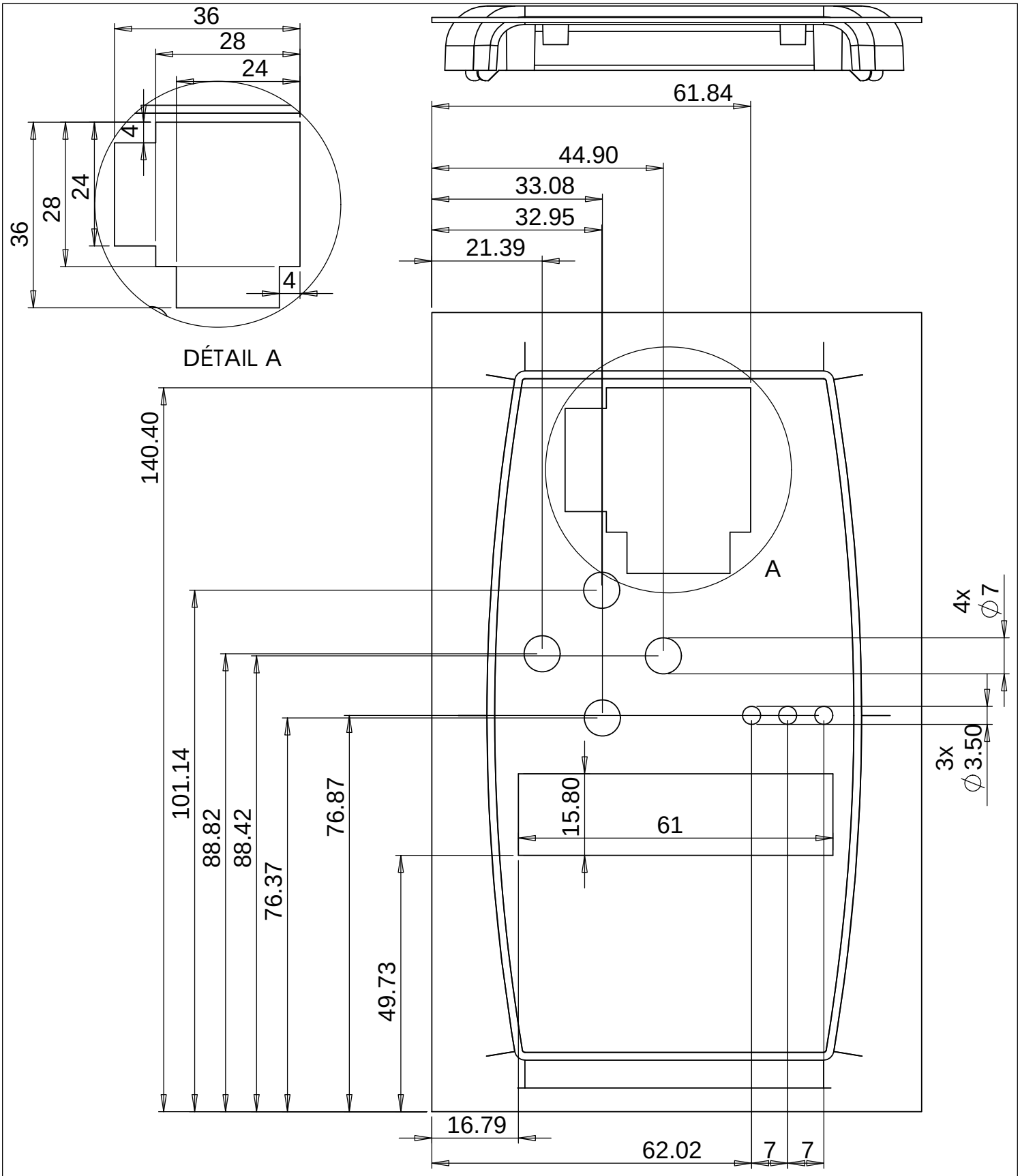
	NAME	DATE
DRAWN	G.C.	01.06.10
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:
ATTENTION : Vue depuis l'intérieur du boitier !

Remote Control, Biorob

FRONT PANEL

SIZE	DWG. NO.	REV.
A	1553D	2.0
SCALE:1:1	WEIGHT:	SHEET 1 OF 2



DIMENSIONS ARE IN MM TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±		
MATERIAL: ABS		
FINISH		
DO NOT SCALE DRAWING		

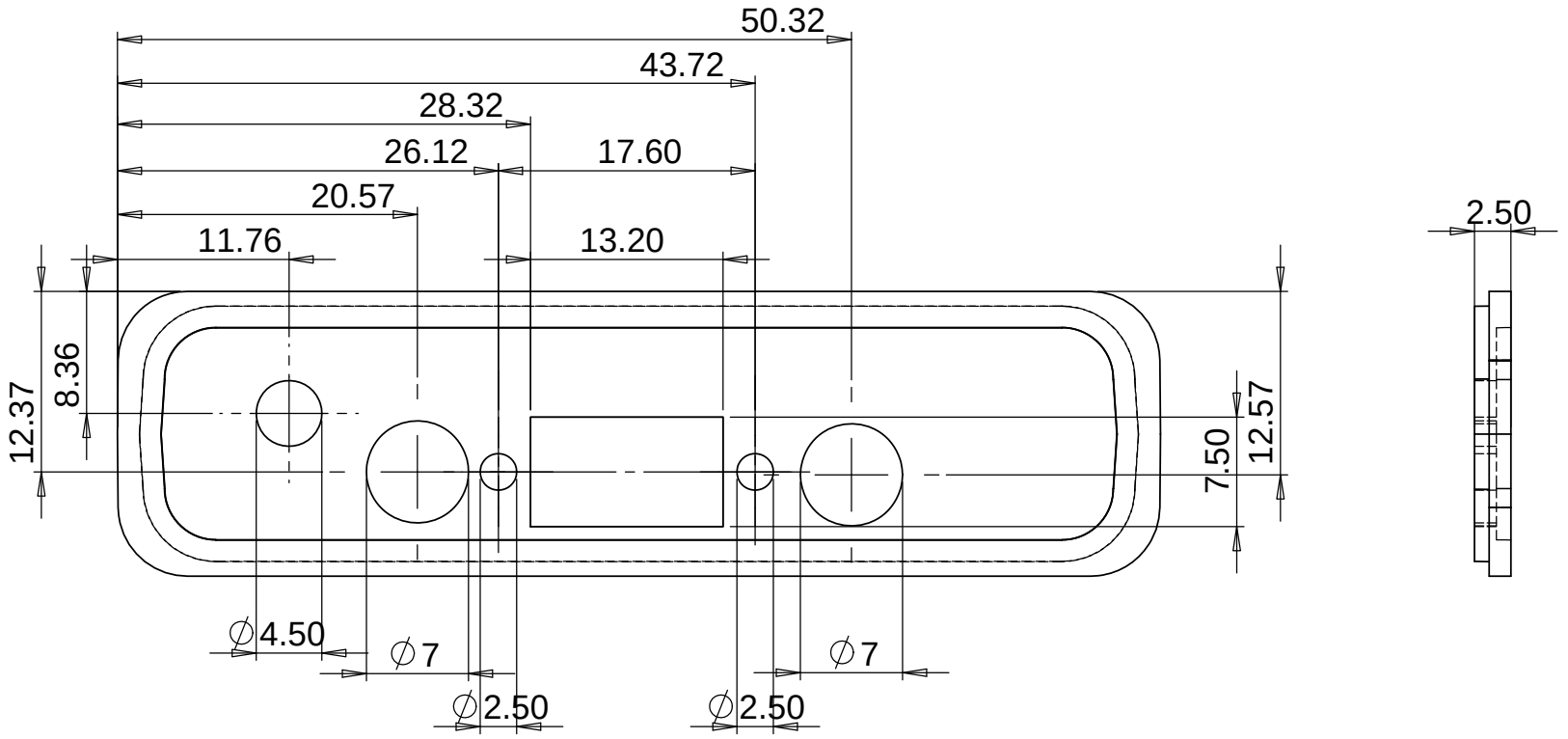
	NAME	DATE
DRAWN	G.C.	01.06.2010
CHECKED		
ENG APPR.		
MFG APPR.		
Q.A.		

COMMENTS:
ATTENTION : Vue depuis l'extérieur du boitier

Remote Control, Biorob

FRONT PANEL

SIZE	DWG. NO.	REV.
A	1553D	
SCALE:1:1	WEIGHT:	SHEET 2 OF 2



UNLESS OTHERWISE SPECIFIED:		NAME	DATE
DIMENSIONS ARE IN MM TOLERANCES: FRACTIONAL ± ANGULAR: MACH ± BEND ± TWO PLACE DECIMAL ± THREE PLACE DECIMAL ±	DRAWN	GC	01.06.10
	CHECKED		
	ENG APPR.		
	MFG APPR.		
INTERPRET GEOMETRIC TOLERANCING PER:	Q.A.		
MATERIAL ABS	COMMENTS:		
FINISH			
DO NOT SCALE DRAWING			

Remote control, Biorob

TITLE:

ENDPANEL

SIZE	DWG. NO.	REV
A	1553D	2.0

SCALE: 2:1	WEIGHT:	SHEET 1 OF 1

5 4 3 2 1

34