WIFI MOUSR

By

Marc Abraldes Velasco

Nicholas Thoele

Isabel Ugedo Perez

Final Report for ECE 445, Senior Design, Spring 2019

TA: Hershel Rege

May 1st 2019

Project No. 70

Abstract

“WIFI Mousr” is an update of the automated cat toy “Mousr” by the start-up company Petronics. This product is equipped with several presence and distance measuring sensors, which allow it to successfully escape from a cat providing a game and distraction for your pet.

In the original version, all movements can be controlled with a phone application via Bluetooth. “WIFI Mousr” is meant to completely recreate the functionality while implementing a WIFI connection with the Smartphone through a new application. Moreover, we are going to design a new functionality, the Auto-Schedule, which allows doing the Auto-Play mode with a designed schedule.

For a successful project, we are going to replace the head microcontroller to the ESP32, which includes WIFI and Bluetooth. We will also rewrite the most part of the drivers for the sensors and main code and design our PCB to place all the sensors. Furthermore, we are going to design a new app to implement the WIFI connection with the microcontroller.

Table of Contents

1. Introduction

Fig. 1 Mousr [1]

"WIFI Mousr "is meant to be an upgrade of an existing product, "Mousr" [1] by Petronics. This device is a combination of several distance and acceleration sensors, engines and a moving tail. Its intended function is to serve as a cat toy/distraction, by running around the room avoiding any obstacles such as walls, or the cat itself. It can be set to Auto-Play mode or controlled in real time by the user through a mobile app, which relies on a Bluetooth connection.

Our role was to replace this Bluetooth bond by a WIFI connection, which should be implemented via an IP address by the client's specification. While this approach would result in a shorter battery life, it has many advantages. First and foremost, WIFI creates a longest signal range which means the user would not have to worry about maintaining proximity to the moving device, which might prove inconvenient and even challenging. The only requirement would be for the home router signal to be strong enough to reach both the smartphone and the Mousr device.

Another advantage would be an increased data transfer rate between the microcontroller and the phone. This is important to Petronics since they plan on adding data analytics to the toy in the future, which makes accessing sensor readings and data in real time essential.

In order to develop the WIFI connection, we set out to recreate Mousr functionality in a new chip, the ESP32 [2] suggested by Petronics. The reasoning behind this was that it is a very popular choice for real time systems, and there was a lot of documentation available to us. It also contains both Bluetooth and WIFI, which is necessary for the connection we aim for. All coding was carried out through FreeRTOS, which is C based. This is Petronics' environment of choice, so we needed to use it to be able to integrate with them.

On the hardware aspect of the project, our work can be summarized as designing the power and data circuits needed for Mousr to work, designing a viable PCB and soldering all components, coding and debugging the I^2C drivers, which is used by most of the sensors, and transmitting instructions to the engines via UART connection. We also created a new app to manage the WIFI connection, and set up a bidirectional bond between ESP32 and IP address as well as between phone app and that same address.

2. Design

The design aspect of our project is mostly centered on the PCB circuits, the components we selected and the FreeRTOS behavioral code. The WIFI and Bluetooth protocols we worked with are standardized and required no design input from us, as is the case with the I^2C sensor drivers. The full project integrations were mapped as shown in our block diagram, in Figure 2.

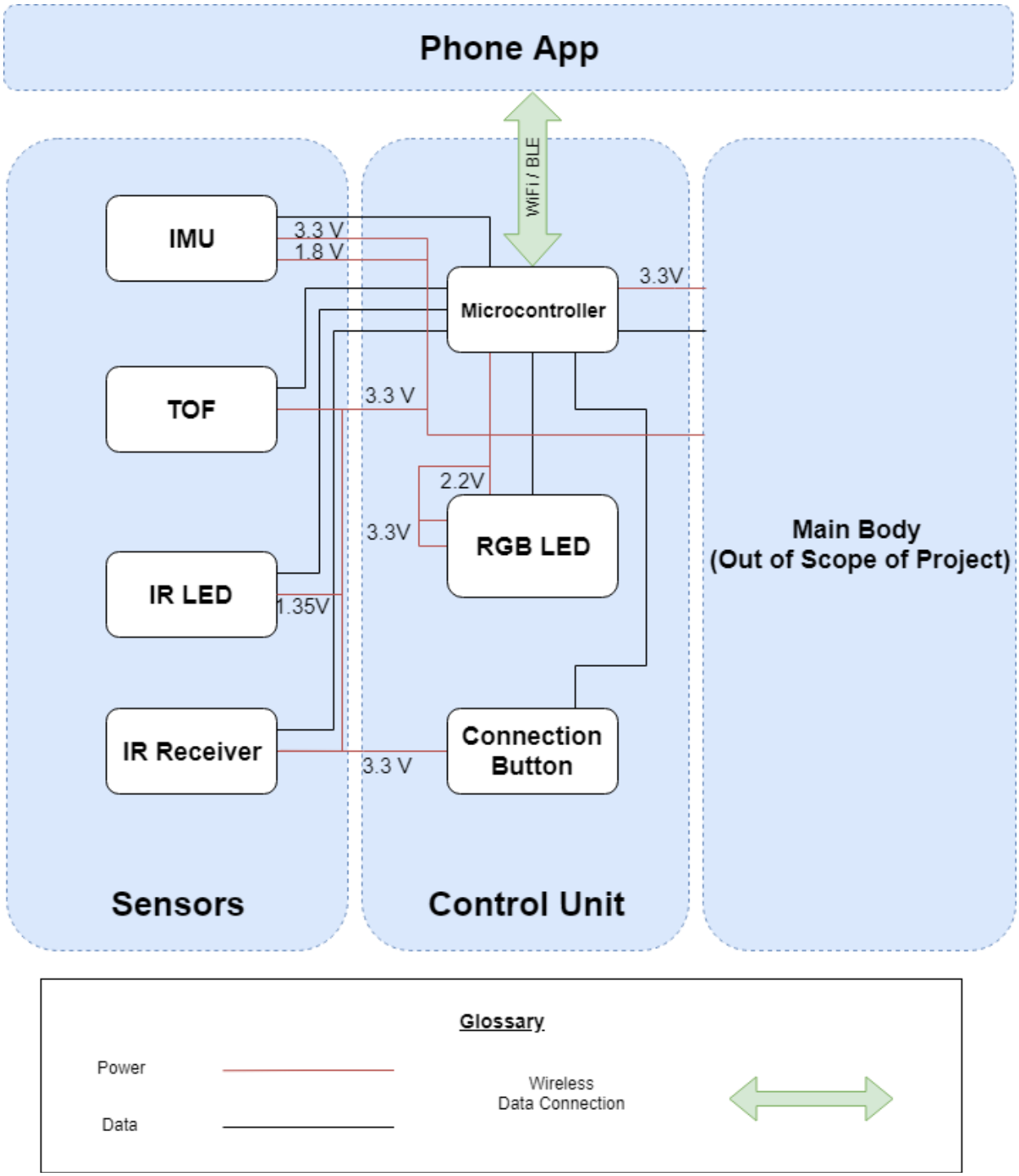

Fig.2 Block Diagram

2.1 Wireless Communication / Phone App Module

The first goal of the new Android application was to create a connection process that was as quick and user friendly as possible without the need to leave the app's interface during the process. From the main menu, shown below in Figure 3, the user would select the connection button, denoted by the standard WIFI symbol.

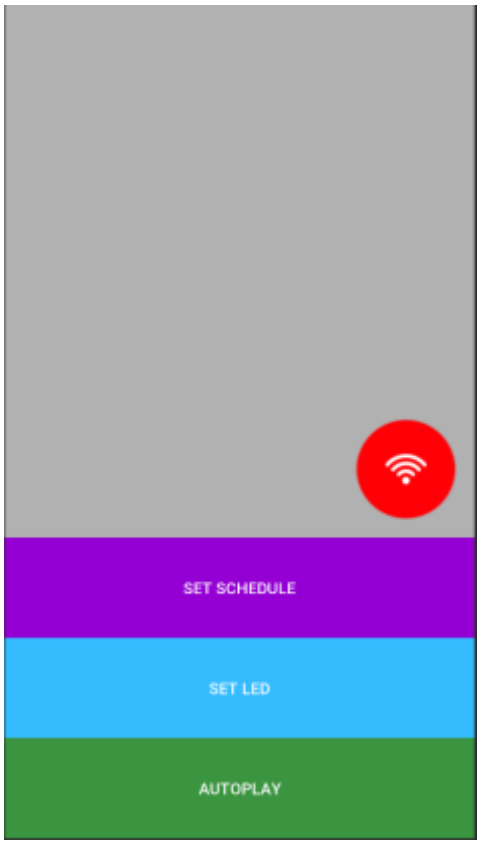

Fig.3 App Main Screen

In the original design, this process required four different screens also known as activities, but after some research we were able to reduce it to the single activity, shown in Figure 4, that displayed a list of discoverable devices over Bluetooth. After selecting a device, the phone and the device would pair and the user would input the password to the WIFI network they were currently connected to and press a button to send it over. Luckily the Android WIFI libraries [3] allow for the access of the SSID, also known as the network name, without the need for searching for all the local networks and selecting the one the user is already on. After sending this SSID and password the ESP32 would set-up the IP address for further communication.

Fig.4 App Paring Screen

After this connection and communication set-up process has completed the rest of the app was to be able to set the Auto-Schedule, change the color of the RGB LED, and set the Mousr into Auto-Play mode with as little user interaction as possible in order to simplify and expedite the user experience as well as the data transfer between the app and Mousr.

In order to set a play schedule the user would select the 'Set Schedule' button on the app's main menu. From here, four lists are shown containing the day of the week, the hour in a standard 12 hour setting, minutes from 0-55 in intervals of five minutes, and a selection of either A.M. or P.M. After selecting from each list, the user would press the 'Set' button and the schedule would be sent to the Mousr.

Similarly in order to set the RGB LED color, the user would select the 'Set LED' button from the main screen. This would present the user with a list of six colors: red, green, blue, purple, yellow, and white. The user would select one and it would be sent to the Mousr.

The Auto-Play functionality ran differently. The user would simply select the 'Autoplay' button on the main menu, and the Mousr would be sent a code to run Auto-Play at that time if it was off or it would turn off it was on.

Unfortunately, as explained later in Design Verification, much of this was unable to be implemented due to lack of the communication channel being properly set up.

2.2 Control Unit

2.2.1 Microcontroller

For the design of the microcontroller, Petronics provides us two different microcontrollers, the ESP32 Serial [4] and the ESP32 Wrover Kit [5]. The main difference between them is the number of pins, so, we decided to use the Wrover Kit because it has more accessible GPIO pins for all the connections that we need from the sensors.

For the power supply, we have 4.1V from the body battery, so we need a voltage regulator to get 3.3V, the voltage that we need for the microcontroller and most of the components of our circuit, as we can see in Figure 5.

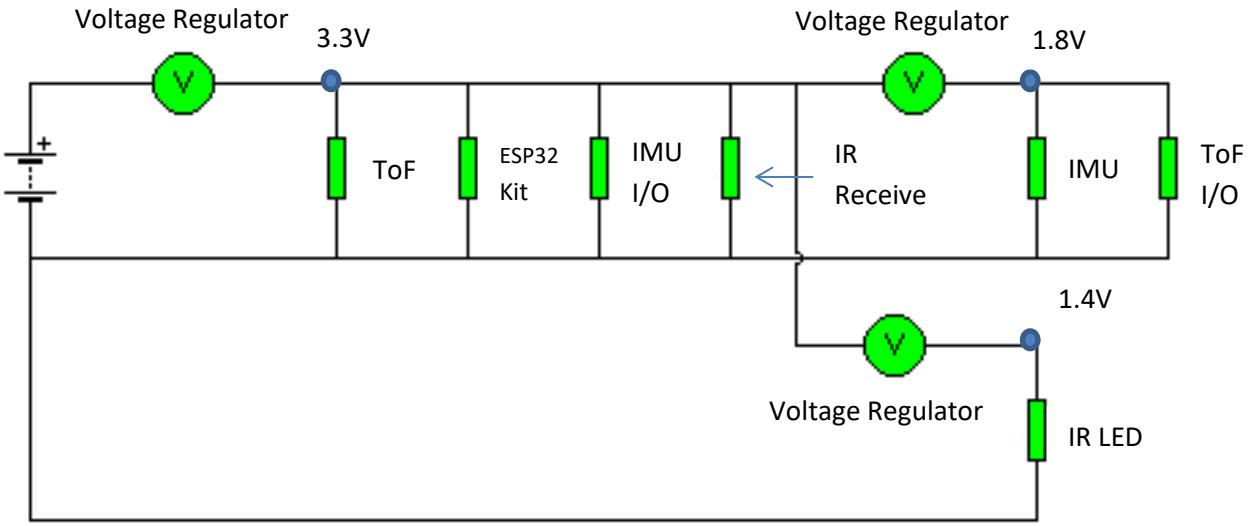

Fig.5 Power Circuit Schematics

For the data connections, we needed to drive all digital inputs and outputs into the ESP32. Our solution was to include through holes in the PCB which will be connected to the PCB by soldering and to the microcontroller. This was meant to allow us to switch between the ESP32 Wrover Kit and the ESP32, of for changing the pins depending on the signal requirements that we need.

2.2.2 Push Button

For the correct functionality of the pushbutton with the microcontroller, we set an interrupt that detects the falling edge in the GPIO. For doing that, we have to design an extra circuit for the button, because if can create a short circuit in the microcontroller. The circuit implemented is in Figure 6.

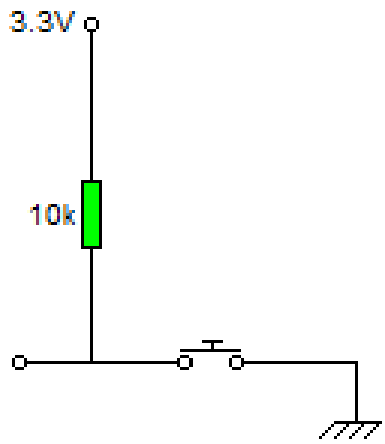

Fig.6 Pushbutton Circuit

As we can see in Figure 6, each time we press the pushbutton, we put the input pin to low voltage, and the interrupt occurs. If it is not pressed, the pin remains high and nothing happens in the microcontroller.

2.2.3 RGB LED

Regarding the RGB LED, straightforward calculations were computed to obtain the required resistor values for each color port. First, we obtained each diode internal resistance by looking at the nominal values provided in the datasheet [6]:

$$R_{int} = \frac{V_{nom}}{I_{typical}} \qquad 2.1$$

Since we knew the LED would be connected to a GPIO from the microcontroller, which provides us a signal of 3.3V, we wrote a simple voltage divider:

$$V_{LED} = \frac{R_{int}}{R_{int} + R_x} \qquad 2.2$$

Resulting values and corresponding parameters are displayed in Table 1:

Table 1. LED Resistances

LED COLOR	R_{int} (Ω)	V_{LED} (v)	R_x (Ω)
Red	140	2.1	140
Green	270	2.7	270
Blue	300	3	300

2.3 Sensor Module

All the sensors that we are using for the project are going to be on the same PCB. As we explained before, the battery of the Mousr body is providing 4.1V and we use a 3.3V voltage regulator for stepping down the voltage. The problem is that some of the sensor needs less voltage than that, so we are going to use two more voltage regulators to have 1.8V and 1.4V, as we can see in Figure 5.

2.3.1 TOF

For the Time of Flight, we follow the schematic connection that the datasheet from the fabricant recommends, which you can see in the Figure 7.

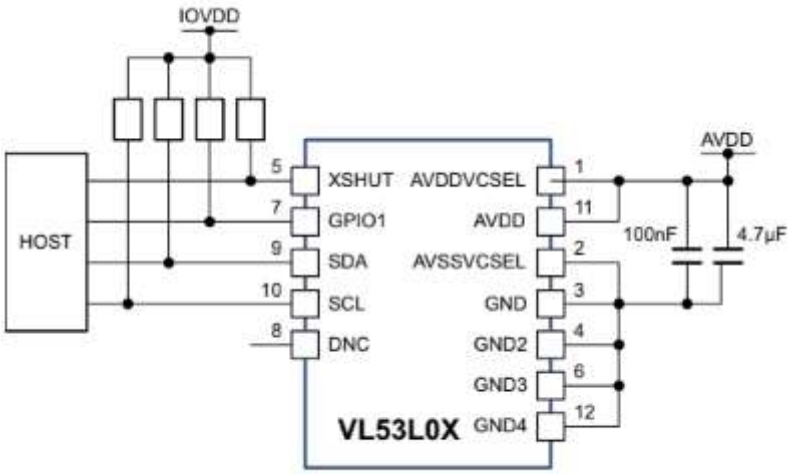

Fig.7 TOF Circuit

For the values of the pull-up resistors of Figure 7, we adapted respecting the required minimum and maximum values given by the following equations:

$$R_{min} = \frac{V_{cc} - V_{ol}}{I_{ol}} \qquad 2.3$$

$$R_{max} = \frac{t_r}{0.8473 * C_b} \qquad 2.4$$

Where: - V_{cc} is the sensors DC supply

- V_{ol} is the max voltage that can be interpreted as a digital low

- I_{ol} is the current which corresponds to this voltage.

- t_r is the maximum rise time

- C_b the capacitive load for each bus line.

A lower resistor value would result in smaller power consumption, but also increased transfer rate due to reduced RC delay. We settled on intermediate values since we did not have any specific power consumption or speed specs.

2.3.2 IMU

For the IMU, the manufacturer provides us the values of the pull-up resistances in the datasheet for the I^2C protocol, so, we designed the PCB making the correct connections between the sensor and the microcontroller, following the indications from Figure 8.

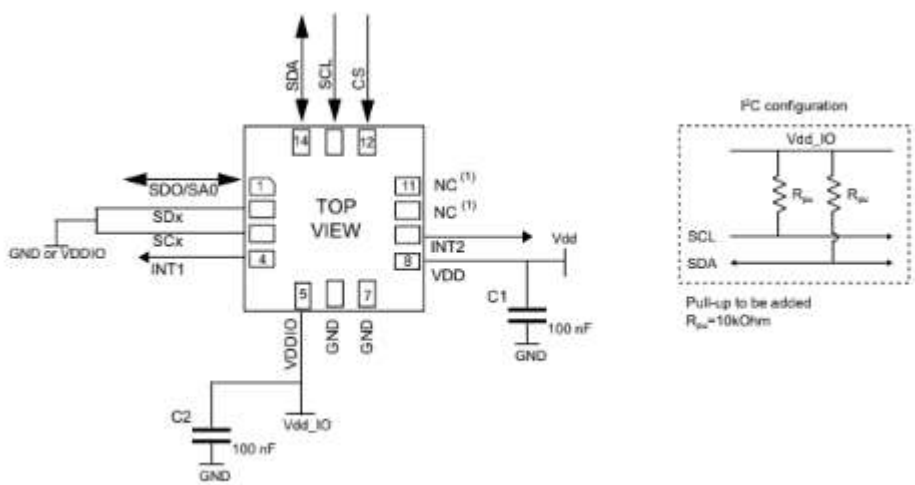

Fig.8 IMU Circuit

2.3.3 IR LED and IR Receiver

The IR LED is used to provide a reference value for the IR Receiver, which is designed to detect variations in heat intensity, rather than presence of heat. The only requirement is that we have to take care in the design the position, because the emitter and the receiver have to be separated by a certain distance and it can not be any object between them.

For the power supply of each component, the IR LED needs a specific sequence of pulses for a correct output in the IR Receiver, which we can see in Figure 9.

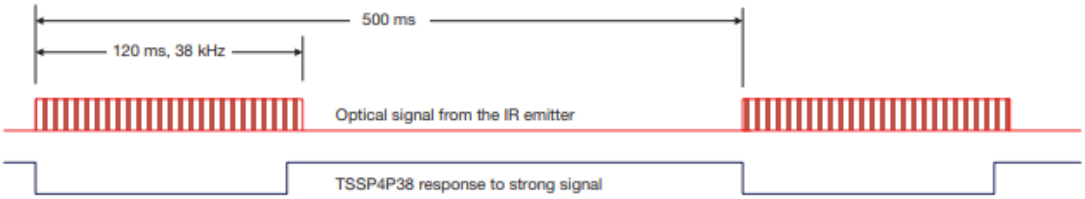

Fig.9 Signal Supply for IR LED and IR Receiver Response

3. Design Verification

Due to all the PCB issues we discussed throughout the design section of this paper, we were not able to set up the TOF and IMU communication. Therefore, many of our block requirements were not met, specifically those regarding sensors and integration. We will now discuss each of the requirements and state weather it was successfully implemented in the prototype.

3.1 Wireless Communication / Phone App Module

3.1.2 Application Communication

There were four main parts of the phone application, connecting to the Mousr over Bluetooth being the first of these. In order to show this we set both the phone and the Mousr into their pairing modes. Unfortunately, the Mousr would not pair, and as we dug deeper found it was discoverable but not connectable. This issue could not be resolved as setting the proper scan codes for the preset microcontroller Bluetooth connection protocol resulted in a crashing of its subroutine and made it undiscoverable entirely.

Not being able to do this, we settled for hardcoding the Mousr's WIFI credentials and allowed it to set its own IP address. This process was successful; however, in testing the connection in the lab we realized the access to this IP address was specific to the user's login in information, meaning one could not access the IP address while logged in to Illinois Net with their login if it was set up using someone else's. In finding this out, we decided it easiest to develop a barebones app designed specifically for this connection. This worked and allowed us to go on to the next of the requirements by changing the RGB LED color from the app.

Using the new app, while connected we simply selected the color we wanted from the small list we developed of red, green, and blue and the ESP32 set the RGB LED to that color. This being a success our next requirement was to be able to set a schedule.

This was not developed due to time constraints, but seeing as the RGB LED color selection worked, it is fair to say that given time to set up a function that worked with the ESP32's Real Time Clock, this would have been achievable. To show this, due to the errors in the PCB and inability to use the Auto-Play functionality, we could simply have the RGB LED change to one of the preset colors at the set time.

3.2 Control Unit

3.2.1 Microcontroller

To test all the ESP32 code we built a small circuit in a breadboard, which included several LEDs and resistors. We also coded a screen display to print a message informing the user of the

battery status. To simulate a battery or power source we built a voltage divider and connected our circuit to different voltage points, thus simulating full, empty and half charged battery.

3.2.2 Push Button

The push button receives current from the PCB circuit, so in order to test it we only needed to connect the board to its input voltage and ground. We then displayed the output in the oscilloscope. After ensuring it worked, we connected the output to the ESP32. We observed the whole prototype was instantly shorted whenever the button was pressed, which led us to realize we needed to connect a resistor between the button input and output ports.

3.2.3 RGB LED

The RGB led was easy to test. First, we recreated its functionality by using three LEDs on our breadboard. After ensuring the code was appropriately handling the LEDs, we moved on to the actual RGB LED inside the PCB, and had no issues with it.

3.3 Sensor Module

3.3.1 TOF and IMU

As we mentioned before, the PCB issues we encountered did not allow us to test the sensors. We tried to download the FreeRtos drivers into the ESP32 and got an invalid architecture error. We then switched to Arduino, which has a simplified I^2C feature and tried the new drivers. We got results from the TOF, but they didn't change at all or show any response to outside activity.

After establishing the I^2C connection, our first PCB had shortcuts. We took all the components off and could not find the source. We decided to start over on a new PCB which we did not have time to complete.

3.3.2 IR LED and IR Receiver

The IR LED was a very hard circuit to debug, which we had not foreseen in our design document. The IR receiver did not respond to the LED when we set it up on the breadboard, and we had no way to know if the LED was flashing. We tried measuring the voltage and current going through the LED and comparing them to those on the datasheet specifications. However, the datasheet only mentioned max current and voltage drop values, not standard ones. We did not end up having enough time to get the system working.

4. Costs

4.1 Parts

Table 2 Detailed Part Cost

Part	Part Specification /Manufacturer	Supplier	Cost / Unit	Quantity	Prototype Cost	Cost of Manufacturing
ESP32Wrover Kit V4.1	*GC-ESP-WROVER-KIT / Espressif*	GridConnect	$40.00	1	$40.00	$5.00
IMU	LSM6DSLTR / STMicroelectronics	Mouser	$3.98	4	$15.92	$2.50
TOF	VL53L0CXV0DH/1 / STMicroelectronics	Mouser	$4.5	4	$18.00	$2.83
RGB LED	CLY6D-FKC CK1N1D1BB7D3D3 m/ Cree	DigiKey	$0.318	10	$3.18	$0.29
IR Receiver	TSSP77038TT / Vishay Semiconductors	Mouser	$1.79	4	$7.16	$0.82
IR LED	VSMB3940X01-GS08 / Vishay Semiconductors	DigiKey	$0.73	5	$3.65	$0.34
Pushbutton	PTS810 SJG 250 SMTR LFS / C&K	DigiKey	$0.31	5	$1.55	$0.24
FCC Connector	687112183722 / Wurth Electronics	Mouser	$1.83	4	$7.32	$1.22
Voltage regulator 3.3V	LM1117T-3.3/NOPB / Texas Instruments	Mouser	$1.54	4	$6.16	$0.89
Voltage regulator 1.8V	LM1117MPX-1.8/NOPB / Texas Instruments	DigiKey	$2.78	4	$11.12	$0.68
Voltage regulator 1.4V	NCP699SN14T1G / ON Semiconductors	Mouser	$0.47	4	$1.88	$0.14
PCB	-	PCBWay	$5.00	1	$5.00	$0.50
Shipment Cost					$42.97	-
TOTAL COST					$163.91	$16.05

4.2 Labor

We obtained an approximation of cost of labor using the following equation:

$$Total\ Cost = \frac{(Avg\ Salary\ of\ BS\ in\ EE) + (Avg\ Salart\ of\ BS\ in\ CE)}{2} * \frac{1\ Year}{Labor\ Weeks * 45\ Hours} * (Total\ Time\ [h]) \quad * (Nº\ Team\ Members) \qquad 4.1$$

The average salary data was obtained from the ECE website of UIUC. Total time was estimated to be 110 hours per worker. The resulting value of the equation 4.1 is shown below.

$$Total\ Cost = \frac{\$67000 + \$84250}{2} * \frac{1\ Year}{52 * 45\ Hours} * (110) * (3)$$

$$Total\ Cost = \$10{,}665.06$$

5. Conclusion

5.1 Encountered difficulties

During the four weeks we worked on the project, we encountered several design issues. They were the main reasons why we were not able to connect to our sensors, and we believe they could have been fixed if we had had a bigger time frame. Our main takeaway is to give the appropriate attention to timing and scheduling in large projects and to not underestimate the extra time that might prove essential to address unforeseen problems.

We also learned a lot from our mistakes, and especially from coming up with ways to work around them and debug the affected circuits. We now feel more prepared to face real world projects than we did at the beginning of the semester, and have a better idea of what to expect from teamwork.

5.2 Accomplishments

Our main accomplishments this semester are software related, particularly the WIFI, Bluetooth and microcontroller FreeRTOS coding. The learning curve was very steep, since the ESP32 language was very low level as we mentioned before, and it was not conductive for debugging. Once we were up to speed with the language, the work sped up considerably.

Overall, we became more skilled in several areas including Eagle PCB design, soldering, creating android apps, setting up Bluetooth and WIFI connections which allow for data transfers and managing UART connections.

5.3 Work methodology

The course is group work based, which we feel was a very efficient way to approach the project. We managed to stay involved in all aspects of the project while splitting the workload and did not have any issues. We also worked with Petronics and attended several meetings, which we believe gave us real work experience in dealing with clients. It gave the project a whole other aspect, which we very much enjoyed and learned from.

5.4 Future work

As we mentioned throughout the paper, our project failed due to PCB printing and design issues. We would like to order a new PCB and try it out, provided we have enough time this semester. This will allow us to test our drivers and integration code, and to see whether we were on the right path.

References

[1] Petronics Inc, "*Mousr*", 2019. [Online]. Available: *https://petronics.io/mousr*

[2] ESP32, "*Features and Specifications, The internet of things with ESP32*", 2019. [Online]. Available: http://esp32.net/

[3] Android Developers, "*Android.net.wifi*", 2019. [Online]. Available: https://developer.android.com/reference/android/net/wifi/package-summary.html

[4] Espressif Systems, "*ESP32 series Datasheet*", 2019. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf

[5] Espressif Systems, "*ESP32-WROVER-B Datasheet*", 2019. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-wrover-b_datasheet_en.pdf

[6] CREE, "*Cree® PLCC6 3 in 1 SMD LED CLY6D-FKC*", 2019. [Online]. Available: https://www.cree.com/led-components/media/documents/ds-1321-CLY6D-FKC.pdf

Appendix A. Requirement and Verification Table

Table 3 System Requirements and Verifications for Phone Application

Requirement	Verification	Status (Y or N)
Able to establish WIFI connection with Mousr with reliability of 90%	1. From the home screen of the app, select the option that shows the WiFi symbol 2. Initiate pairing state of Mousr 3. From phone app, select the Mousr device 4. Select the desired WiFi network to connect to 5. Enter the network's password 6. Ensure both app and RGB LED indicate connection is successful (app will show success screen. LED will show turn green) 7. Repeat 9 more times, ensuring a connection 9/10	Y
Set schedule that Mousr auto plays ± 5 minutes from	1. Assure app is connected to the Mousr 2. From home screen of app, select Set Schedule option 3. Set time and date to be within a minute of the time at testing 4. Ensure that at the desired time the device begins moving	N
Able to change color of RGB LED	1. Assure app is connected to Mousr 2. From home screen of app, select LED color option 3. Select color from available options 4. Ensure RGB LED is bright and visibly the selected color	Y
Able to set Mousr to auto-play mode	1. Assure app is connected to Mousr 2. From home screen of app, select auto-play option 3. Ensure Mousr is moving	N

Table 4 System Requirements and Verifications for Microcontroller

Requirement	Verification	Status (Y or N)
Analog/Digital/PWM pins to make the connections.	1. Connect RGB LED circuit 2. Set PWM signal to 2.2 V for microcontroller to allow red output of LED 3. Ensure LED is bright and showing red color	Y

Table 5 System Requirements and Verifications for Button

Requirement	Verification	Status (Y or N)
Must be easy to press	Press button without unnecessary amount of work	Y

Table 6 System Requirements and Verifications for RGB LED

Requirement	Verification	Status (Y or N)
Colored light must be clearly visible, for all three colors.	1. Connect RGB LED circuit 2. Set PWM signal to 2.2V from microcontroller to allow red output of LED 3. Ensure LED is bright and showing red color 4. Repeat steps 2 & 3 for green and blue except instead setting signal to 3.3V	Y
LED should be adjusted to flash on a .5 ± .1 s period.	1. Connect RGB LED circuit 2. Generate 3.3V pulse signal from microcontroller to blue pin of LED 3. Ensure LED is flashing and blue	Y

Table 7 System Requirements and Verifications for Time of Flight

Requirement	Verification	Status (Y or N)
Performance should accomplish a field of vision of 20° ± 1°	1. Power TOF sensor on breadboard and connect to oscilloscope 2. Measure out 20° area 3. Hold notebook 80cm away from sensor and sweep through measured FOV 4. Ensure oscilloscope signal indicates an obstruction	N
Should reliably detect between 800-900mm ± 50mm	1. Power TOF sensor on breadboard and connect to oscilloscope 2. Measure out area 800-900mm away from sensor 3. Sweep a notebook through area 4. Ensure oscilloscope signal indicates an obstruction	N

Table 8 System Requirements and Verifications for IMU

Requirement	Verification	Status (Y or N)
IMU is operating in selected [by micro] power consumption mode	1. For high performance mode: current input must be 0.65 mA ±10% 2. For normal mode: current input must be 0.45 mA ±10% 3. For low power mode: current input must be 0.29 mA±10%	N
Readings are 0g, 0g and 1g ±10% for axes X,Y and Z respectively when placed horizontally	1. Place sensor horizontally 2. Output must show a 0g force in both X and Y axes 3. Output must show a +1g±10% force in Z axis, which will be in binary two's complement.	N

Reading in Z axis changes from +1g ±10% to -1g ±10% when sensor is flipped.	1. Put the device on a straight surface and read output with oscilloscope 2. Flip device and read output again, converting from two's complement 3. Change in Z axis output should be indicative of 180 degree turn	N

Table 9 System Requirements and Verifications for IR LED

Requirement	Verification	Status (Y or N)
Must emit IR reference frequency	1. Set up IR receiver and IR LED circuits 2. Probe IR receiver output pin 3. Ensure that pin produces signal verifying a wavelength of 940 ± 20nm is detected	Y

Table 10 System Requirements and Verifications for IR Receiver

Requirement	Verification	Status (Y or N)
The directivity angle should allow for a tolerance of 40° ± 5°	1. Power sensor on breadboard and connect to oscilloscope 2. Measure out 40° area 3. Sweep a notebook through area 4. Ensure oscilloscope signal indicates an obstruction	Y

Appendix B. Microcontroller Code

```
/*Define Libraries*/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <time.h>
#include <sys/time.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "esp_sleep.h"
#include "esp_log.h"
#include "rom/uart.h"
#include "driver/rtc_io.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "driver/ledc.h"
#include "esp_err.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "freertos/semphr.h"
#include "driver/adc.h"
#include "esp_adc_cal.h"
#include "driver/uart.h"
#include "soc/uart_struct.h"
#include "string.h"


/*Define Constants*/
/*Constants for TIMERS -> PWM SIGNAL and LEDs OUTPUT PIN*/
#define LEDC_HS_TIMER          LEDC_TIMER_0
#define LEDC_HS_MODE           LEDC_HIGH_SPEED_MODE
#define LEDC_HS_CH0_GPIO       (18) /*GREEN*/
#define LEDC_HS_CH0_CHANNEL    LEDC_CHANNEL_0
#define LEDC_HS_CH1_GPIO       (19) /*BLUE*/
#define LEDC_HS_CH1_CHANNEL    LEDC_CHANNEL_1
#define LEDC_HS_CH2_GPIO       (4) /*RED*/
#define LEDC_HS_CH2_CHANNEL    LEDC_CHANNEL_2
#define LEDC_TEST_CH_NUM       (3)    /*Number of LED*/
#define LEDC_DUTY_BLINK        (4000) /*Bright BLINK*/
#define LEDC_DUTY_CSTN         (8000) /*Bright CONSTANT*/
#define LEDC_DUTY_ZERO         (0) /*Bright OFF*/

/*Button PIN and level to compare*/
#define BUTTON_GPIO_NUM_DEFAULT     13
#define BUTTON_WAKEUP_LEVEL_DEFAULT     0   /*Button connected to high, compare with low = 0*/
```

```
#define DEFAULT_VREF    1100        //Use adc2_vref_to_gpio() to obtain a better estimate
#define NO_OF_SAMPLES   100          //Multisampling

#define TXD_PIN (GPIO_NUM_16)
#define RXD_PIN (GPIO_NUM_5)

struct LEDs{
   int LED_number;
   int blink;
};

static esp_adc_cal_characteristics_t *adc_chars;
static const adc_channel_t channel = ADC_CHANNEL_6;     //GPIO34 if ADC1
static const adc_atten_t atten = ADC_ATTEN_DB_11;
static const adc_unit_t unit = ADC_UNIT_1;

static const int RX_BUF_SIZE = 1024;

void LED_control_ON(ledc_channel_config_t ledc_channel[], struct LEDs LED_number){
   if(LED_number.blink){
      /*LED ON*/
      ledc_set_duty(ledc_channel[LED_number.LED_number].speed_mode,
ledc_channel[LED_number.LED_number].channel, LEDC_DUTY_BLINK);   /*Set PWM signal*/
      ledc_update_duty(ledc_channel[LED_number.LED_number].speed_mode,
ledc_channel[LED_number.LED_number].channel);              /*Update PWM signal*/
   }
   else{
      /*LED ON*/
      ledc_set_duty(ledc_channel[LED_number.LED_number].speed_mode,
ledc_channel[LED_number.LED_number].channel, LEDC_DUTY_CSTN);   /*Set PWM signal*/
      ledc_update_duty(ledc_channel[LED_number.LED_number].speed_mode,
ledc_channel[LED_number.LED_number].channel);              /*Update PWM signal*/

   }
}
void LED_control_OFF(ledc_channel_config_t ledc_channel[], struct LEDs LED_number){
      /*LED OFF*/
      int i;
      for(i=0; i<3; i++){
         LED_number.LED_number = i;
         ledc_set_duty(ledc_channel[LED_number.LED_number].speed_mode,
ledc_channel[LED_number.LED_number].channel, LEDC_DUTY_ZERO);              /*Set PWM signal*/
         ledc_update_duty(ledc_channel[LED_number.LED_number].speed_mode,
ledc_channel[LED_number.LED_number].channel);              /*Update PWM signal*/
      }
      vTaskDelay(pdMS_TO_TICKS(500));
}
static void check_efuse(){
```

```
    //Check TP is burned into eFuse
    if (esp_adc_cal_check_efuse(ESP_ADC_CAL_VAL_EFUSE_TP) == ESP_OK) {
        printf("eFuse Two Point: Supported\n");
    } else {
        printf("eFuse Two Point: NOT supported\n");
    }

    //Check Vref is burned into eFuse
    if (esp_adc_cal_check_efuse(ESP_ADC_CAL_VAL_EFUSE_VREF) == ESP_OK) {
        printf("eFuse Vref: Supported\n");
    } else {
        printf("eFuse Vref: NOT supported\n");
    }
}
static void print_char_val_type(esp_adc_cal_value_t val_type){
    if (val_type == ESP_ADC_CAL_VAL_EFUSE_TP) {
        printf("Characterized using Two Point Value\n");
    } else if (val_type == ESP_ADC_CAL_VAL_EFUSE_VREF) {
        printf("Characterized using eFuse Vref\n");
    } else {
        printf("Characterized using Default Vref\n");
    }
}
int get_battery_level(){
    uint32_t adc_reading = 0;
    //Multisampling
    for(int i = 0; i < NO_OF_SAMPLES; i++){
        if(unit == ADC_UNIT_1){
            adc_reading += adc1_get_raw((adc1_channel_t)channel);
        }
        else{
            int raw;
            adc2_get_raw((adc2_channel_t)channel, ADC_WIDTH_BIT_12, &raw);
            adc_reading += raw;
        }
    }
    adc_reading /= NO_OF_SAMPLES;
    //Convert adc_reading to voltage in mV
    uint32_t voltage = esp_adc_cal_raw_to_voltage(adc_reading, adc_chars);
    return voltage;
}

int wifi_connection(){
    int64_t t_before_us = esp_timer_get_time();
    int64_t t_after_us = esp_timer_get_time();
            do {
                vTaskDelay(pdMS_TO_TICKS(10));
                t_after_us = esp_timer_get_time();
```

```
            }   while ((t_after_us - t_before_us)<5000000);
    static int i=0;

    if(i==0){
        i = 1;
    }
    else{
        i = 0;
    }
    return i;
}
void init() {
    const uart_config_t uart_config = {
        .baud_rate = 115200,
        .data_bits = UART_DATA_8_BITS,
        .parity = UART_PARITY_DISABLE,
        .stop_bits = UART_STOP_BITS_1,
        .flow_ctrl = UART_HW_FLOWCTRL_DISABLE
    };
    uart_param_config(UART_NUM_1, &uart_config);
    uart_set_pin(UART_NUM_1, TXD_PIN, RXD_PIN, UART_PIN_NO_CHANGE, UART_PIN_NO_CHANGE);
    // We won't use a buffer for sending data.
    uart_driver_install(UART_NUM_1, RX_BUF_SIZE * 2, 0, 0, NULL, 0);
}

int sendData(const char* logName, const char* data){
    const int len = strlen(data);
    const int txBytes = uart_write_bytes(UART_NUM_1, data, len);
    ESP_LOGI(logName, "Wrote %d bytes:\t%s", txBytes, data);
    return txBytes;
}

/*Main Function*/
void app_main(){
/*Define Variables*/
    int status = 1, voltage, connected = 0, succes;
    struct LEDs LED_contoller;
    int no_change=1, tof=0, IMU=0, turn=0;

    /*Configure the button GPIO as input, enable wakeup */
    const int button_gpio_num = BUTTON_GPIO_NUM_DEFAULT;
    const int wakeup_level = BUTTON_WAKEUP_LEVEL_DEFAULT;
    gpio_config_t config = {
            .pin_bit_mask = BIT64(button_gpio_num),
            .mode = GPIO_MODE_INPUT
    };
    ESP_ERROR_CHECK(gpio_config(&config));
    gpio_wakeup_enable(button_gpio_num,
```

```
        wakeup_level == 0 ? GPIO_INTR_LOW_LEVEL : GPIO_INTR_HIGH_LEVEL);

    int ch;
    /*Prepare and set configuration of timers that will be used by LED Controller*/
      ledc_timer_config_t ledc_timer = {
      .duty_resolution = LEDC_TIMER_13_BIT, // resolution of PWM duty
      .freq_hz = 38000,                     // frequency of PWM signal
      .speed_mode = LEDC_HS_MODE,           // timer mode
      .timer_num = LEDC_HS_TIMER            // timer index
    };
    // Set configuration of timer0 for high speed channels
    ledc_timer_config(&ledc_timer);

    /*Prepare individual configuration for each channel of LED Controller by selecting:
     * - controller's channel number
     * - output duty cycle, set initially to 0
     * - GPIO number where LED is connected to
     * - speed mode, either high or low
     * - timer servicing selected channel
     *   Note: if different channels use one timer, then frequency and bit_num of these channels will be
the same*/
    ledc_channel_config_t ledc_channel[LEDC_TEST_CH_NUM] = {
      {
        .channel    = LEDC_HS_CH0_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_CH0_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
      },
      {
        .channel    = LEDC_HS_CH1_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_CH1_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
      },
      {
        .channel    = LEDC_HS_CH2_CHANNEL,
        .duty       = 0,
        .gpio_num   = LEDC_HS_CH2_GPIO,
        .speed_mode = LEDC_HS_MODE,
        .hpoint     = 0,
        .timer_sel  = LEDC_HS_TIMER
      },
    };
    // Set LED Controller with previously prepared configuration
```

```
    for (ch = 0; ch < LEDC_TEST_CH_NUM; ch++) {
        ledc_channel_config(&ledc_channel[ch]);
    }

        //Check if Two Point or Vref are burned into eFuse
    check_efuse();

    //Configure ADC
    if (unit == ADC_UNIT_1) {
        adc1_config_width(ADC_WIDTH_BIT_12);
        adc1_config_channel_atten(channel, atten);
    } else {
        adc2_config_channel_atten((adc2_channel_t)channel, atten);
    }

    //Characterize ADC
    adc_chars = calloc(1, sizeof(esp_adc_cal_characteristics_t));
    esp_adc_cal_value_t val_type = esp_adc_cal_characterize(unit, atten, ADC_WIDTH_BIT_12,
DEFAULT_VREF, adc_chars);
    print_char_val_type(val_type);

    printf("\t\tMOUSR SYSTEM INITIAL --> STATUS : POWER ON\n");
    LED_contoller.LED_number = 0;            /*LED 0*/
    LED_control_ON(ledc_channel, LED_contoller);

    esp_sleep_enable_gpio_wakeup();        /*Configuration WAKE_UP*/


    init();     /*Initialize UART*/
    static const char *TX_TASK_TAG = "TX_TASK";
    esp_log_level_set(TX_TASK_TAG, ESP_LOG_INFO);

    while(true) {
        if(connected){
            LED_contoller.LED_number = 0;
            LED_contoller.blink = 0;
            LED_control_ON(ledc_channel, LED_contoller);

            if(no_change){ /*No detection --> Move Forward*/
                printf("MOVE FORWARD\n");
                sendData(TX_TASK_TAG, "L+5");
                sendData(TX_TASK_TAG, "R+5");
                vTaskDelay(1000 / portTICK_PERIOD_MS);
                no_change = 0;
            }
            else{
                if(tof){
                    printf("TURN\n");
```

```
            if(turn){
                sendData(TX_TASK_TAG, "L+3");
                sendData(TX_TASK_TAG, "R+7");
                vTaskDelay(1000 / portTICK_PERIOD_MS);
                no_change = 1;
                turn = 0;
            }
            else{
                sendData(TX_TASK_TAG, "L+7");
                sendData(TX_TASK_TAG, "R+3");
                vTaskDelay(1000 / portTICK_PERIOD_MS);
                no_change = 1;
                turn = 1;
            }
        }
        else{
            if(IMU){
                printf("STOP MOTORS\n");
                sendData(TX_TASK_TAG, "L+0");
                sendData(TX_TASK_TAG, "R+0");
                vTaskDelay(5000 / portTICK_PERIOD_MS);
                no_change = 1;
            }
        }
    }
}
else{
    LED_contoller.LED_number = 0;
    LED_contoller.blink = 1;
    LED_control_ON(ledc_channel, LED_contoller);
}
if(rtc_gpio_get_level(button_gpio_num) == wakeup_level){
    /* Get timestamp before entering sleep */
    int64_t t_before_us = esp_timer_get_time();
    do {
            vTaskDelay(pdMS_TO_TICKS(10));
        }   while (rtc_gpio_get_level(button_gpio_num) == wakeup_level);
    /* Get timestamp after waking up from sleep */
    int64_t t_after_us = esp_timer_get_time();

    /*MODES*/
    printf("%lld\n", (t_after_us-t_before_us));

    if((t_after_us-t_before_us) >= 100000 && (t_after_us-t_before_us) < 1000000 && status){
        printf("\t\tUPDATE STATUS: MODE BATERRY STATUS\n");
        voltage = get_battery_level();
        LED_control_OFF(ledc_channel, LED_contoller);
```

```
            if(voltage <= 1100){
                printf("\t\tLOW BATTERY --> PLEASE CHARGE");
                LED_contoller.LED_number = 2;
                LED_contoller.blink = 1;
                LED_control_ON(ledc_channel, LED_contoller);
                vTaskDelay(pdMS_TO_TICKS(1000));
                LED_control_OFF(ledc_channel, LED_contoller);
            }
            else{
                if(voltage <= 2200 && voltage > 1100){
                    printf("\t\tMEDIUM BATTERY");
                        LED_contoller.blink = 1;
                        LED_contoller.LED_number = 2;
                        LED_control_ON(ledc_channel, LED_contoller);
                        LED_contoller.LED_number = 0;
                        LED_control_ON(ledc_channel, LED_contoller);
                        vTaskDelay(pdMS_TO_TICKS(1000));
                        LED_control_OFF(ledc_channel, LED_contoller);
                }
                else{
                    if(voltage > 2200){
                        printf("\t\tFULL BATTERY");
                        LED_contoller.LED_number = 0;
                        LED_contoller.blink = 1;
                        LED_control_ON(ledc_channel, LED_contoller);
                        vTaskDelay(pdMS_TO_TICKS(1000));
                        LED_control_OFF(ledc_channel, LED_contoller);
                    }
                }
            }
            printf("\nVoltage: %dmV\n", voltage);
        }
        else{
            if((t_after_us-t_before_us) >= 1000000 && (t_after_us-t_before_us)< 2500000){ /*PARING
MODE*/
                printf("\t\tUPDATE STATUS : PARING MODE\n");
                LED_control_OFF(ledc_channel, LED_contoller);
                LED_contoller.LED_number = 1;
                LED_contoller.blink = 1;
                LED_control_ON(ledc_channel, LED_contoller);
                succes = wifi_connection();
                if(succes){
                    printf("WIFI CONNECTED\n");
                    connected = 1;
                    LED_control_OFF(ledc_channel, LED_contoller);
                    LED_contoller.LED_number = 0;
                    LED_contoller.blink = 1;
                    LED_control_ON(ledc_channel, LED_contoller);
```

```
                vTaskDelay(pdMS_TO_TICKS(1000));
                LED_control_OFF(ledc_channel, LED_contoller);
            }
            else{
                printf("WIFI NO-CONNECTED\n");
                connected = 0;
                LED_control_OFF(ledc_channel, LED_contoller);
                LED_contoller.LED_number = 2;
                LED_contoller.blink = 1;
                LED_control_ON(ledc_channel, LED_contoller);
                vTaskDelay(pdMS_TO_TICKS(1000));
                LED_control_OFF(ledc_channel, LED_contoller);
            }
        }
        else{
            if((t_after_us-t_before_us) >=2500000 || status){
                if(status){
                    LED_control_OFF(ledc_channel, LED_contoller);
                    LED_contoller.LED_number = 2;
                    LED_control_ON(ledc_channel, LED_contoller);
                    vTaskDelay(1000 / portTICK_PERIOD_MS);
                    LED_control_OFF(ledc_channel, LED_contoller);
/*Delay 1 second*/
                    printf("\t\tUPDATE STATUS : POWER OFF\n");
                    status = 0;
                /* Enter sleep mode */
                    uart_tx_wait_idle(CONFIG_CONSOLE_UART_NUM);
                    esp_light_sleep_start();
                    printf("\t\tUPDATE STATUS : POWER ON\n");
                    status = 1;
                }
                else{
                    printf("\t\tUPDATE STATUS : POWER ON\n");
                    status = 1;
            }
        }
    }
  }
 }
}
}
```

Appendix C. Phone Application Code

Main Code:

```
package com.example.mousrapp;

import android.content.Intent;
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageButton;
import android.widget.Toast;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.ServerSocket;
import java.net.Socket;

public class MainActivity extends AppCompatActivity {

    private static Socket sock;
    private static ServerSocket serversock;
    private static PrintWriter printWriter;
    private static final String SERVER_IP = "170.20.10.9"; //ESP IP address
    public static final int SERVER_PORT = 1234;
    TcpClient client;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        new ConnectTask().execute("");

        /*
        array for the return messages
        0: color
```

```
1: Day
2: Hour
3: Minute
4: AM/PM
5: Connect Messages
*/
String[] messages = new String[6];

//get returned messages
Bundle extras = getIntent().getExtras();
if (extras != null) {
    for (String key : extras.keySet()) {
        switch(key){
            case "color":
                messages[0] = extras.getString(key);
                break;
            case "Day":
                messages[1] = extras.getString(key);
                break;
            case "Hour":
                messages[2] = extras.getString(key);
                break;
            case "Minute":
                messages[3] = extras.getString(key);
                break;
            case "ampm":
                messages[4] = extras.getString(key);
                break;
            case "connect":
                messages[5] = extras.getString(key);
            default:
                break;
        }
    }
}

// send messages to ESP
for( int key=0; key<messages.length; key++){
    if(messages[key] != null) {
        Log.d("MainActivity","making sure IP is connected");
        Log.d("MainActivity", "Sending message");
        client.sendMessage(messages[key]);
    }
}
```

```
}

/** Called when the user taps Connection Button */
public void setWiFi (View view){
   ImageButton btn = (ImageButton) findViewById(R.id.connection_button);

   btn.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View v) {

         //Launch WiFi Connection Setup
         Intent intent = new Intent(MainActivity.this, connectActivity.class);
         startActivity(intent);
         //startActivityForResult(intent,REQUEST_CODE_WIFI);
      }
   });
}

/** called when the user taps Set Schedule Button */
public void setSchedule(View view){
   Button btn = (Button) findViewById(R.id.sched_button);

   btn.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View v) {

         //Launch WiFi Connection Setup
         Intent intent = new Intent(MainActivity.this, SetScheduleActivity.class);
         startActivity(intent);
         //startActivityForResult(intent,REQUEST_CODE_SCHED);
      }
   });
}

/** called when the user taps Set LED Button */
public void setLED(View view){
   Button btn = (Button) findViewById(R.id.led_button);

   btn.setOnClickListener(new View.OnClickListener(){
      @Override
      public void onClick(View v) {

         //Launch WiFi Connection Setup
         Intent intent = new Intent(MainActivity.this, SetLEDActivity.class);
```

```
            startActivity(intent);
        }
    });
}

/** called when the user taps Autoplay Button */
//public void autoPlay(){
//    Button btn = (Button) findViewById(R.id.autoplay_button);

//    btn.setOnClickListener(new View.OnClickListener(){
//        @Override
//        public void onClick(View v) {

//            //Launch WiFi Connection Setup
//            //Intent intent = AutoActivity.makeIntent(MainActivity.this);
//           // startActivityForResult(intent,REQUEST_CODE_AUTO);
//        }
//    });
//}

private void showToast(String msg){
    Toast.makeText(this,msg,Toast.LENGTH_SHORT).show();
}

public class ConnectTask extends AsyncTask<String, String, TcpClient> {

    @Override
    protected TcpClient doInBackground(String... message) {

        //we create a TCPClient object
        client = new TcpClient(new TcpClient.OnMessageReceived() {
            @Override
            //here the messageReceived method is implemented
            public void messageReceived(String message) {
                //this method calls the onProgressUpdate
                publishProgress(message);
            }
        });
        client.run();

        return null;
    }

    @Override
```

```
        protected void onProgressUpdate(String... values) {
            super.onProgressUpdate(values);
            //response received from server
            Log.d("test", "response " + values[0]);
            //process server response here....

        }
    }
}
```

Connect Activity:

```
package com.example.mousrapp;

import android.Manifest;
import android.app.Activity;
import android.bluetooth.BluetoothAdapter;
import android.bluetooth.BluetoothDevice;
import android.bluetooth.BluetoothManager;
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
import android.content.IntentFilter;
import android.os.Build;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.ListView;
import android.widget.Toast;

import java.nio.charset.Charset;
import java.util.ArrayList;
import java.util.List;
import java.util.Set;
import java.util.UUID;

public class connectActivity extends AppCompatActivity implements
AdapterView.OnItemClickListener{
```

```
    private static final String TAG = "connectActivity";

    BluetoothAdapter mBluetoothAdapter;
    Button sendbutton;
    EditText editpassword;

    public ArrayList<BluetoothDevice> mDevices = new ArrayList<>();
    public DeviceListAdapter mDeviceListAdapter;
    ListView newDevices;

    private static final UUID MY_UUID_INSECURE = UUID.fromString("8ce255c0-200a-11e0-ac64-
0800200c9a66");
    //BluetoothConnectionService mBluetoothConnection;
    BluetoothDevice mDevice;

    //String ssid = getWiFiName(this);

    private final BroadcastReceiver enableReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            if(action.equals(mBluetoothAdapter.ACTION_STATE_CHANGED)){
                final int state =
intent.getIntExtra(BluetoothAdapter.EXTRA_STATE,mBluetoothAdapter.ERROR);

                switch(state){
                    case BluetoothAdapter.STATE_OFF:
                        Log.d(TAG, " OnReceive: STATE OFF");
                        break;
                    case BluetoothAdapter.STATE_TURNING_OFF:
                        Log.d(TAG, " OnReceive: STATE TURNING OFF");
                        break;
                    case BluetoothAdapter.STATE_ON:
                        Log.d(TAG, " OnReceive: STATE ON");
                        break;
                    case BluetoothAdapter.STATE_TURNING_ON:
                        Log.d(TAG, " OnReceive: STATE TURNING ON");
                        break;
                }
            }
        }
    };
```

```
    private final BroadcastReceiver discoverReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();

            if(action.equals(BluetoothAdapter.ACTION_SCAN_MODE_CHANGED)){
                final int mode =
intent.getIntExtra(BluetoothAdapter.EXTRA_SCAN_MODE,BluetoothAdapter.ERROR);

                switch(mode){
                    case BluetoothAdapter.SCAN_MODE_CONNECTABLE_DISCOVERABLE:
                        Log.d(TAG, " discoverReceiver: Discoverability Enabled");
                        break;
                    case BluetoothAdapter.SCAN_MODE_CONNECTABLE:
                        Log.d(TAG, " discoverReceiver: Discoverability Disabled. Able to receive connections");
                        break;
                    case BluetoothAdapter.SCAN_MODE_NONE:
                        Log.d(TAG, " discoverReceiver: Discoverability Disabled. Not able to receive
connections");
                        break;
                    case BluetoothAdapter.STATE_CONNECTING:
                        Log.d(TAG, " discoverReceiver: Connecting...");
                        break;
                    case BluetoothAdapter.STATE_CONNECTED:
                        Log.d(TAG, " discoverReceiver: Connected");
                        break;
                }
            }
        }
    };

    private final BroadcastReceiver discoveryReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            String action = intent.getAction();

            if(action.equals(BluetoothDevice.ACTION_FOUND)){
                BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                mDevices.add(device);
                Log.d(TAG, "OnReceive: " + device.getName() + ": " + device.getAddress());

                mDeviceListAdapter = new DeviceListAdapter(context, R.layout.device_adapter_view,
mDevices);
                newDevices.setAdapter(mDeviceListAdapter);
```

```
        }
    }
};

private final BroadcastReceiver bondReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        final String action = intent.getAction();

        if(action.equals(BluetoothDevice.ACTION_BOND_STATE_CHANGED)){
            BluetoothDevice device = intent.getParcelableExtra((BluetoothDevice.EXTRA_DEVICE));

            //device is already bonded
            if(device.getBondState() == BluetoothDevice.BOND_BONDED){
                Log.d(TAG, "bondReceiver: BOND_BONDED");
            }

            //device is bonding
            if(device.getBondState() == BluetoothDevice.BOND_BONDING){
                Log.d(TAG, "bondReceiver: BOND_BONDING");
            }

            //device is breaking bond
            if(device.getBondState() == BluetoothDevice.BOND_NONE){
                Log.d(TAG, "bondReceiver: BOND_NONE");
            }
        }
    }
};

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_connect);

    mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
    newDevices = (ListView) findViewById(R.id.deviceLv);
    mDevices = new ArrayList<>();

    sendbutton = (Button) findViewById(R.id.passwordbutton);
    editpassword = (EditText) findViewById(R.id.editpassword);

    final Intent ret = new Intent(connectActivity.this, MainActivity.class);
```

```
        //check if device has Bluetooth capabilities
        if(mBluetoothAdapter == null){
            ret.putExtra("connect","Device does not have Bluetooth");
            startActivity(ret);
        }

        //make sure device is Bluetooth enabled
        if(!mBluetoothAdapter.isEnabled()){
            Intent enable = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivity(enable);

            IntentFilter enableIntent = new IntentFilter(BluetoothAdapter.ACTION_STATE_CHANGED);
            registerReceiver(enableReceiver, enableIntent);
        }

        //make device discoverable
        Intent discoverable = new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
        discoverable.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,300);
        startActivity(discoverable);

        IntentFilter discoverableIntent = new
IntentFilter(mBluetoothAdapter.ACTION_SCAN_MODE_CHANGED);
        registerReceiver(discoverReceiver,discoverableIntent);

        //begin scanning for devices
        if(mBluetoothAdapter.isDiscovering()){
            mBluetoothAdapter.cancelDiscovery();
        }

        mBluetoothAdapter.startDiscovery();
        IntentFilter discoveryIntent = new IntentFilter(BluetoothDevice.ACTION_FOUND);
        registerReceiver(discoveryReceiver, discoveryIntent);

        //pairing
        newDevices.setOnItemClickListener(this);

        sendbutton.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {

                //get WiFi SSID and password
                String password = editpassword.getText().toString();

                //convert to byte packets
```

```
            //byte[] ssidbytes = ssid.getBytes(Charset.defaultCharset());
            byte[] passbytes = password.getBytes(Charset.defaultCharset());

            //send values
            //mBluetoothConnection.write(passbytes);

            startActivity(ret);
        }
    });
}

@Override
public void onItemClick(AdapterView<?> adapterView, View view, int i, long l){

    //cancel discovery
    mBluetoothAdapter.cancelDiscovery();

    Log.d(TAG, "onItemClick: device clicked");
    String deviceName = mDevices.get(i).getName();
    String deviceAddress = mDevices.get(i).getAddress();

    Log.d(TAG, "onItemClick: deviceName = " + deviceName);
    Log.d(TAG, "onItemClick: deviceAddress = " + deviceAddress);

    //pair device
    if(Build.VERSION.SDK_INT > Build.VERSION_CODES.JELLY_BEAN_MR2){
        Log.d(TAG, "Trying to pair with " + deviceName);

        mDevice = mDevices.get(i);
        mDevice.createBond();

        /*
        //start bluetooth communication thread
        mBluetoothConnection = new BluetoothConnectionService(this);
        startConnection(mDevice,MY_UUID_INSECURE);

        //get password with alert dialog





        //send over wiFi information
```

```
            //bytes = password.getBytes(Charset.defaultCharset());
            //mBluetoothConnection.write(bytes);
            */
        }
    }


    //starts communication thread to send WiFi credentials over
    public void startConnection(BluetoothDevice device, UUID uuid){
        Log.d(TAG,"startConnection: Initialize RFCOM Bluetooth Connection");
        //mBluetoothConnection.startClient(device,uuid);
    }

}
```

Device List Adapter:

```
package com.example.mousrapp;

import android.bluetooth.BluetoothDevice;
import android.content.Context;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ArrayAdapter;
import android.widget.TextView;

import org.w3c.dom.Text;

import java.util.ArrayList;

public class DeviceListAdapter extends ArrayAdapter<BluetoothDevice> {

    private LayoutInflater mLayoutInflater;
    private ArrayList<BluetoothDevice> mDevices;
    private int mViewResourceId;

    public DeviceListAdapter(Context context, int tvResouceId, ArrayList<BluetoothDevice> devices){
        super(context,tvResouceId,devices);
        this.mDevices = devices;
        mLayoutInflater = (LayoutInflater)
context.getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        mViewResourceId = tvResouceId;
    }
```

```
    public View getView(int position, View convertView, ViewGroup parent){
        convertView=mLayoutInflater.inflate(mViewResourceId, null);

        BluetoothDevice device = mDevices.get(position);

        if(device != null){
            TextView deviceName = (TextView) convertView.findViewById(R.id.deviceNameTv);
            TextView deviceAddress = (TextView) convertView.findViewById(R.id.deviceAddressTv);

            if(deviceName != null){
                deviceName.setText(device.getName());
            }
            if(deviceAddress != null){
                deviceAddress.setText(device.getAddress());
            }
        }

        return convertView;
    }
}
```

Set LED Activity:

```
package com.example.mousrapp;

import android.content.Context;
import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.TextView;
import android.widget.Toast;

public class SetLEDActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_set_led);
```

```
        //create list of colors
        String[] myColors = {"Red", "Green", "Blue", "Yellow", "Purple", "White"};

        //Build Adapter
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, myColors);

        //Configure ListView
        ListView colorlist = findViewById(R.id.colorsLv);
        colorlist.setAdapter(adapter);


        colorlist.setOnItemClickListener(
                new AdapterView.OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {

                        Intent ret = new Intent(SetLEDActivity.this, MainActivity.class);

                        String color = String.valueOf(parent.getItemAtPosition(position));

                        ret.putExtra("color",color);

                        startActivity(ret);

                    }
                }
        );
    }
}
```

Set Schedule Activity:

```
package com.example.mousrapp;

import android.content.Intent;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.ListView;
```

```
public class SetScheduleActivity extends AppCompatActivity {

    String day,hour,minute,ampm;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_set_schedule);

        //create lists of Days, Hours, Minutes,and AM/PM
        String[] myDays = {"Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday",
"Saturday"};
        String[] myHours = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12"};
        String[] myMinutes = {"00", "05", "10", "15", "20", "25", "30", "35", "40", "45", "50", "55"};
        String[] myAMPM = {"AM", "PM"};

        //Build Adapters
        ArrayAdapter<String> day_adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, myDays);
        ArrayAdapter<String> hours_adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, myHours);
        ArrayAdapter<String> minutes_adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, myMinutes);
        ArrayAdapter<String> ampm_adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, myAMPM);

        //Configure ListViews
        ListView day_list = findViewById(R.id.DayList);
        ListView hour_list = findViewById(R.id.HourList);
        ListView minute_list = findViewById(R.id.MinuteList);
        ListView ampm_list = findViewById(R.id.AMPMList);
        day_list.setAdapter(day_adapter);
        hour_list.setAdapter(hours_adapter);
        minute_list.setAdapter(minutes_adapter);
        ampm_list.setAdapter(ampm_adapter);

        day_list.setOnItemClickListener(
                new AdapterView.OnItemClickListener() {
                    @Override
                    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                        day = String.valueOf(parent.getItemAtPosition(position));
                    }
                }
        );
```

```
        hour_list.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                    hour = String.valueOf(parent.getItemAtPosition(position));
                }
            }
        );

        minute_list.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                    minute = String.valueOf(parent.getItemAtPosition(position));
                }
            }
        );

        ampm_list.setOnItemClickListener(
            new AdapterView.OnItemClickListener() {
                @Override
                public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
                    ampm = String.valueOf(parent.getItemAtPosition(position));
                }
            }
        );

        final Button set = findViewById(R.id.sched_button);
        set.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent ret = new Intent(SetScheduleActivity.this, MainActivity.class);

                ret.putExtra("Day",day);
                ret.putExtra("Hour",hour);
                ret.putExtra("Minute",minute);
                ret.putExtra("ampm",ampm);

                startActivity(ret);
            }
        });
    }
}
```

Top Client:

```
package com.example.mousrapp;

import android.util.Log;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.InputStreamReader;
import java.io.OutputStreamWriter;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;

public class TcpClient {

    public static final String SERVER_IP = "192.168.0.100"; //your computer IP address
    public static final int SERVER_PORT = 4444;
    // message to send to the server
    private String mServerMessage;
    // sends message received notifications
    private OnMessageReceived mMessageListener = null;
    // while this is true, the server will continue running
    private boolean mRun = false;
    // used to send messages
    private PrintWriter mBufferOut;
    // used to read messages from the server
    private BufferedReader mBufferIn;

    /**
     * Constructor of the class. OnMessagedReceived listens for the messages received from server
     */
    public TcpClient(OnMessageReceived listener) {
        mMessageListener = listener;
    }

    /**
     * Sends the message entered by client to the server
     *
     * @param message text entered by client
     */
    public void sendMessage(String message) {
        if (mBufferOut != null && !mBufferOut.checkError()) {
            mBufferOut.println(message);
```

```
            mBufferOut.flush();
        }
    }

    /**
     * Close the connection and release the members
     */
    public void stopClient() {

        // send mesage that we are closing the connection
        sendMessage(Constants.CLOSED_CONNECTION+"Kazy");

        mRun = false;

        if (mBufferOut != null) {
            mBufferOut.flush();
            mBufferOut.close();
        }

        mMessageListener = null;
        mBufferIn = null;
        mBufferOut = null;
        mServerMessage = null;
    }

    public void run() {

        mRun = true;

        try {
            //here you must put your computer's IP address.
            InetAddress serverAddr = InetAddress.getByName(SERVER_IP);

            Log.e("TCP Client", "C: Connecting...");

            //create a socket to make the connection with the server
            Socket socket = new Socket(serverAddr, SERVER_PORT);

            try {

                //sends the message to the server
                mBufferOut = new PrintWriter(new BufferedWriter(new
OutputStreamWriter(socket.getOutputStream())), true);
```

```
            //receives the message which the server sends back
            mBufferIn = new BufferedReader(new InputStreamReader(socket.getInputStream()));
            // send login name
            sendMessage(Constants.LOGIN_NAME+"Kazy");

            //in this while the client listens for the messages sent by the server
            while (mRun) {

                mServerMessage = mBufferIn.readLine();

                if (mServerMessage != null && mMessageListener != null) {
                    //call the method messageReceived from MyActivity class
                    mMessageListener.messageReceived(mServerMessage);
                }

            }

            Log.e("RESPONSE FROM SERVER", "S: Received Message: '" + mServerMessage + "'");

        } catch (Exception e) {

            Log.e("TCP", "S: Error", e);

        } finally {
            //the socket must be closed. It is not possible to reconnect to this socket
            // after it is closed, which means a new socket instance has to be created.
            socket.close();
        }

    } catch (Exception e) {

        Log.e("TCP", "C: Error", e);

    }

}

//Declare the interface. The method messageReceived(String message) will must be implemented in the
MyActivity
//class at on asynckTask doInBackground
public interface OnMessageReceived {
    public void messageReceived(String message);
}
}
```