University of Messina

Department of Engineering

Master’s Degree Course in Engineering and Computer Science

Embedded Systems Project:

Alarm system

Student :

Mariano Di Nuzzo

Teacher :

Prof. Francesco Longo

Academic Year 2019/2020

Contents

Chapter 1
Introduction

The goal of this project is the understanding of the various base functionality of embedded systems. In particular, this project is realized thanks to using of a series of components (sensors), of a code of programming and one Arduino board, that in this work can be define as “the heart of system". Arduino, moreover, is a programmable microcontroller capable of “execute and make execute” some “commands” to the other components in the system.

This work perfectly simulates the operation of a programmable alarm system, i.e. it simulates obstacle detection within a certain range.

In order to provide concrete functionality, the system requires some components as keys and sensors to collect the various inputs and an "Arduino" processing structure, sensors and motors to produce an output.

The next chapters explain in detail the notions of how to do this project, the Chapter 2 describes the list of materials used and the circuit diagrams, the Chapter 3 provides the functioning of this project for the users and the source code, finally, the last Chapter gives some conclusions and future implementations.

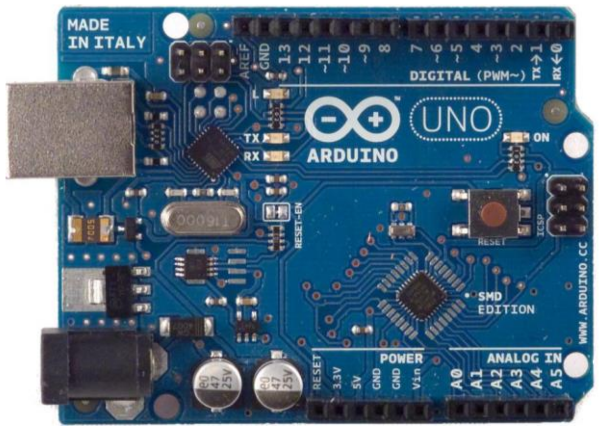


Fig.1 Arduino board

Chapter 2

2.1 Hardware aspects

2.1.1 DISPLAY LDC 20x4

Quimat display LCD 20X4

The Quimat display LCD II C/ I2C/ TWI is a module-display LCD 2004/20x4 (4 rows with 20 characters), it is essential for the user to interact with the configurations of the project and to view the various output.

The module is powered to 5 V and it is also equipped of a driver with contrast regulator via a potentiometer (it is adjustable by a simple Italian screwdriver), all optimized to have better results both from a circuit point of view (in fact, only 4 pins are needed), that from a functional point of view (the visualization is decidedly clearer).

Its configuration on the 4 pins can be summarized as: SDA, SCL, VCC, GND, and its dimensions are approximately: 60mmx99mm.

In order to use the display, it is necessary to import its LiquidCrystal_I2C.h library and to set appropriately in the code: LiquidCrystal_I2C lcd (0x27, 20, 4).

Finally, all its configurations and uses are similar to the 20x2 version configurations.

2.1.2 PASSIVE BUZZER MODULE

Buzzer

Module used to manage and emit the acoustic signal, this integrated module has 3 pins to be configured as follows:

VCC: 3.3V-5V.
GND: the ground.
I / O: Interface I / O di SCM.

It is possible to control the sound frequency and the duration of the sound directly from the software through the appropriate function:

```
tone(10, 2000, 300);
```

it is therefore possible even to emulate the behavior of musical notes by appropriately setting the configurations for this module.

2.1.3 SERVOMOTOR SG90

Servomotor SG90

The servomotor is a device capable of performing mechanical movements according to the signal applied to its input.

Basically, a servo is composed of an electric motor, a gearmotor and a feedback circuit for managing the position.

On the market, there is a wide choice of servos, each characterized by torque value and precision.

So the common characteristics in the servos are mainly: the rotation angle, the rotation speed and the driving torque.

Common servomotors generally have a rotation angle of about 180°, however there are other types of servos that have continuous rotation, so instead of piloting their position, we can manage their speed.

For any servo chosen, there are always 3 contacts: two are for power (5V and GND) while the third is the control pin to be connected to the Arduino pin.

The easiest way to drive the servomotor is to use the appropriate library, which translates angles into signals; the library in question is the Servo.

Within this library it is possible find some methods that greatly simplify the objective, such as:

attach(): allows to specify to which pin servo is connected and it allows to tie it to the Servo object;

attached(): check that a Servo type object is connected to a pin;

detach(): removes the connection between the Servo object and the pin to which it was tied;

read(): reads the angular position of servo, returns the last value passed with **write();**

write(): gives the servo the angle to position itself, on continuous rotation servo sets the rotation speed where 0 is the maximum speed in one direction, 90 when is stopped and 180 indicates the maximum speed in the reverse direction;

writeMicroseconds(): sets the rotation speed of the servo, in a standard servo the value goes from 1000 to 2000, in a continuous rotation servo it behaves in the same way as the **write().**

The servomotor was used in the project to provide the necessary rotation at the base for the HC-SR sensor in this way it has a 180° recognition of the surface (rather than in a single direction).

2.1.4 ULTRASONIC SENSOR HC-SR04

Ultrasonic sensor HC-SR04

The ultrasonic sensor HC-SR04, is an economic device and with a good operating range, however this sensor does not directly provide the measurement of the distance of the closest object, but measures the time taken by a sound signal to reach a certain obstacle and return to the sensor (the operation is similar to that of the sonar).

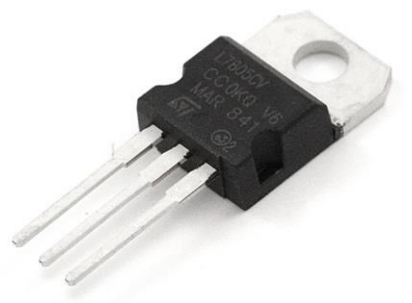
A high pulse is sent on the Trigger pin for at least 10 microseconds, at this point the sensor will send the sound ping and it'll wait for the return of the reflected waves, the sensor will respond on the Echo pin with a high pulse of the duration corresponding to that of travel of the sound waves, after 38 milliseconds of inactivity, it considers that no obstacles have been encountered.

Typically, the ultrasonic sensor HC-SR04 expects 50-60 milliseconds to ensure that there is no interference with the next measurement.

The ultrasound pulse sent by the HC-SR04 is about 40KHz and the time is measured in microseconds, the operating voltage as for many components is 5V, so it is possible to power it directly from Arduino.

The HC-SR04 sensor has 4 pins: Vcc (+ 5V), Trigger, Echo, GND.

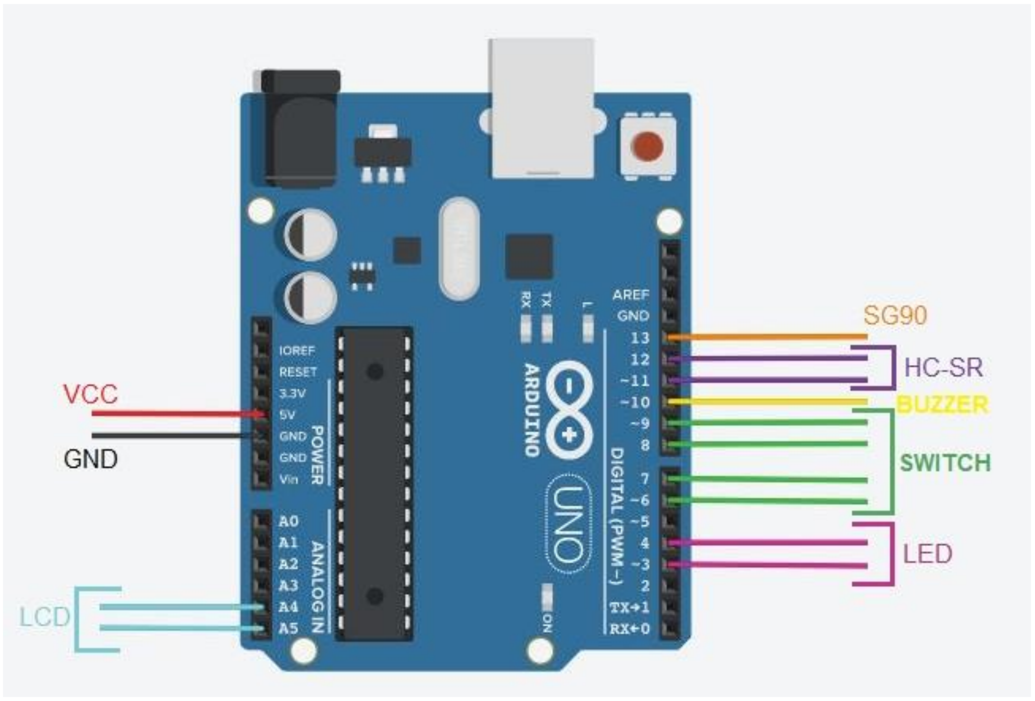
2.1.5 <u>OTHER COMPONENTS USED:</u>



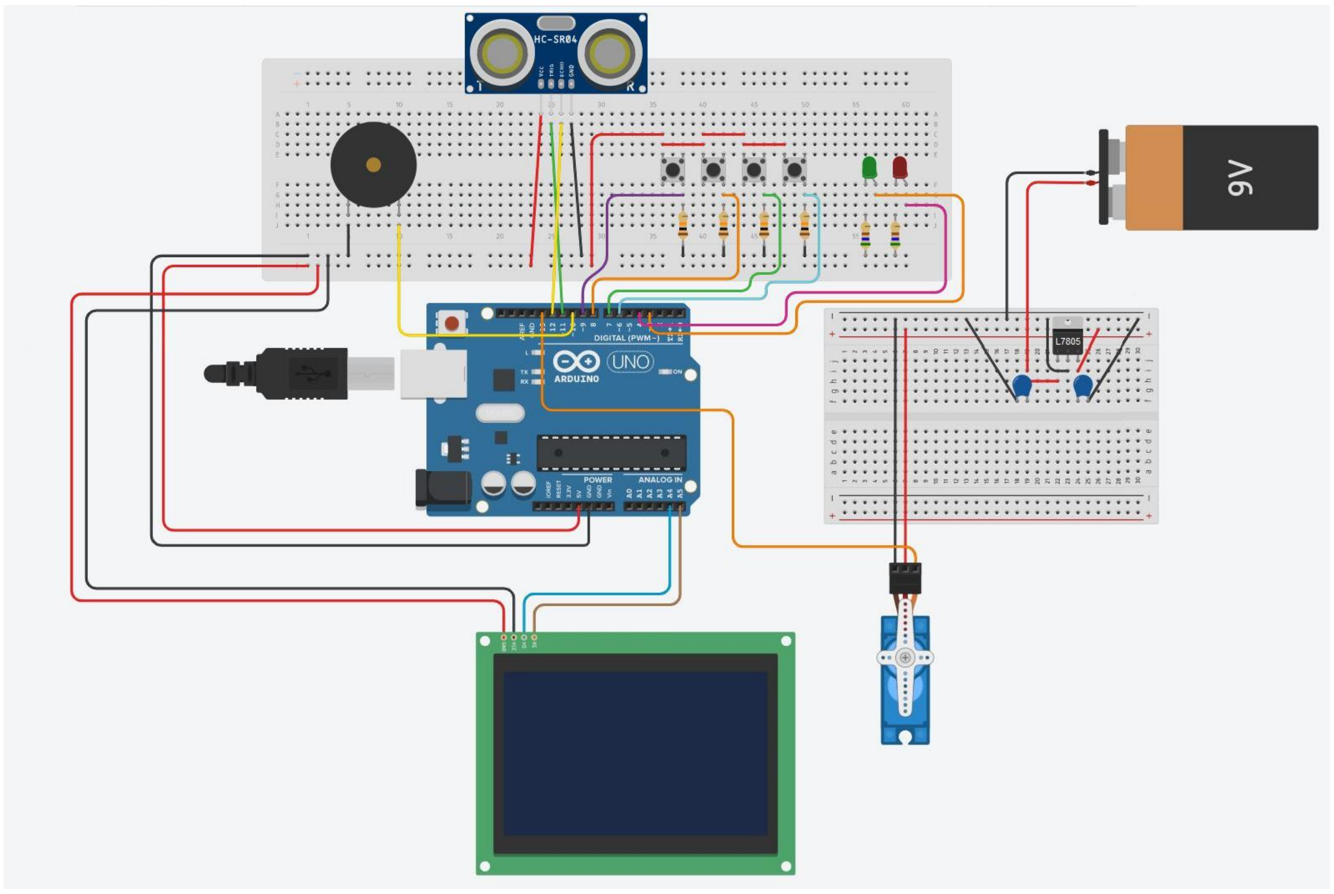
Regulator of 5V

- 2 capacitors: $22\ \mu F$ and $30\ \mu F$ respectively; they together with a regulator of 5V are essential to perform the circuit with external power source.
- 1 regulator of 5V: it has a linear potential difference of 5V, 1.5 A and 2% of tolerance.
- 4 buttons: they are tactile switches of 6 mm, with equivalent action click to a switch without contacts.
- 4 resistances of 10 KΩ.
- 2 resistance of 560 Ω.
- 2 led diodes.
- 2 adapters for cells of 9 V.
- 2 cells of 9 V, 1 A.
- Some jumpers: cables employed to the connections and bridges.

2.2 Hardware connections



Connections of pin on Arduino Board



Connections of various components on alarm system

Chapter 3

3.1 Functioning description

This work based on Arduino Uno microcontroller, consists of a programmable alarm system.

The user has the option of setting a timer via a push-button panel which starts the sensor-based recognition system when the countdown has ended.

More generally, the embedded device recognizes an object within a 180 ° angle and a maximum length of 10 cm, therefore two cases can be identified:

First case: the sensor recognizes an object that falls within the predetermined range (called the **red zone**)

Second case: the sensor does not recognize anything, since the object is out of range (called the **white zone**)

in the first case the device will reply:

- by emitting an acoustic signal
- illuminating a red led
- and showing the red zone on the display

in the second case, on the other hand, it will simply report:

- the distance of the first recognized object, via output on the display
- the white area written on the display
- lighting with green led

in general, its operation can be summarized in 3 phases:

1. setting the countdown time via 4 buttons (Phase 1)
2. waiting time (Phase 2)
3. start of operation (Phase 3)

Thus, one of the most important aspects in this project is the current mode, this alarm system can only work in one mode at a time:

• IDLE: the system is waiting for input, it shows the currently set amount of time; this is also the initial mode after power on or reset.

• SETUP - you can access this mode by long pressing the reset button; in this setting you can use the start-stop button to choose which time value to modify.

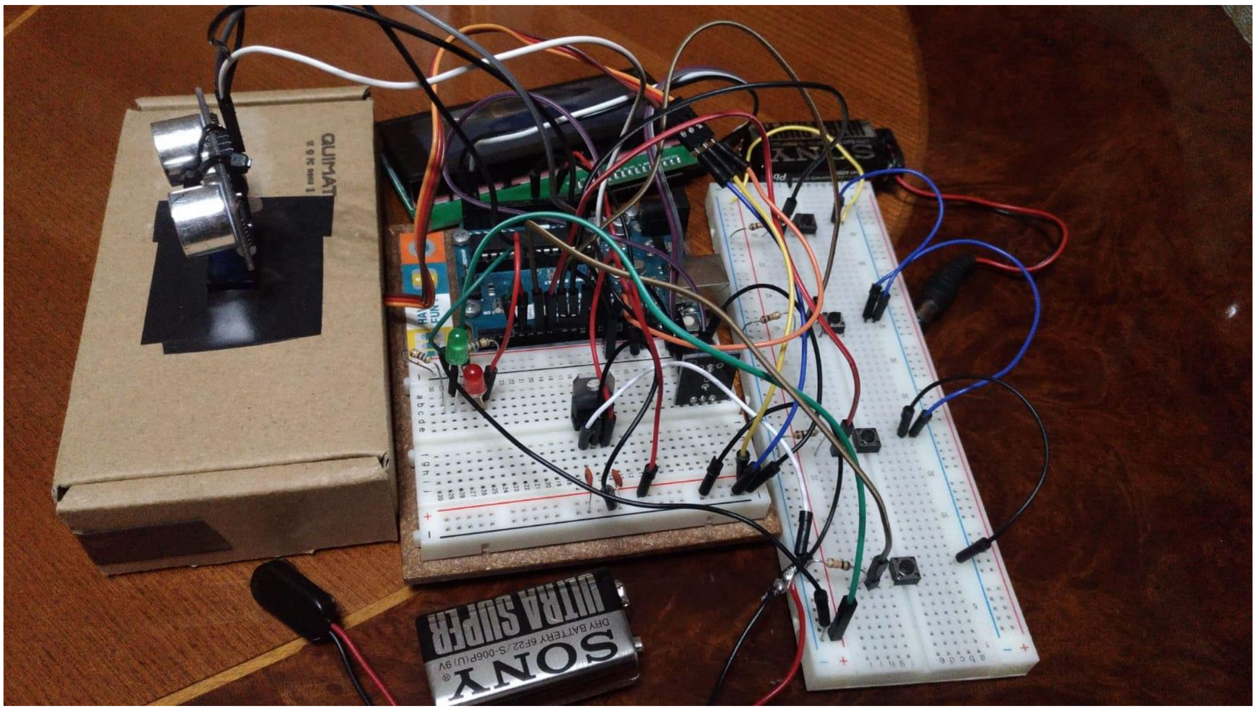
• RUNNING - in this mode it is possible to enter by pressing the start-stop button; it identifies the waiting state before the effective functioning of the alarm system.

• RINGING - this mode is activated automatically after the desired time; it represents the real functionality of the system.

The change of mode therefore identifies the following execution flow:

Step 1

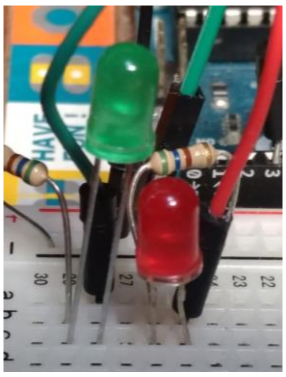
Initially the project looks like in figure (1).

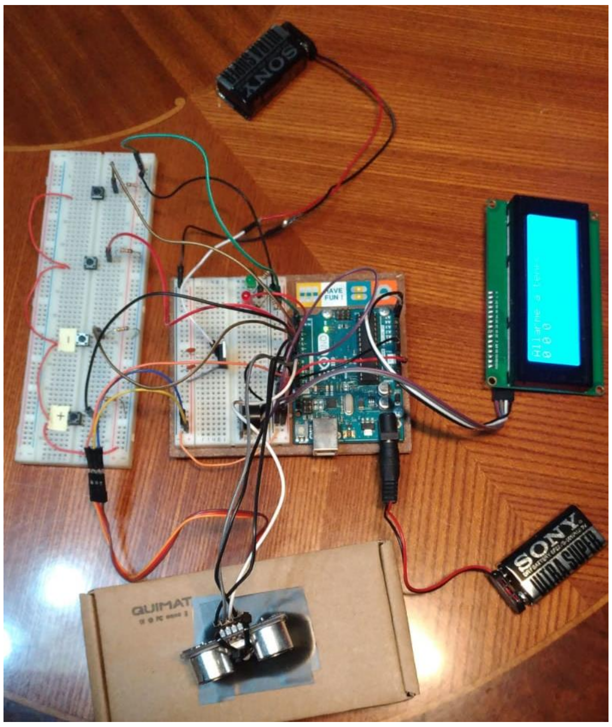


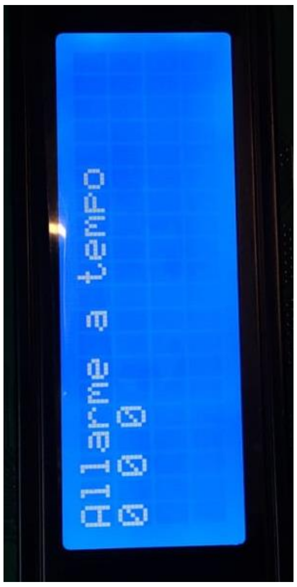
(1) Vision of the assembled project

Step 2

After supplying power to the Arduino board and the servo-motor respectively, the various parts of the system are shown in figure (2), in particular, in figure (3) is showed led off and in figure (4) the display in this condition.







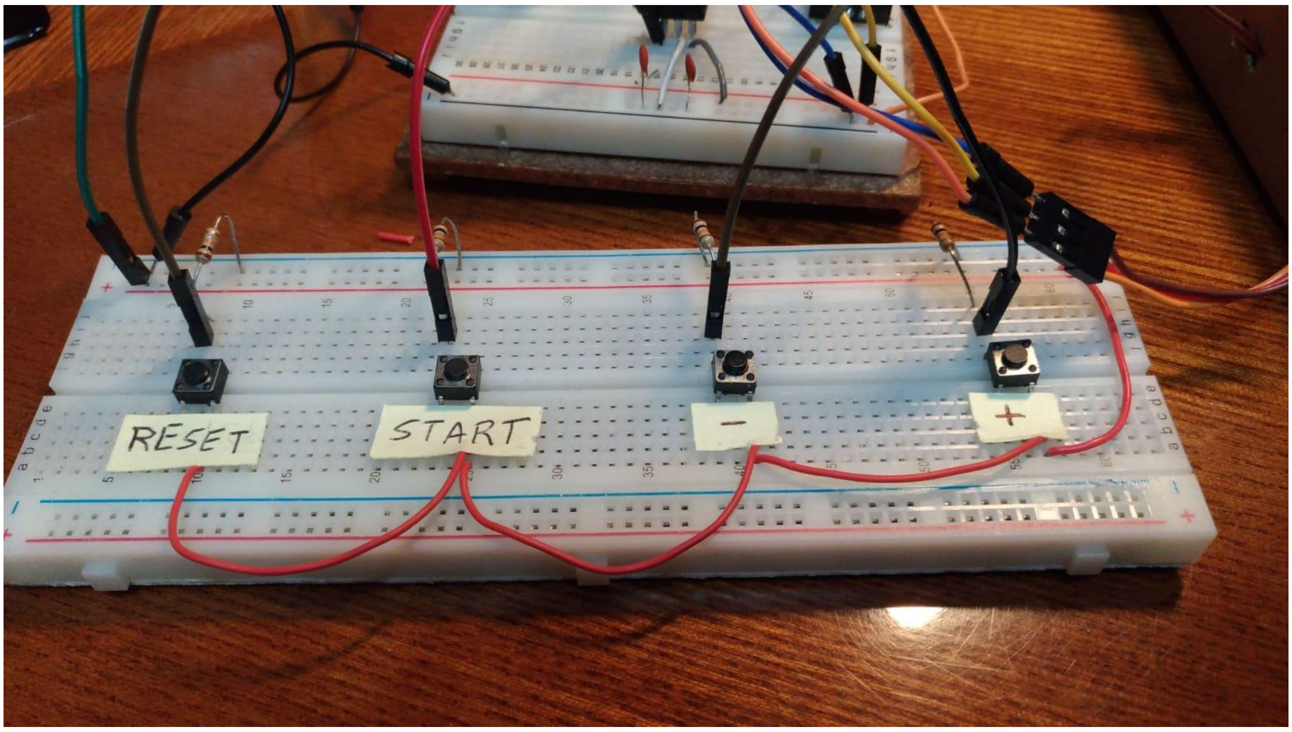
(2) Led off

(2)Initial state of the system

(4) Initial user interface

Step 3

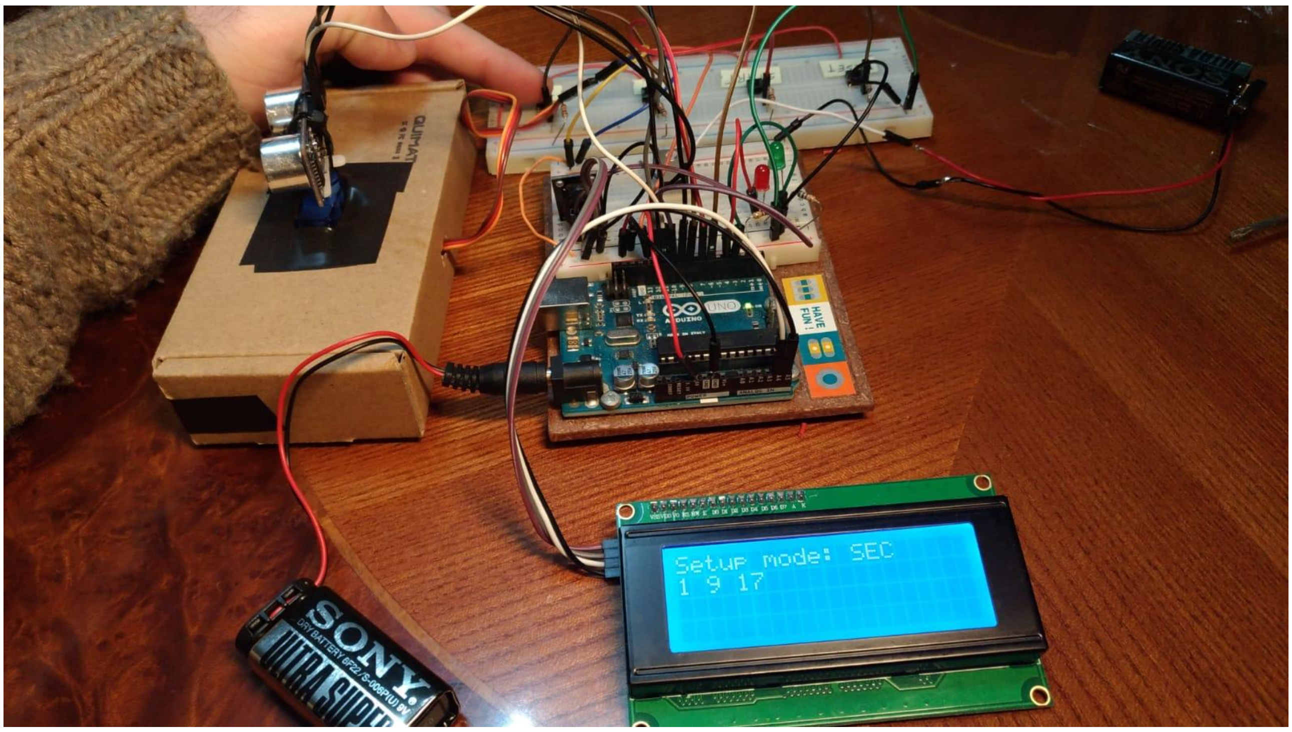
It is now possible to interface with the push-button panel (figure (5)), to provide the settings necessary to enter the following modes.



(5) Push-button panel

Step 4

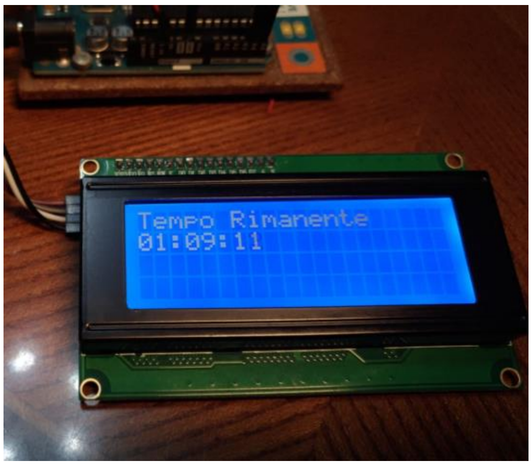
Using the four keys, in this phase it is possible to set the time required to activate the system (figure (6))

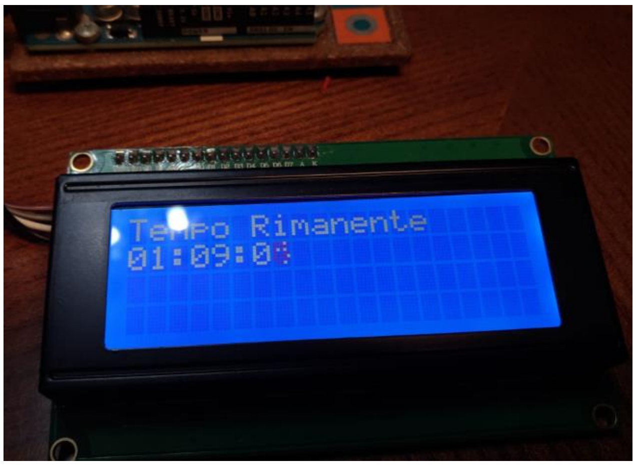


(6) system configuration, consequently switching from different modes such as setting of hour min and sec, finally starting the timer.

Step 5

Once the countdown has started, the display is in the condition as in figure (7), decreasing each time the variables set for hours, min and sec, figure (8).

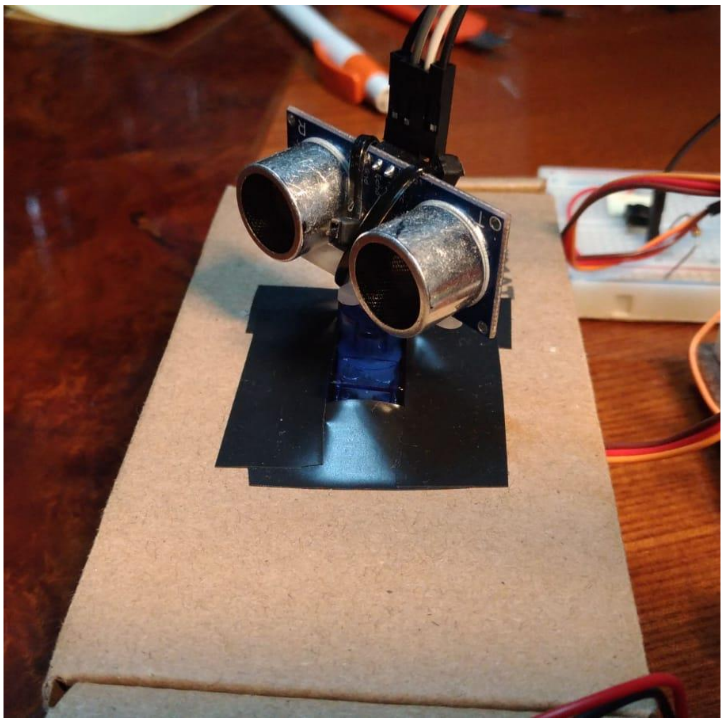




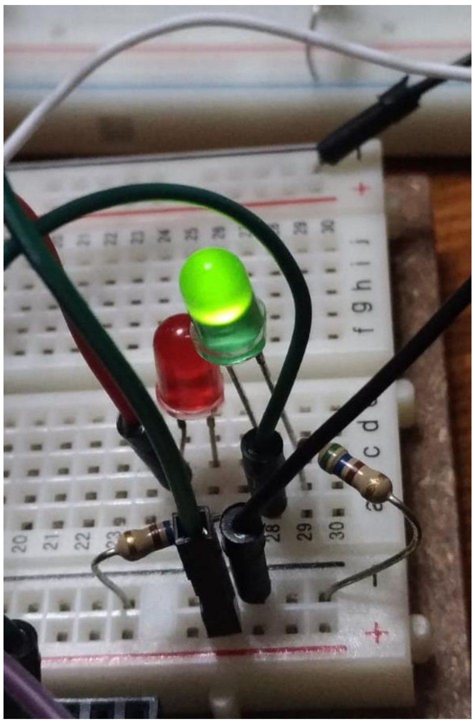
(7) Timer (8)

Step 6

The countdown has ended: the servomotor starts its run (figure (9)), the hc-sr sensor recognizes (or not) the obstacles and one of the two LEDs lights up (figure (10)) according to the presence or absence of the obstacle.



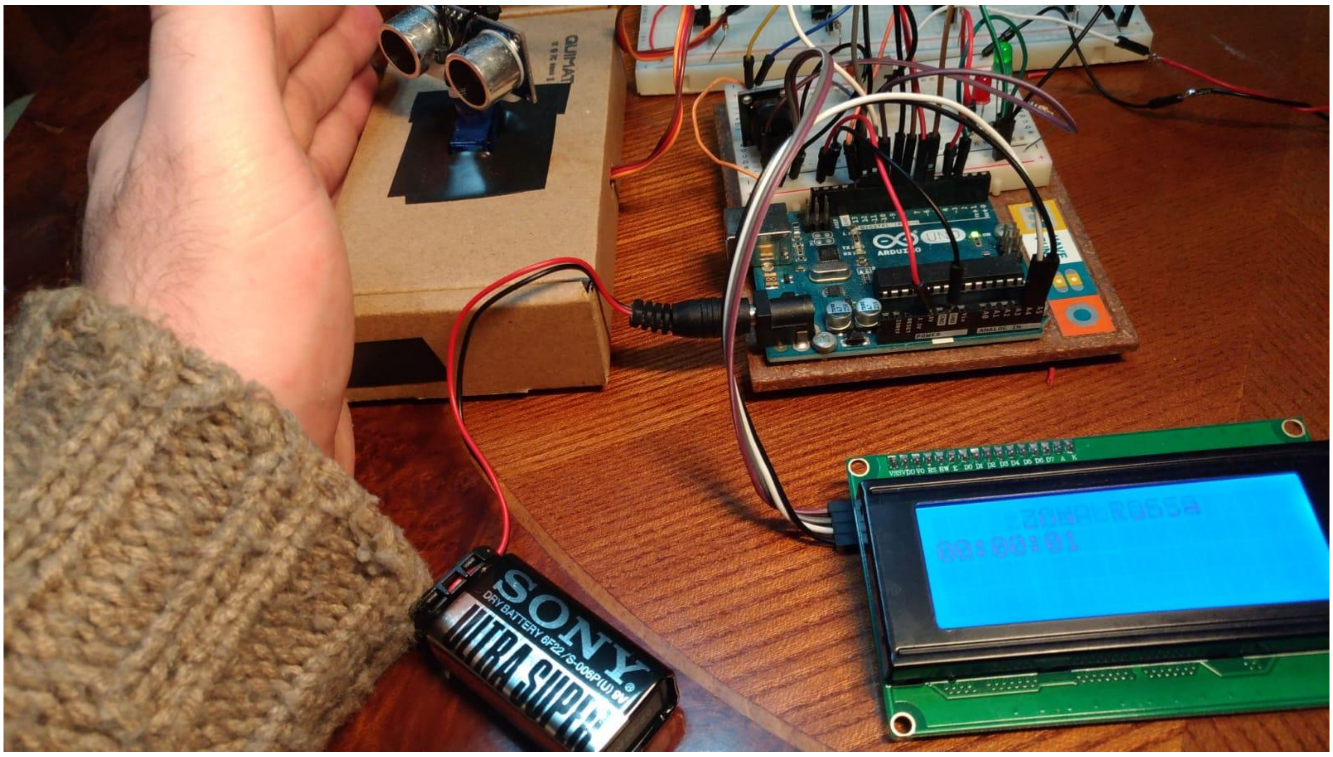
(9) start of movement of the servomotor



(10) the two LEDs will alternate from now on

Step 7

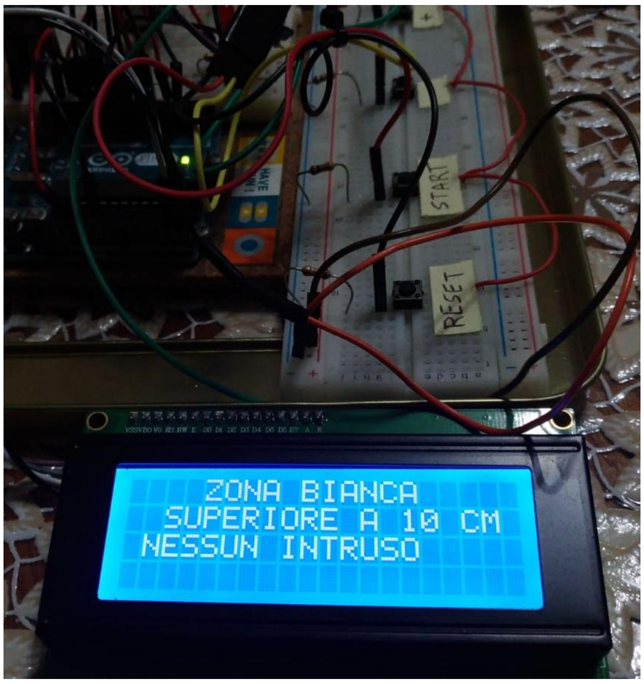
Once an obstacle on the trajectory has been effectively recognized as in figure (11), the behavior of the system varies and consequently also all the sensors for managing the output behave differently.

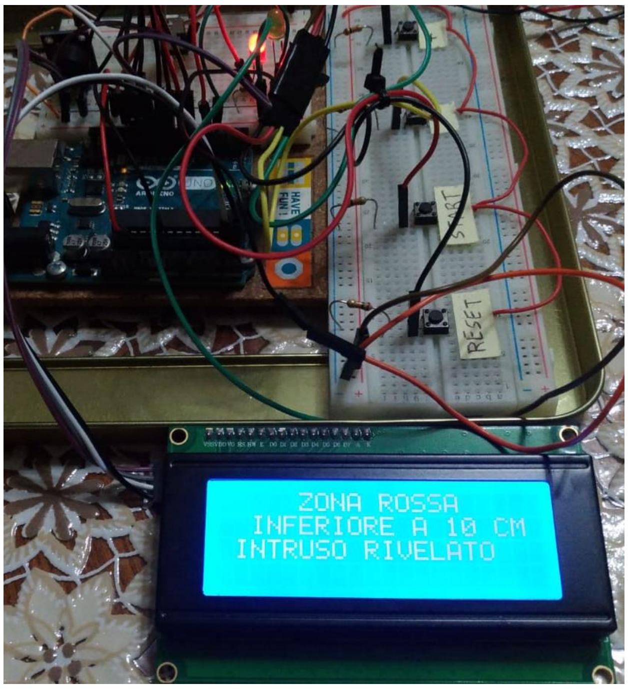


(11)

Step 8

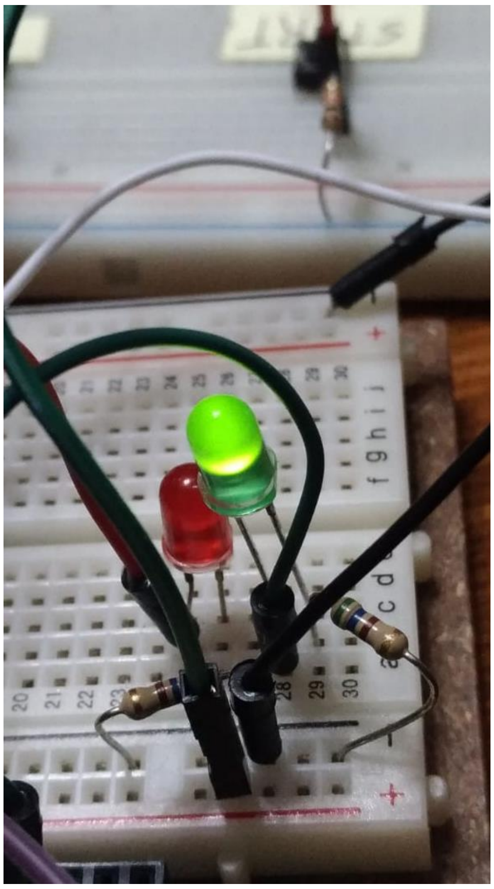
To actually understand the difference between one recognition status and the other, look at the following figures (12-15), (in addiction to listening to the acoustic signal from the buzzer in figure (16)).



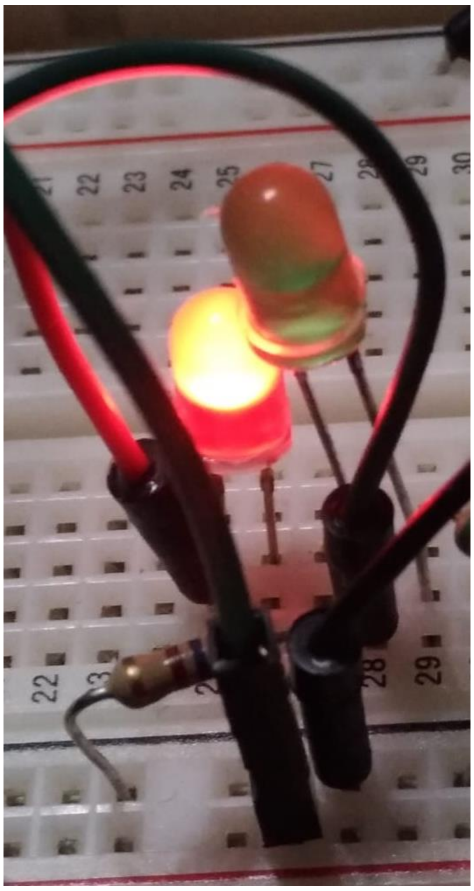


(12) display without any recognition

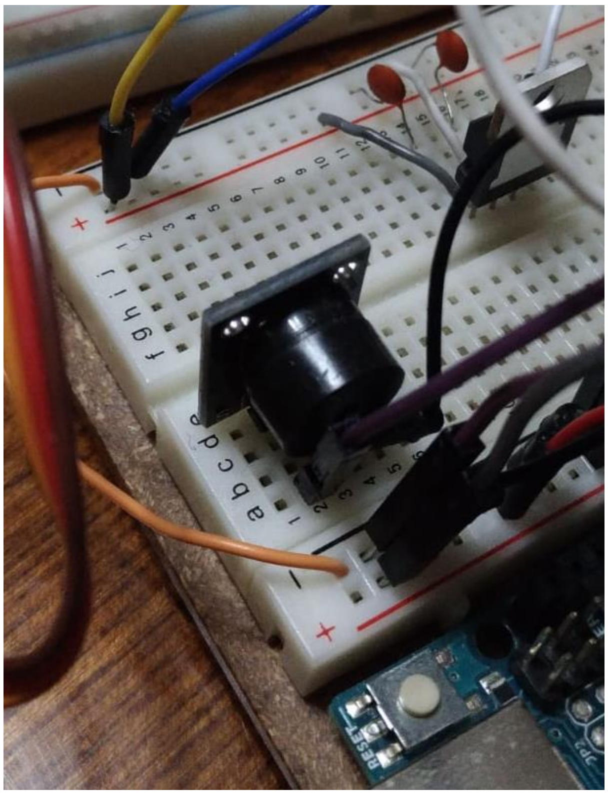
(13) display with recognition



(14) green led: no recognition



(15) red led: obstacle recognition



(16) buzzer

3.2 Source code

In this part is described software aspects of this project, for first the Arduino Sketch developed for this project

3.2.1 Arduino Sketch description

This Sketch can be summarized in three sections respectively:

- The first where we declare the variables and their status (behavior in the system).
- The second one where the passage of modes and the real assignment of the timer values are managed.
- The third which includes the real functioning of the system.

In detail, the code in question takes this form:

```
#include <DS3232RTC.h>                    // library for real time clock
#include <TimeLib.h> // The Time library adds timekeeping functionality to Arduino with or without external timekeeping hardware
#include <Wire.h>    // for analog pins and for I2C (Inter Integrated Circuit), serial communication systems used between integrated
circuits
#include <LiquidCrystal_I2C.h>            // library used to manage the display lcd
#include <Servo.h>                        // library required for the sg90 servo motor

LiquidCrystal_I2C lcd(0x27, 20, 4);       // Set the LCD address to 0x27 for a 20 chars and 4 line display

const int buzzerPin = 10;                 // pin for buzzer
int triggerPort = 11;                     // pin for the output module HC-SR
int echoPort = 12;                        // pin for the input module HC-SR

const int resetButtonPin = 6;             // pins dedicated to the keys
const int startStopButtonPin = 7;         // pins dedicated to the keys
const int downButtonPin = 8;               // pins dedicated to the keys
const int upButtonPin = 9;                 // pins dedicated to the keys

Servo servo;                               // name of the servo
int angle = 10;                           // variable needed to set the angle of the servo
int setupHours = 0;                       // How many hours will count down when started
int setupMinutes = 0;                      //How many min will count down when started
int setupSeconds = 0;                     //How many sec will count down when started
time_t setupTime = 0;
int currentHours = 0;                     // variable required for calculating hours
int currentMinutes = 0;                   // variable required for calculating min
int currentSeconds = 0;                   // variable required for calculating sec
time_t currentTime = 0;
time_t startTime = 0;
time_t elapsedTime = 0;
int resetButtonState = LOW;
long resetButtonLongPressCounter = 0;
int startStopButtonState = LOW;
int upButtonState = LOW;
int downButtonState = LOW;
int resetButtonPrevState = LOW;
int startStopButtonPrevState = LOW;
int upButtonPrevState = LOW;
int downButtonPrevState = LOW;
bool resetButtonPressed = false;
bool resetButtonLongPressed = false;
bool startStopButtonPressed = false;
bool upButtonPressed = false;
bool downButtonPressed = false;

const int MODE_IDLE = 0;                    // configurations required for the switch case
const int MODE_SETUP = 1;
const int MODE_RUNNING = 2;
const int MODE_RINGING = 3;

int currentMode = MODE_IDLE;            // 0=idle 1=setup 2=running 3=ringing
                                                                              i.    // Power up --> idle
                                                                             ii.    // Reset --> idle
                                                                            iii.    //  Start/Stop --> start or stop counter
                                                                             iv.    //  Up / Down --> NOP
                                                                              v.    // Reset (long press) --> enter setup
                                                                             vi.    //   Start/Stop --> data select
                                                                            vii.    //   Up --> increase current data value
                                                                           viii.    //   Down --> decrease current data value
                                                                             ix.    //   Reset --> exit setup (idle)

int dataSelection = 0;                          // Currently selected data for edit (setup mode, changes with Start/Stop)
                                                        // 0=hours (00-99) 1=minutes (00-59) 2=seconds (00-59)

void setup() {                         // installation code to run only once
servo.attach(13);                               // pin to drive the servo
servo.write(angle);                             // set servo to initial point
pinMode(triggerPort, OUTPUT);           // setting pin for HC-SR04 sensor
pinMode(echoPort, INPUT);               // setting pin for HC-SR04 sensor
lcd.begin();                             // Initializes the interface to the LCD screen, and specifies the dimensions of the display.
                                         begin() needs to be called before any other LCD library commands.
pinMode(resetButtonPin, INPUT);                     // setting the reset pin as input
pinMode(startStopButtonPin, INPUT);                 // setting the startStop pin as input
pinMode(upButtonPin, INPUT);                        // setting the up pin as input
pinMode(downButtonPin, INPUT);                      // setting the down pin as input
pinMode(buzzerPin, OUTPUT);                         // setting the buzzerPin previously declared at 10 as output
pinMode(10, OUTPUT);                                // setting the digital pin as an output for the passive buzzer
pinMode(4, OUTPUT);                                 // setting the pin for the red led
pinMode(3, OUTPUT);                                 // setting the pin for the green led
}
void loop() {       // start of the loop function which has the task of providing the board with all the information relating to
                        the program information and the necessary commands
startStopButtonPressed = false;        // setting of the boolean variables needed by the keys
upButtonPressed = false;
downButtonPressed = false;

/*
                          Reset button management
*/
resetButtonPressed = false;
resetButtonLongPressed = false;
resetButtonState = digitalRead(resetButtonPin);
if (resetButtonState != resetButtonPrevState) {                 // control on the state of the pressure of the key
resetButtonPressed = resetButtonState == HIGH;
resetButtonPrevState = resetButtonState;
} else // Long press management...
{
if (resetButtonState == HIGH) {             // control on how long the reset button is pressed, if the button has been pressed
            resetButtonLongPressCounter++;                     // start the increment of the variable and if it reaches 100 then
                                                               change the configurations
            if (resetButtonLongPressCounter == 100) {
                        resetButtonPressed = false;
                        resetButtonLongPressed = true;
                        resetButtonLongPressCounter = 0;
                        }
            } else {                                // otherwise if the count has not been reached, the counter resets and consequently also
                                                the status of the key
                        resetButtonLongPressCounter = 0;
```

```
                resetButtonPressed = false;
                resetButtonLongPressed = false;
}
}

/*
* Start/Stop button management                         check the status of the start/stop button
*/
startStopButtonPressed = false;
startStopButtonState = digitalRead(startStopButtonPin);  // A digitalRead is issued on the relevant pin and the result is
                                                            compared with a previously read value
if (startStopButtonState != startStopButtonPrevState) {  // if something has changed, the new value is stored for future
                                                            reference and the static variable "ButtonPressed"
startStopButtonPressed = startStopButtonState == HIGH; // bool is set to true if the button is pressed
startStopButtonPrevState = startStopButtonState;
}

/*
* Down button management                               check on the status of the down button
*/
downButtonPressed = false;
downButtonState = digitalRead(downButtonPin);
if (downButtonState != downButtonPrevState) {
downButtonPressed = downButtonState == HIGH;
downButtonPrevState = downButtonState;
}

/*
* Up button management                                 check on the status of the up button
*/
upButtonPressed = false;
upButtonState = digitalRead(upButtonPin);
if (upButtonState != upButtonPrevState) {
upButtonPressed = upButtonState == HIGH;
upButtonPrevState = upButtonState;
}

/*
                Mode management
*/
switch (currentMode) {                          // switch case related to mode management
                                                // here we examine the button state triggers ("xxxButtonPressed") and redirect the
                                                flow to the new correct state or perform the correct action
case MODE_IDLE:                                 // inactivity status equal to reset
if (resetButtonPressed) {                       // this block of code is able to detect the above prolonged pressure to access the
                                                respective modes
                Reset();
}
if (resetButtonLongPressed) {                   //if the reset button is pressed for a long period, it enters the setup mode
                currentMode = MODE_SETUP;
}
if (startStopButtonPressed) {
                currentMode = currentMode == MODE_IDLE ? MODE_RUNNING : MODE_IDLE;
                                    // if the value of mode idle is true then mode running is evaluated and mode idle is ignored
                if (currentMode == MODE_RUNNING) {              // while if mode idle is evaluated as false mode idle is valuated
                                                            and mode running ignored
                // STARTING TIMER!
                startTime = now();
                }
}
break;

case MODE_SETUP:                                            // setup mode
if (resetButtonPressed) {
                // Exit setup mode
```

```
        setupTime = setupSeconds + (60 * setupMinutes) + (3600 * setupHours);  // set the setupTimer variable
        currentHours = setupHours;
        currentMinutes = setupMinutes;                          // setting of variables to count the time
        currentSeconds = setupSeconds;
        dataSelection = 0;                  // variable required to switch from one time mode (min and sec hours) to another
        currentMode = MODE_IDLE;                                     // the current mode remains idle
}
if (startStopButtonPressed) {                // if the startstop button has been pressed it increases the selection mode of
                                             hours min and seconds
        // Select next data to adjust
        dataSelection++;
        if (dataSelection == 3) {                               // while if this reaches a certain tot it starts again
        dataSelection = 0;
        }
}
if (downButtonPressed) {      //if the down button has been pressed according to the time mode it is in, it decreases
                              (hours, min and sec) or starts again
        switch (dataSelection) {
        case 0: // hours
        setupHours--;
        if (setupHours == -1) {
              setupHours = 99;
        }
        break;
        case 1: // minutes
        setupMinutes--;
        if (setupMinutes == -1) {
                setupMinutes = 59;
        }
        break;
        case 2: // seconds
        setupSeconds--;
        if (setupSeconds == -1) {
                setupSeconds = 59;
        }
        break;
        }
}
if (upButtonPressed) {      //if the up button has been pressed according to the time mode it is in, it increases (hours, min
                              and sec) or starts again
        switch (dataSelection) {
        case 0: // hours
        setupHours++;
        if (setupHours == 100) {
                setupHours = 0;
        }
        break;
        case 1: // minutes
        setupMinutes++;
        if (setupMinutes == 60) {
                setupMinutes = 0;
        }
        break;
        case 2: // seconds
        setupSeconds++;
        if (setupSeconds == 60) {
                setupSeconds = 0;
        }
        break;
        }
}
break;

case MODE_RUNNING:                      //running mode, if the startstop button or the reset button are pressed, it returns
                                        to the idle state
```

```
if (startStopButtonPressed) {
		currentMode = MODE_IDLE;
}
if (resetButtonPressed) {
		Reset();
		currentMode = MODE_IDLE;
}
break;

case MODE_RINGING:		//ringing if one of the 4 buttons has been pressed, the program goes into the inactivity state
if (resetButtonPressed || startStopButtonPressed || downButtonPressed || upButtonPressed) {
		currentMode = MODE_IDLE;
}
break;
}

switch (currentMode) {
case MODE_IDLE:
case MODE_SETUP:
// NOP
break;
case MODE_RUNNING:
currentTime = setupTime - (now() - startTime);
if (currentTime <= 0) {
		currentMode = MODE_RINGING;
}
break;
case MODE_RINGING:
analogWrite(buzzerPin, 0);
break;
}

lcd.setCursor(0, 0);
switch (currentMode) {
case MODE_IDLE:
lcd.print("Allarme a tempo");
lcd.setCursor(0, 1);
lcd.print(currentHours);
lcd.print(" ");
lcd.print(currentMinutes);
lcd.print(" ");
lcd.print(currentSeconds);
lcd.print(" ");
break;
case MODE_SETUP:			// setup case
lcd.print("Setup mode: ");
switch (dataSelection) {	// another internal switch to select the hours, minutes and seconds
case 0:
		lcd.print("ORE ");
		break;
case 1:
		lcd.print("MIN");
		break;
case 2:
		lcd.print("SEC");
		break;
}
lcd.setCursor(0, 1);
lcd.print(setupHours);
lcd.print(" ");
lcd.print(setupMinutes);
lcd.print(" ");
lcd.print(setupSeconds);
lcd.print(" ");
break;
```

```
case MODE_RUNNING:                    // CASE OF COUNTDOWN, the display of min and sec hours is shown on the display
lcd.print("Tempo Rimanente");
lcd.setCursor(0, 1);
if (hour(currentTime) < 10) lcd.print("0");
lcd.print(hour(currentTime));
lcd.print(":");
if (minute(currentTime) < 10) lcd.print("0");
lcd.print(minute(currentTime));
lcd.print(":");
if (second(currentTime) < 10) lcd.print("0");
lcd.print(second(currentTime));
delay(10);
break;

case MODE_RINGING:                    // OPERATING CASE
//start servo
for(angle = 10; angle < 180; angle++)      // cycle for the movement of the servo in the first half of its run
{

digitalWrite(triggerPort, LOW);           // start of functioning of the HC-SR04 sensor with respective calibration
digitalWrite(triggerPort, HIGH);          // sends a 10 microsec pulse on trigger
delayMicroseconds(10);
digitalWrite(triggerPort, LOW);

long duration = pulseIn(echoPort, HIGH);   //calculation and configuration of the distance required for the HC-SR04 sensor
long r = 0.034 * duration / 2;

if (r <= 9) {                               // condition on distance r less than 10 cm (case of buzzer activation)
        digitalWrite(4, HIGH);              // in this case the green and red LEDs are inverted
        digitalWrite(3, LOW);
        lcd.setCursor(0, 0);              // set the cursor to write on the display in the character present in the column and
                                           row in position 0 and 0
        lcd.print("    ZONA ROSSA");
        lcd.setCursor(0, 1);              // set the cursor to write on the display in the character present in the second line
                                           and the first character
        lcd.print("  INFERIORE A 10 CM ");
        lcd.setCursor(0, 2);
        lcd.print(" INTRUSO RIVELATO");
        tone(10, 2000, 300);               // signal for pin 10 of the buzzer, activated by the respective method associated
                                           with frequency and duration
}

if (r <= 200 && r > 9) {            // condition on distance r in the range between 10 and 200
        digitalWrite(3, HIGH);              // in this case the green LED lights up and the red one remains off
        digitalWrite(4, LOW);
        lcd.setCursor(0, 0);              // set the cursor to write on the display in the character present in the column and
                                           row in position 0 and 0
        lcd.print("    ZONA BIANCA");
        lcd.setCursor(0, 1);              // set the cursor to write on the display in the character present in the second line
                                           and the first character
        lcd.print("  SUPERIORE A 10 CM ");
        lcd.setCursor(0, 2);
        lcd.print(" NESSUN INTRUSO");
}

servo.write(angle);                        // servo command necessary to perform the rotation, the argument is in fact the
                                           rotation angle that increases and decreases according to the case
}

// servo return
for(angle = 180; angle > 10; angle--)      // cycle for the movement of the servo in the second half of its run
{

digitalWrite(triggerPort, LOW);           // start of operation of the HC-SR04 sensor with respective calibration
digitalWrite(triggerPort, HIGH);          // send a 10 microsec pulse on trigger
delayMicroseconds(10);
digitalWrite(triggerPort, LOW);

long duration = pulseIn(echoPort, HIGH);          // calculation and configuration of the distance required for the HC-SR04
                                                   sensor
long r = 0.034 * duration / 2;

            if (r <= 200 && r > 9) {              // condition on distance r in the range between 10 and 200
            digitalWrite(3, HIGH);                // in this case the green LED lights up and the red one remains off
            digitalWrite(4, LOW);
            lcd.setCursor(0, 0);                  // set the cursor to write on the display in the character present in the column
                                                   and row in position 0 and 0
            lcd.print("    ZONA BIANCA");
            lcd.setCursor(0, 1);                  // set the cursor to write on the display in the character present in the second
                                                  line and the first character
            lcd.print("  SUPERIORE A 10 CM ");
            lcd.setCursor(0, 2);
            lcd.print(" NESSUN INTRUSO");
}

if (r <= 9) {                                    // condition on distance r less than 10 cm (case of buzzer activation)
            digitalWrite(4, HIGH);               // in this case the green and red LEDs are inverted
            digitalWrite(3, LOW);
            lcd.setCursor(0, 0);                  // set the cursor to write on the display in the character present in the column
                                                   and row in position 0 and 0
            lcd.print("     ZONA ROSSA");
            lcd.setCursor(0, 1);                  // set the cursor to write on the display in the character present in the second
                                                   line and the first character
            lcd.print("  INFERIORE A 10 CM ");
            lcd.setCursor(0, 2);
            lcd.print(" INTRUSO RIVELATO");
            tone(10, 2000, 300);                   // signal for pin 10 of the buzzer, activated by the respective method associated
                                                    with frequency and duration
}

servo.write(angle);                              //servo command needed to rotate, the argument is in fact the rotation angle
                                                 which increases and decreases according to the case
}
}
}

void Reset() {
currentMode = MODE_IDLE;
currentHours = setupHours;
currentMinutes = setupMinutes;
currentSeconds = setupSeconds;
}
```

Chapter 4

Conclusions and future implementations

This work confirmed the great success of Arduino in communities around the world, in fact, its multiple uses allow anyone to create a customized embedded system, according to their needs and with costs often lower than other devices already integrated on the market, which very often don't satisfy fully the needs of various users.

In particular, the project in question represents an excellent case study that can be implemented in more than one way: via hardware, connecting other types of sensors, such as the infrared one, the one for detecting heat, or simply increasing the number of servos and HC-SR to have a more accurate detection and on more trajectories; or via software, in this case it is possible to provide further specific conditions such as the possibility of controlling one sensor with respect to another or the possibility of simply introducing conditions for the different inputs, or alternatively it is possible to decide whether to operate the device in the second phase making its use limited to a certain period of time.

Any project of this kind can be implemented in 1000 different ways, without ever arriving at a definitive version, the beauty of Arduino lies in this, that is, it allows us to customize every aspect according to our own objectives.

In conclusion, we could further optimize the work by welding the various components on an Arduino stripboard, furthermore, in order to further improve the work, we could provide a suitable case for the device, this would allow the project to be use concretely and provide a more solid and resistant structure to breakdowns and any damage from external factors.