ANKARA YILDIRIM BEYAZIT UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCE

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

Summer Project Report

(Smart door & Proximity alert)

Prof. Dr. Hüseyin CANBOLAT

Prepared by:

Mohammad MUTTAQI – 17050241030

Introduction:

My purpose in this project is to create a way for certain crowd to be able to gather in a place (especially in these unfortunate times) without breaching the safe distance. I decided to do so in two separate steps. The first step is to control the number of people in said place, and the second step is to control the distance between said people.

The first step will be implemented on the gates, there would be displays on the gates showing the maximum number of people allowed inside and the number of people present inside. When a person (employee, costumer, …etc.) enters or exits said place, the number of the people would change accordingly.

The second step will be implemented on the people, the necessary components for this part are very small so they could be easily wearable. The components can be implemented on a face shield, an arm band, a bracelet …etc. Before entering, the person would wear the device, and when they come close to another person they would be alerted.

As mentioned above the project is done in two separate steps. Hence, I will be reporting the project in two parts. So, without farther ado let us get started.

In the link below you can find all the files used in this project (schematic diagrams, videos, codes…etc.) also a word and pdf copy of this report:

https://drive.google.com/drive/folders/1SxYMUPyfha_U6FuJY3NLfNBJFkf0XaUM?usp=sharing

Part-1 (Smart door):

Components:

For the lack of better naming, I named this part as the “Smart Door”. First, let us get familiar with the components used in this part. For the main processing I used an Arduino UNO R3 clone that uses a CH340 chip, then I used an RC522 RFID NFC Module (13.56 MHz) as gate control. For the display I used a 16x2 unilluminated green HY-1602F-001 LCD display and hence it has lots of connection pins I also used a PCF8574 I2C convertor. To simulate the gate itself I used a Tower Pro SG90 RC Mini (9gr) Servo Motor, and I used four LEDs and a push button.

Below is the link for the videos of part-1:

https://drive.google.com/file/d/1ZC1_cADVyP9l3HF3P_HXwIomKJ5kXv8c/view?usp=sharing

https://drive.google.com/file/d/1X9BHz4XMdIg-pZ7y5xHJ6OZ0U3caB42w/view?usp=sharing

Working principle:

First, let me explain the LEDs. When it is available inside the blue LED would be lit, meaning that you can enter. When the maximum number is reached the yellow LED will be lit and the door won’t open. When an authorized card is shown, the green LED will lit, but when an unauthorized card is shown the red LED will lit. The number of people inside will only change if an authorized card is shown and the door will only open likewise or when the maximum number is not reached.

My initial plan was for the exit action to be also performed using the RFID Module, but unfortunately, I did not succeed. First, I wanted to implement two RFID Modules. One for the entering gate and the other for the exiting gate. I learned that I could connect them in serial but, I also learned that even if I connect two RFID Modules they would perform as one device. Later I decided to use only one RFID Module, but when a card is shown twice, the second time would mean that the person has exited. But as you can guess from that video, this didn’t work

either. So, I decided to use a push button to simulate a sensor for when someone is exiting.

You might wonder why I used an RFID Module? I could have used a button for the entering process too, it would've been much easier. But I choose the RFID method because it is more reliable, and you have an authentication method too, so no one could enter randomly. And as I mentioned before I was planning to use it for the exiting process too, but I couldn't succeed.

By the way, another benefit of an RFID reader is that you can use any NFC chip for identification instead of just an RFID card or chip.

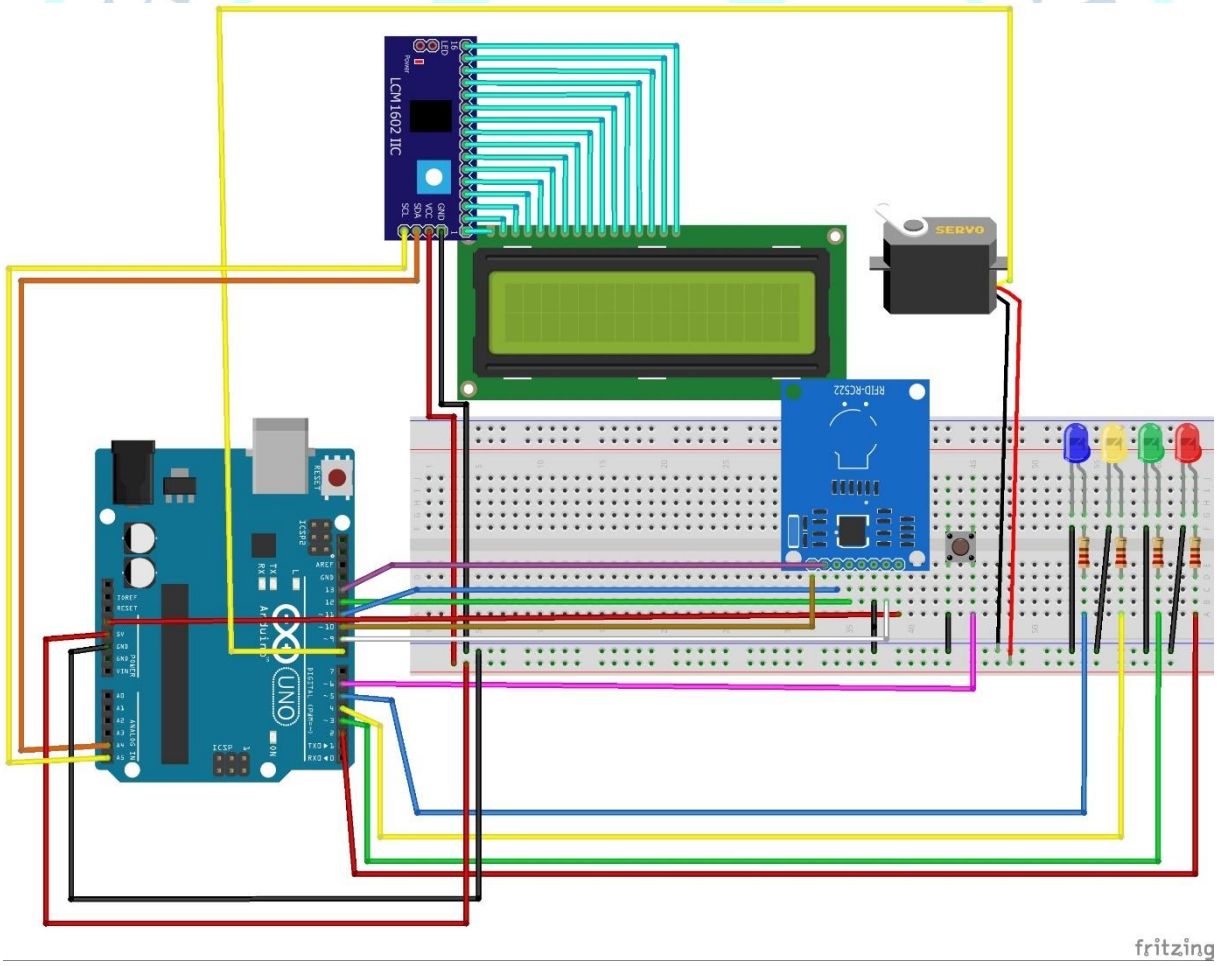

Figure 1.1 – The schematics diagram for the part-1 (Smart door 😁).

Code review:

Below in *figure 1.2*, is the section where I have defined the necessary libraries and variables.

Part_1

```
//Mohammad MUTTAQI - 17050241030
//Summer project-1
//Smart door

#include <FaBoLCD_PCF8574.h>
#include <Wire.h>
#include <MFRC522.h>
#include <Servo.h>
#include <SPI.h>

int RST_PIN = 9;                    //defining the reset pin for the RC522 module.
int SS_PIN = 10;                    //defining the chip select pin for the RC522 module.
int servoPin = 8;                   //defining the Servo motor pin.
int leds[]={2,3,4,5};               //our LEDs.
int exit_buttonPIN = 6;             //the exit button.

FaBoLCD_PCF8574 lcd;
Servo motor;                        //Creating a variable fot the Servo motor.
MFRC522 rfid(SS_PIN, RST_PIN);      //setting up the RC522 module.

//Defining the ID number of the authorized cards.
byte ID1[4] = {244, 110, 154, 42};
byte ID2[4] = {192, 174, 198, 50};
byte ID3[4] = {233, 229, 102, 194};
byte ID4[4] = {217, 190, 211, 162};

int max_p = 5;                      // maximum number of people allowed
int people = 0;                     // number of people inside
int exit_buttonState = 0;           // current state of the exit button
int exit_lastButtonState = 0;       // previous state of the exit button
```

Figure 1.2

In *figure 1.3*, in the first line I included the library for the I^2C convertor. In the 2nd line I included the wire library because to activate the I^2C protocol. In the 3rd line the library for the RFID Module is included. In the 4th line the library for the servo motor and in the 5th line the library for the SPI protocol is included. The SPI protocol is used for the RFID module.

```
#include <FaBoLCD_PCF8574.h>
#include <Wire.h>
#include <MFRC522.h>
#include <Servo.h>
#include <SPI.h>
```

Figure 1.3

```
int RST_PIN = 9;                        //defining the reset pin for the RC522 module.
int SS_PIN = 10;                        //defining the chip select pin for the RC522 module.
int servoPin = 8;                       //defining the Servo motor pin.
int leds[]={2,3,4,5};                   //our LEDs.
int exit_buttonPIN = 6;                 //the exit button.

FaBoLCD_PCF8574 lcd;
Servo motor;                            //Creating a variable fot the Servo motor.
MFRC522 rfid(SS_PIN, RST_PIN);          //setting up the RC522 module.
```

Figure 1.4

In *figure 1.4* in the 1st line I defined the reset pin for the RFID Module as pin number 9 on the Arduino.

In the 2nd line I defined the chip select pin for the RFID Module as pin number 10 on the Arduino.

And in the 3rd line I defined the pin number 8 on the Arduino as the Servo motor pin.

In the 4th line I declared an integer array that holds the LEDs. The pins 2-5 on the Arduino are defined as the LED pins.

And in the 5th line I declared the variable the pin number 6 on the Arduino as the exit button pin.

In the 6th line I created a variable for the LCD display and in the 7th line a created variable for the Servo motor.

Finally, in the 8th line I set up the RFID Module.

In *figure 1.5*, I defined the ID numbers of the authorized RFID cards/chips. I defined four because that is what I had available. You can add more IDs if desired.

```
//Defining the ID number of the authorized cards.
byte ID1[4] = {244, 110, 154, 42};
byte ID2[4] = {192, 174, 198, 50};
byte ID3[4] = {233, 229, 102, 194};
byte ID4[4] = {217, 190, 211, 162};
```

Figure 1.5

```
int max_p = 5;                          // maximum number of people allowed
int people = 0;                         // number of people inside
int exit_buttonState = 0;               // current state of the exit button
int exit_lastButtonState = 0;           // previous state of the exit button
```

Figure 1.6

In *figure 1.6* the 1st line I declared the integer `max_p`. It holds the maximum number of people allowed. Again, I defined the max. number as 5 for experimenting purposes, but it can be changed as whished.

In the 2nd line a declared the integer `people`. It is going to hold the number of people inside.

In the 3rd and 4th lines, I declared two integers. One for the current state of the exit button, and the other for its previous state.

Below in *figure 1.7*, is the set-up function.

```
void setup() {

  lcd.begin(16,2);
  lcd.setCursor(0,0);
  lcd.print("max sayi = ");
  lcd.print(max_p);
  lcd.setCursor(0,1);
  lcd.print("mevcut sayi= ");
  lcd.print(people);
  delay(500);

  motor.attach(servoPin);
  Serial.begin(9600);
  SPI.begin();
  rfid.PCD_Init();

  pinMode (exit_buttonPIN, INPUT_PULLUP);
  for (int i=0; i<4; i++){
    pinMode(leds[i], OUTPUT);
    }
}
```

Figure 1.7

In the first line the LCD display is initialized and its type as a 16x2 display is declared.

In the 2nd line the cursor is set to the 1st column and the 1st line of the LCD display.

In the 3rd the line the words `"max sayi ="` are displayed on the LCD. And in the 4th line the maximum number is being displayed on the LCD.

In the 5th line the cursor is set to the 1st column and the 2nd line of the LCD display.

Then, in the 6th line the words `"mevcut sayi ="` are displayed on the LCD. And in the 7th line the number of people inside is being displayed on the LCD.

In the 8th line I have set a 500-millisecond delay.

In the 9th line I am defining which pin the servo has been connected to using the `attach` command.

In the 10th, to read the IDs of the cards/chips on the serial monitor, I initialize the serial communication with a baud rate of 9600.

Then, in the 11th line for the Arduino to be able to communicate with the RFID Module I initialize the SPI protocol.

In the 12th line, the RFID Module is initialized.

In the 13th line, I have set the pin mode for the exit button as a pull-up input.

And finally, in the lines 14 – 16, I am setting the pin mode for the LEDs as output and I am doing so using a for loop so I wouldn't have to set every LED separately.

Below in *figure 1.8*, are the first lines of the main function.

```
void loop() {

  checkDown();

  if (people>= max_p){
    digitalWrite (5, LOW);
    digitalWrite (4, HIGH);
  }
  else {
    digitalWrite (4, LOW);
    digitalWrite(5, HIGH);
  }

  if ( ! rfid.PICC_IsNewCardPresent())    // waiting for the new card to be scanned.
    return;

  if ( ! rfid.PICC_ReadCardSerial())      // waiting if the card is not scanned.
    return;
```

Figure 1.8

In the first line I have recalled the `checkDown` function. Said function is responsible for the exit button and decreasing the number of the people inside. I will be explaining it later in detail.

Then in lines 2 – 9, I have set the conditions for the Red and the Blue LEDs. As I explained before, the conditions are set so when the number of people inside is less than the max. number the Blue LED will be lit and the Red LED will be off, and the opposite if the number of people has reached the max. number.

Then, in the 10th line the condition is set so the program would wait until a new card/chip is shown to the reader.

And in the 11th line the condition is set so the program would wait if no card/chip is scanned.

Below in *figure 1.9*, the comparison process between the scanned card/chip and the predefined IDs is happening.

And if the conditions are satisfied (meaning: if the scanned ID matches any of the pre-determined IDs) then the program will start performing the commands in figure 1.10.

```
if ((rfid.uid.uidByte[0] == ID1[0] &&     // comparing the scanned card with the ID1 variable.
rfid.uid.uidByte[1] == ID1[1] &&
rfid.uid.uidByte[2] == ID1[2] &&
rfid.uid.uidByte[3] == ID1[3])||
(rfid.uid.uidByte[0] == ID2[0] &&         // comparing the scanned card with the ID2 variable.
rfid.uid.uidByte[1] == ID2[1] &&
rfid.uid.uidByte[2] == ID2[2] &&
rfid.uid.uidByte[3] == ID2[3]) ||
(rfid.uid.uidByte[0] == ID3[0] &&        // comparing the scanned card with the ID3 variable.
rfid.uid.uidByte[1] == ID3[1] &&
rfid.uid.uidByte[2] == ID3[2] &&
rfid.uid.uidByte[3] == ID3[3]) ||
(rfid.uid.uidByte[0] == ID4[0] &&       // comparing the scanned card with the ID4 variable.
rfid.uid.uidByte[1] == ID4[1] &&
rfid.uid.uidByte[2] == ID4[2] &&
rfid.uid.uidByte[3] == ID4[3])) {
```

Figure 1.9

Below in *figure 1.10*, in the first line I have set the condition for when the number of people is less than the max. number.

```
if (people < max_p){
digitalWrite(5, LOW);
digitalWrite(3, HIGH);
Serial.println("Door has opened");
print_on_screen();
motor.write(180);
delay(2000);
digitalWrite(3, LOW);
motor.write(0);
delay(1000);

people++;
lcd.clear();
lcd.setCursor(0,0);
lcd.print("max sayi = ");
lcd.print(max_p);

lcd.setCursor(0,1);
lcd.print("mevcut sayi=");
lcd.print(people);
}
```

Figure 1.10

In the lines 2 – 3, the Blue LED is turned off and the green LED is lit.

Then in line 4, the words `"Door has opened"` will be displayed to the serial screen. Afterwards, in the 5th line the `print_on_screen` function has been recalled. This function will print the scanned ID to the serial screen, later I have explained it in detail.

In the 6th line, using the `write` command, the motor's position is changed to 180 degrees. Simulating the opening of the door lock. Then, in the 7th line we hold said position for 2 seconds.

In the 8th line, the Green LED is tuned off. And in the 9th line the position of the motor is changed back to 0 degrees. Then, in the 10th line we wait for another second.

And finally, in the 11th line the number of the people inside is increased, then in lines 12 – 18, the updated number is displayed on the LCD display.

The condition below in *figure 1.11* activates if an ID is scanned while the max. number of people is reached.

```
else {                          // works in case we reach the maximum number.
lcd.clear();
lcd.setCursor(0,0);
lcd.print("-max sayi dolu-");

lcd.setCursor(0,1);
lcd.print("max sayi = ");
lcd.print(max_p);
}
  }
```

Figure 1.11

In said condition, the words `"-max sayi dolu-"` will be displayed on the LCD.

If the scanned ID does not match the pre-defined IDs the condition below in *figure 1.12* will be satisfied.

```
  else{                                  // works in case of an unauthorized attempt.
    Serial.println("Unauthorized Card");
    print_on_screen();
    digitalWrite(5, LOW);
    digitalWrite(2, HIGH);
    delay(2000);
    digitalWrite(2, LOW);
  }


  rfid.PICC_HaltA();
}
```

Figure 1.12

In lines 2 – 3 the words `"Unauthorized Card"` will be printed on the serial screen then, the unauthorized ID will be also displayed on the serial screen.

Then, in lines 4 – 7 for two seconds the Blue LED is turned off and the Red LED is lit instead.

Finally, in the last line, using the `PICC_HaltA` command we stop the reading process of the RFID reader until a new ID is scanned. If we don't do this, the reader will continuously read the ID, and we won't be able to observe the information from the RFID Module.

Below in *figure 1.13*, is the `print_on_screen` function.

```
void print_on_screen()
{
  Serial.print("ID number: ");
  for(int j = 0; j < 4; j++){
    Serial.print(rfid.uid.uidByte[j]);
    Serial.print(" ");
  }
  Serial.println("");
}
```

Figure 1.13

As mentioned before, utilizing a for loop, this function prints the ID that is scanned from a card/chip to the serial display. This way, it can be stored, or it can be used to authorize unauthorized IDs.

Lastly in *figure 1.14*, is the `checkDown` function.

```
void checkDown(){

  exit_buttonState = digitalRead(exit_buttonPIN);

  if (exit_buttonState != exit_lastButtonState){
    if (exit_buttonState == LOW){

        people --;
        if (people < 0){
          people = 0;
        }
        lcd.clear();
        lcd.setCursor(0,0);
        lcd.print("max sayi = ");
        lcd.print(max_p);

        lcd.setCursor(0,1);
        lcd.print("mevcut sayi=");
        lcd.print(people);

    }
    delay(50);
  }
  exit_lastButtonState = exit_buttonState;
}
```

Figure 1.14

As briefly mentioned, this function performs the duty of the exit button, decreases the number of people inside, and prints the updated details to the LCD display.

In the first line of the function, first the state of the button is read using the `digitalRead` command then, the value is assigned to the `exit_buttonState` variable as the current state of the exit button.

Then, in the lines 2 and 3 we are checking if the current state of the button is not similar to the previous state (LOW). In short, it is checking if the state is HIGH or not.

In the 4th line, if the conditions above are satisfied, then the number of people inside will be decreased.

In lines 5 and 6, the condition is set to when the number of people gets less than zero, it is changed into zero.

This is in case, let's say if the button is pushed wile the number of people is zero the value won't change. Because, if there is no one inside then, no one will be able to activate the sensor.

Afterwards, in lines 7 – 14 the new data is printed to the LCD display and the program wait for 50 milliseconds.

In the end the current state is assigned to the last state, so the current state could become the next state and the present state would be the past state.

Part-2 (Proximity alert):

Components:

I named this part as the "Proximity alert". As before, let us start with components used in this part. Just like before, Arduino UNO R3 clone is used as the main processing unit. To calculate the distance, I used an HC-SR04 ultrasonic sensor, for vocal alerts I used a buzzer, and for visual alert I used a Red and a Blue LED.

Below is the link for the video of part-2:

https://drive.google.com/file/d/1fW5NOQqMyAPODl7zOCSUYi1qQfEZ5IrZ/view?usp=sharing

Working principle:

In comparison to the first part this part is very simple. The sensor sends sound waves and catches them back. We calculate the time that takes sound to travel back, and by doing so we know if something is in the path of the sound waves or not. The range of the sensor is four meters. Which is more than enough for this project. Even though the range is very high, I minimized it to 20 cm. Because it would be easier to check if it is working or not. Otherwise, I would have to conduct the experiment in a very large space.

In short, when an object/person is in the range of the sensor, the buzzer would start beeping and the Red LED would also start flashing. But when the range is clear the Blue LED will be lit.

I would also like to add that, since it is aimed to be a wearable device, and we don't have lots of components, it is preferred to use an Arduino Nano for its smaller size. But since, I already had an Arduino UNO, so I used it.

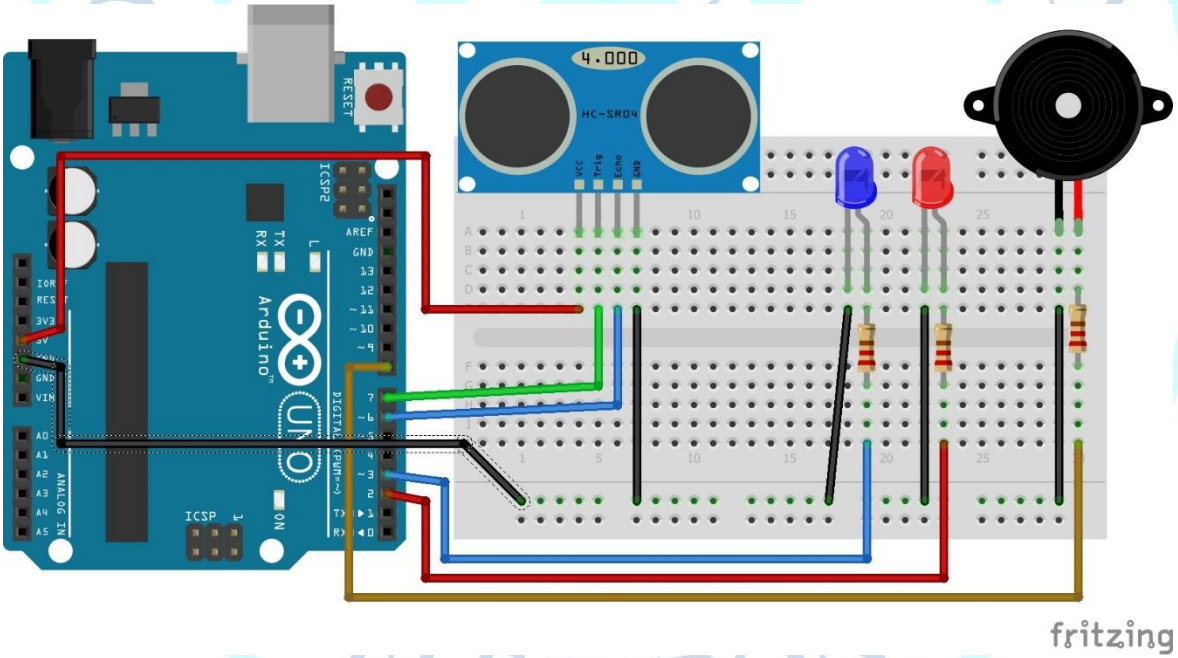

Figure 2.1 – The schematics diagram for the part-2 (Proximity alert).

Code review:

Below in *figure 2.2*, is the section where I have defined the pins and necessary variables. Unlike the first part, this part does not need any additional libraries.

Part_2 §

```
//Mohammad MUTTAQI - 17050241030
//Summer project-1
//Proximity alert

#define echoPIN 6
#define trigPIN 7
#define buzzerPIN 8

int RED = 2;
int BLUE = 3;

int max_range = 20;
int min_range = 0;
```

Figure 2.2

In the first line of the code, the pin number 6 from the Arduino is set as the echo pin of the ultrasonic sensor. Similarly, in the 2nd line I defined the 7th pin as the trig pin.

Then, in line 3, I defined the 8th pin as the pin that is connected to the buzzer.

In the 4th and the 5th lines, I defined the Red and Blue LEDs, and assigned to them the 2nd and the 3rd pins.

In the 6th line, I declared an integer variable that will hold the value for the maximum range. I set the value as 20 cm, but it can be changed with any value up to 4 meters.

And in the 7th line I declared an integer variable that will hold the value for the minimum range. Which I set as zero.

Below in *figure 2.3*, is the set-up function.

First, the trig pin mode is set to output. Then, the echo pin is set to input, and then the buzzer pin and the LEDs are set as outputs.

```
void setup() {
  pinMode(trigPIN, OUTPUT);
  pinMode(echoPIN, INPUT);
  pinMode(buzzerPIN, OUTPUT);
  pinMode(RED, OUTPUT);
  pinMode(BLUE, OUTPUT);
}
```

Figure 2.3

Below in *figure 2.4*, is the main function.

```
void loop() {

  int measured = measurement(max_range, min_range);
  melody(measured*10);
}
```

Figure 2.4

First, the integer `measured` has been declared. Then, the function `measurement` has been assigned to the integer.

Afterwards, the function `melody` is declared, and the integer `measured*10` has been assigned as a variable in the function melody.

Below in *figure 2.5*, is the function `measurement`. In this function the necessary measurements will be done, so in result we get the measured distance.

```
//the necessery measurements will be done in this function
int measurement(int maxrange, int minrange)
{
  long duration, distance;

  digitalWrite(trigPIN,LOW);
  delayMicroseconds(2);
  digitalWrite(trigPIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPIN, LOW);

  duration = pulseIn(echoPIN, HIGH);
  distance = duration / 58.2;
  delay(50);
  if (distance >= maxrange){
    digitalWrite(3, HIGH);
  }
  else{
    digitalWrite(3, LOW);
  }
  if(distance >= maxrange || distance <= minrange)
  return 0;
  return distance;
}
```

Figure 2.5

First, we declare to new variables `duration` and `distance` of type `long`, so they could store larger values.

In lines 3 – 7, using the `digitalWrite` command, the trig pin is first set to LOW, then we wait for two microseconds and then, the trig pin is set to HIGH. After waiting for 10 microseconds, the trig pin is set to LOW again.

In line 8 I used the `pulseIn` command, which stores the duration that the echo pin has been HIGH as integer. Then assigned the stored value to the variable `duration`.

Then in the 9th, by dividing the duration of the sound waves (that we obtained from the sensor) by 58.2 we get the distance in cm and store it to the variable `distance`. Afterwards, we wait for 50 milliseconds.

In lines 11 – 14, the conditions are set for the Blue LED to be lit if the max. range is not breached, and for it to be off while something is in range.

In lines 15 – 17 the conditions are set to check for the maximum range. Since I defined the max. range as 20 cm, if the distance is greater than it, it will be changed to zero, and if the distance is not greater then, the distance is not changed.

Below in *figure 2.6*, is the `melody` function.

```
int melody(int dly)
{
  tone(buzzerPIN, 440);
  digitalWrite(2, HIGH);
  delay(dly);
  noTone(buzzerPIN);
  digitalWrite(2, LOW);
  delay(dly);

}
```

Figure 2.6

This function will give us the beeping noise and the blinking from the Red LED.

The variable `dly` is the `measurement*10`.

First, the melody with code 440 is broadcasted using the `tone` command. The Red LED will also be lit, and the function waits for the period that we got from the sensor. Then, using the `noTone` command the melody stops broadcasting. Also, the LED is turned off, and like before, the function waits for the period that we got from the sensor.

Since these happen very fast, we get the illusion of beeping and blinking.