**Chapter 5**

**Implementation**

**5.1 Tracking User activity**

In this module, we have implemented existing system features in which user activity such as Heart rate, steps, calories and distance are being tracked. These tracking can be achieved with the help of Pulse rate sensor and MPU6050 hardware. This hardware takes analog input and produces digital output which is further processed to get the desired output.

In case of Pulse rate sensor, this module has 3 pins: Data pin, VCC pin and Ground pin. The data pin requires Analog input and produces digital output. The data pin of this module is inserted to A3 pin of Arduino and VCC and Ground to Arduino’s VCC and Ground pins respectively. Pseudo code:

*void pulseRateFromSensor() {*

*myBPM = pulseSensor.getBeatsPerMinute(); // Calls function on our pulseSensor object that returns BPM as an "int".*

*// "myBPM" hold this BPM value now.*

*if (pulseSensor.sawStartOfBeat()) { // Constantly test to see if "a beat happened".*

*Serial.println("A HeartBeat Happened ! "); // If test is "true", print a message "a heartbeat happened".*

*Serial.print("BPM: "); // Print phrase "BPM: "*

*Serial.print(myBPM); // Print the value inside of myBPM.*

*}*

*}*

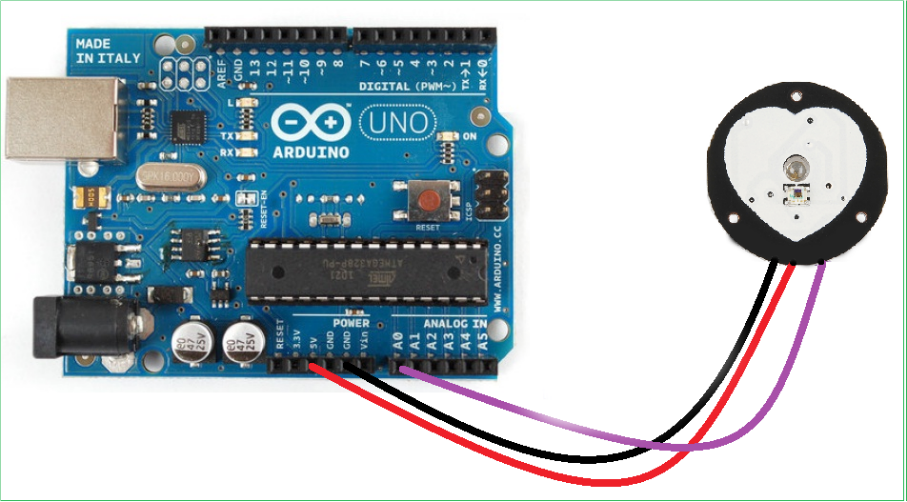


Figure 5.1.1: Pulse rate interfacing with Arduino

In case of MPU6050, this module is used for counting steps of the user with of help of accelerometer and gyroscope in it. This module consists of 6 pins out of which we require 4 pins VCC, GND, SCL and SDA. SCL and SDA pins are connected to A4 and A5 pin respectively, VCC and GND to Arduino’s VCC and GND pin. Pseudo code:

*void recordAccelRegisters() {*

*Wire.beginTransmission(0b1101000); //I2C address of the MPU*

*Wire.write(0x3B); //Starting register for Accel Readings*

*Wire.endTransmission();*

*Wire.requestFrom(0b1101000, 6); //Request Accel Registers (3B - 40)*

*while (Wire.available() < 6);*

*accelX = Wire.read() << 8 | Wire.read(); //Store first two bytes into accelX*

*accelY = Wire.read() << 8 | Wire.read(); //Store middle two bytes into accelY*

*accelZ = Wire.read() << 8 | Wire.read(); //Store last two bytes into accelZ*

*processAccelData();*

*}*

*void processAccelData() {*

*gForceX = accelX / 16384.0;*

*gForceY = accelY / 16384.0;*

*gForceZ = accelZ / 16384.0;*

*}*

*void recordGyroRegisters() {*

*Wire.beginTransmission(0b1101000); //I2C address of the MPU*

*Wire.write(0x43); //Starting register for Gyro Readings*

*Wire.endTransmission();*

*Wire.requestFrom(0b1101000, 6); //Request Gyro Registers (43 - 48)*

*while (Wire.available() < 6);*

*gyroX = Wire.read() << 8 | Wire.read(); //Store first two bytes into accelX*

*gyroY = Wire.read() << 8 | Wire.read(); //Store middle two bytes into accelY*

*gyroZ = Wire.read() << 8 | Wire.read(); //Store last two bytes into accelZ*

*processGyroData();*

*}*

*void processGyroData() {*

*rotX = gyroX / 131.0;*

*rotY = gyroY / 131.0;*

*rotZ = gyroZ / 131.0;*

*}*

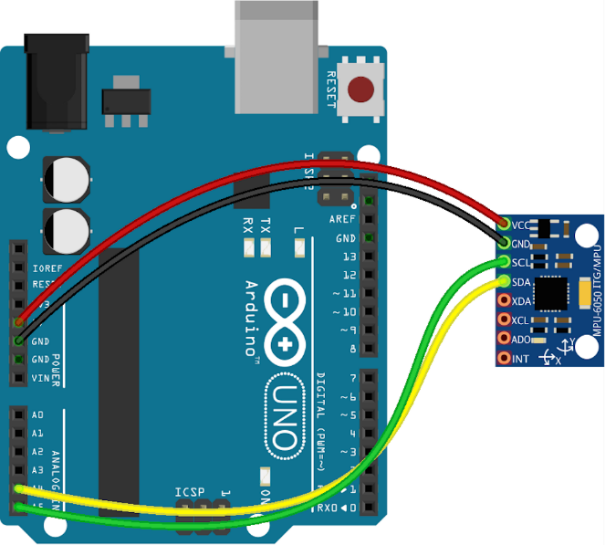


Figure 5.1.2: MPU6050 interfacing with Arduino

With the help of the above code we get the raw values of accelerometer and gyroscope and these raw values are further processed to get the appropriate steps, distance and calories. Pseudo code:

*void calculateSteps() {*

*gForceX = float(gForceX);*

*gForceY = float(gForceY);*

*gForceZ = float(gForceZ);*

*total = sqrt((gForceX \* gForceX) + (gForceY \* gForceY) + (gForceZ \* gForceZ));*

*total = (total + (total - 1)) / 2;*

*Serial.println(total);*

*delay(200);*

*//cal steps*

*if (total > threshhold && flag == 0)*

*{*

*steps = steps + 1;*

*flag = 1;*

*}*

*else if (total > threshhold && flag == 1)*

*{//do nothing}*

*if (total < threshhold && flag == 1)*

*{*

*flag = 0;*

*}*

*Serial.println('\n');*

*Serial.print("Steps= ");*

*Serial.println(steps);*

*distanceInCM = (averageStepLength \* steps);*

*distanceInKM = (distanceInCM / 100000);*

*distanceInM = (distanceInCM / 100);*

*}*

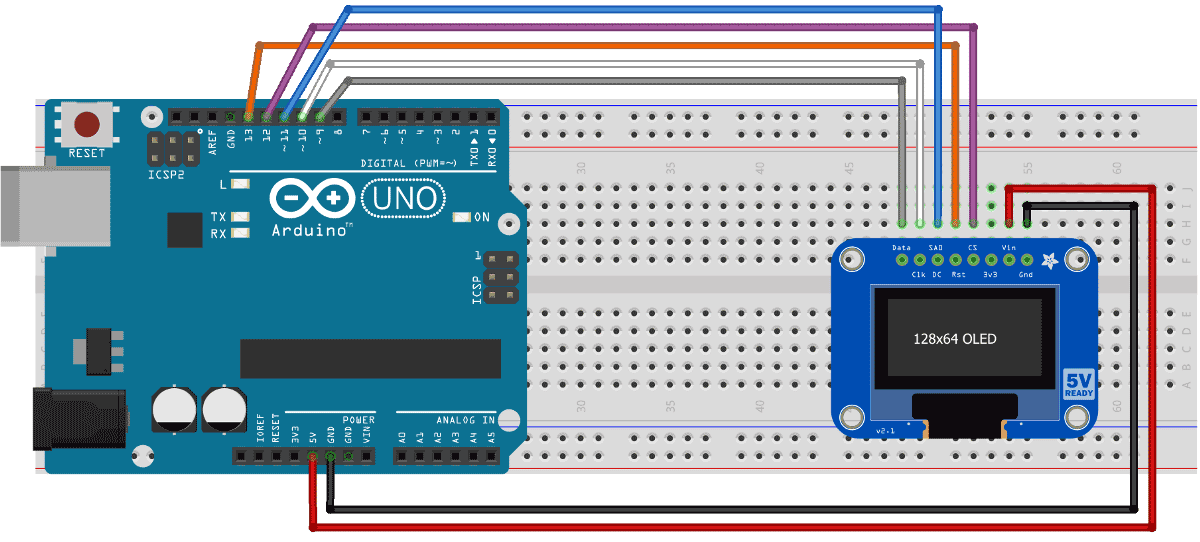


Figure 5.1.3: OLED interfacing with Arduino

Output:

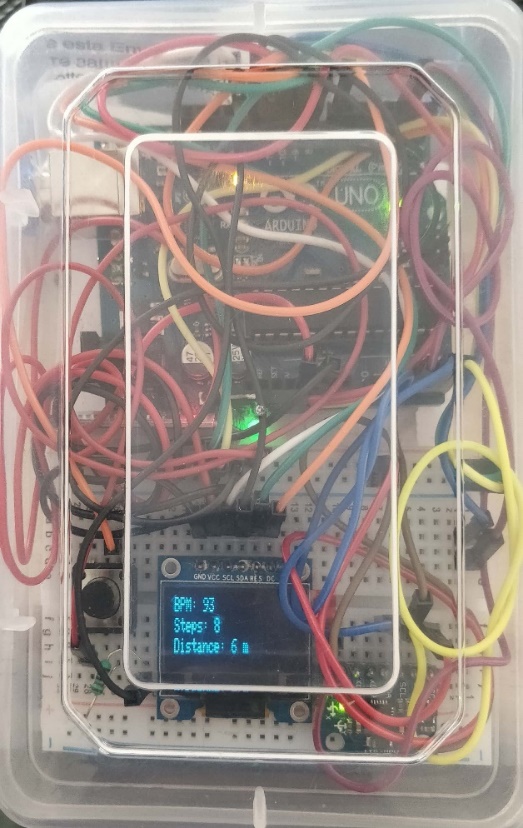


Figure 5.1.4: Interfacing of all devices with Arduino

**5.2 Emergency alert system**

In this module, as the name suggests that some action will be performed when an emergency is applied. So this system is basically made for women’s security as whenever a switch in this system will be triggered a message will be sent to selected contacts from android app stating with the current location of the victim. So in this system, in case of hardware only one switch is required. So this switch has 3 pins: Data pin, VCC and GND pin. Data pin requires digital input and produces digital output in the form of 0’s and 1’s. So whenever a switch is clicked or pressed it produces 1 as output and 0 when it is not pressed. This module will be working only when band is connected to android app via Bluetooth. Pseudo code:

*int stateButton = digitalRead(pinButton); //read the state of the button*

*if (stateButton == 1) { //if is pressed*

*mySerial.print(";");*

*mySerial.print("sw"); //data sent to android app*

*}*

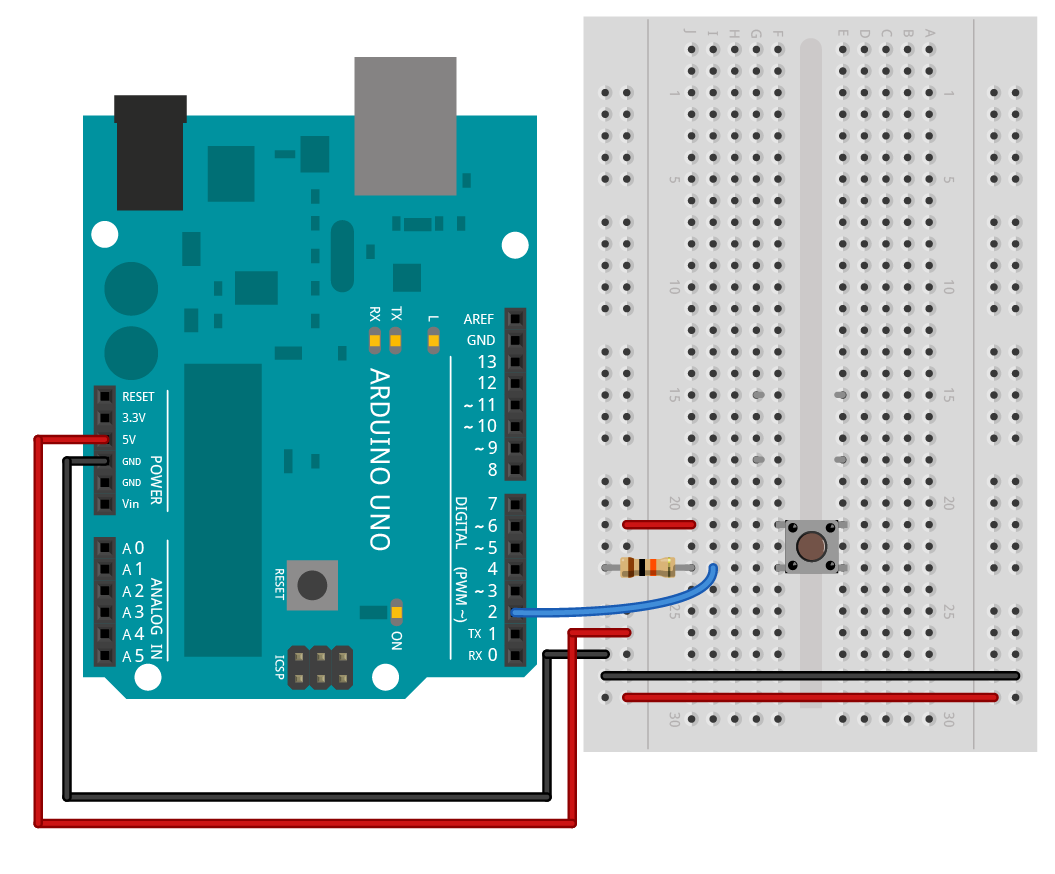
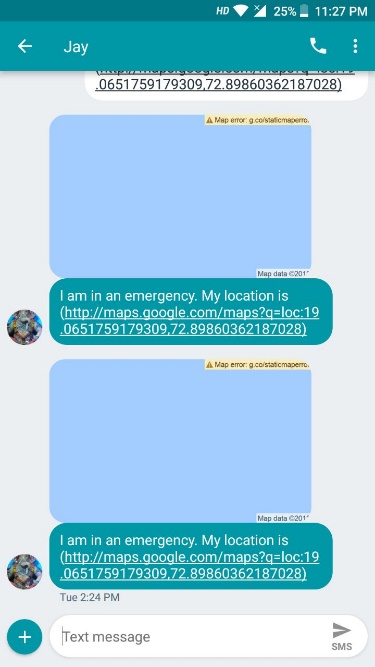


Figure 5.2.1: Switch button interfacing with Arduino

Output:

  
Figure 5.2.2: Message received on Android app

**5.3 Behaviour Tagging in Home automation**

In this module, we have tried to make Home automation easier with the help of behaviour tagging. So, behaviour tagging is a feature in which an action can be performed based on the gestures made. So, with the help of this our Home can be controlled with simple gestures of which it will be easier to control.

Now for this, we have made two separate hardware one will be in the band and another will at our home of which our home appliances will be connected. So, for this we have used MPU6050 for determining the gestures of the user which can be identified by gyroscope sensor. This will be inside band and will send data according to home side hardware system. This can be only achieved when they are connected via Bluetooth with each other. Pseudo code of Hardware at home:

*if (BTserial.available() > 0) {*

*String data1 = BTserial.readStringUntil(',');*

*Serial.read();*

*String data2 = BTserial.readStringUntil(',');*

*Serial.read();*

*String data3 = BTserial.readStringUntil('|');*

*valx = data1.toInt();*

*valy = data2.toInt();*

*valz = data3.toInt();*

*if (valx < 1000 && valy < -8000) {*

*Serial.println("BACKWARD");*

*digitalWrite(led1, LOW);*

*digitalWrite(led4, LOW);*

*}*

*if (valx < 1000 && valy > 8000) {*

*Serial.println("FORWARD");*

*digitalWrite(led1, HIGH);*

*digitalWrite(led4, HIGH);*

*}*

*if (valx > 8000 && valy < 1000) {*

*Serial.println("RIGHT");*

*digitalWrite(led2, LOW);*

*digitalWrite(led3, LOW);*

*}*

*if (valx < -8000 && valy < 1000) {*

*Serial.println("LEFT");*

*digitalWrite(led2, HIGH);*

*digitalWrite(led3, HIGH);*

*}*

Pseudo code of data sent via band:

*void sendDataToAnotherArduino() {*

*String btData = (String)gyroX + "," + (String)gyroY + "," + (String)gyroZ + "|";*

*Serial.print("Sending data");*

*Serial.print(btData);*

*mySerial.print(btData);*

*}*

In this we have implemented 4 kind of gestures of which some actions will be performed based on those gestures. So will demonstrate those by glowing leds.

Case 1: When there are no gestures made thus, all the leds will stay off as we are in steady state.

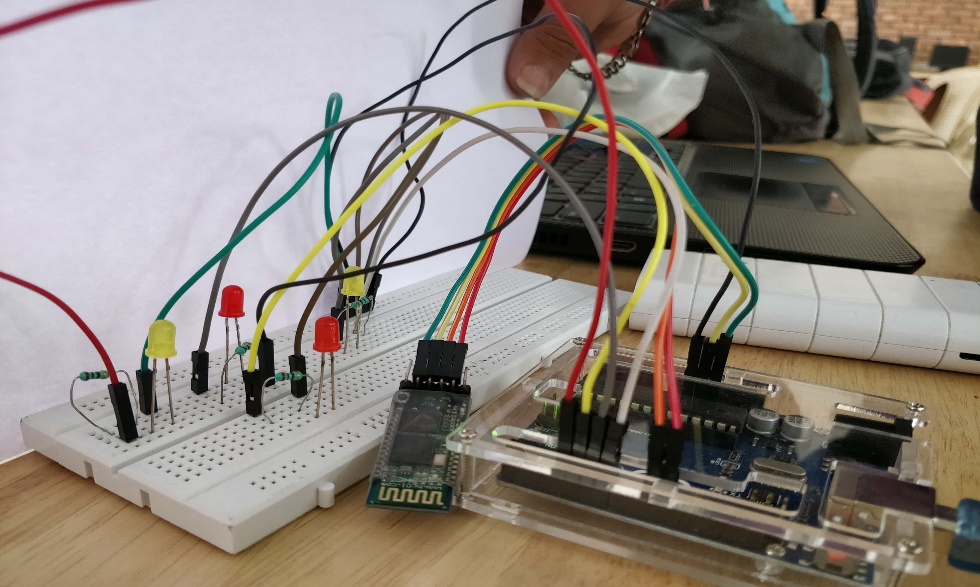


Figure 5.3.1: Steady state gesture output

Case 2: When the hand is moved to left side of which the gesture is performed, red led will glow as shown in below figure and when the hand is moved to right side, this gesture will turn off red led.

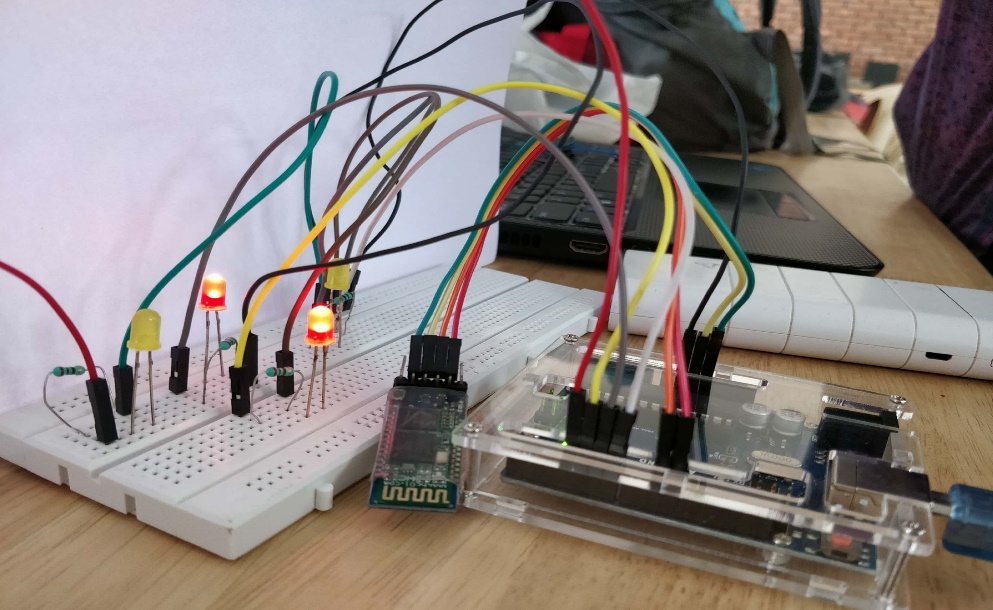


Figure 5.3.2: Gesture in left directon

Case 3: When will lean our hand in forward direction, this gesture will turn on yellow led as shown in below figure and when the wrist of hand is moved in backward direction, this will turn off yellow led.

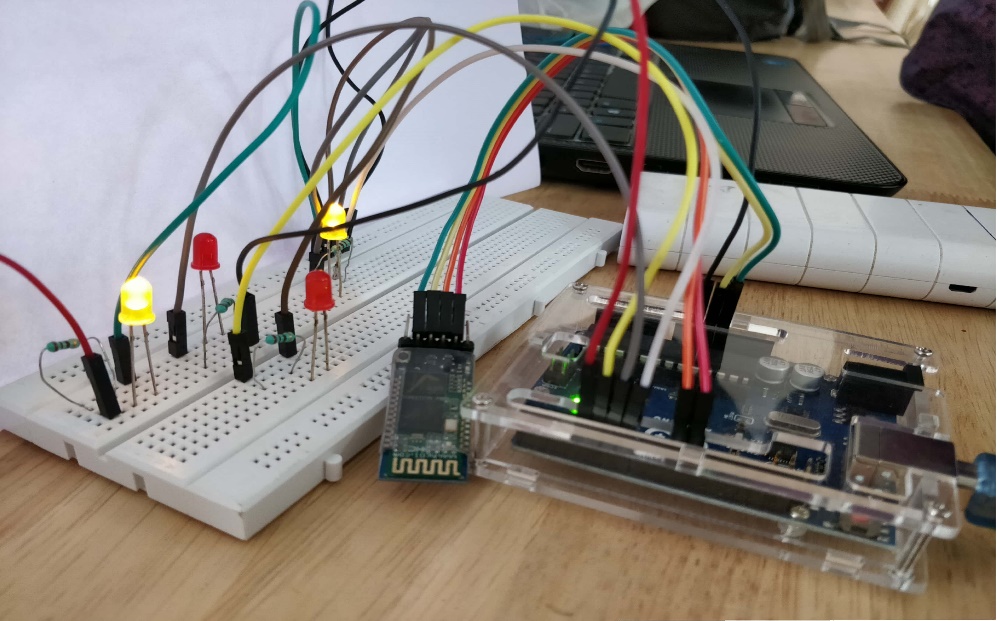


Figure 5.3.3: Gesture in Forward direction

Case 4: In this case, when both the gestures are performed simultaneously both led will glow and accordingly led will be turned off when appropriate actions or gestures are performed.

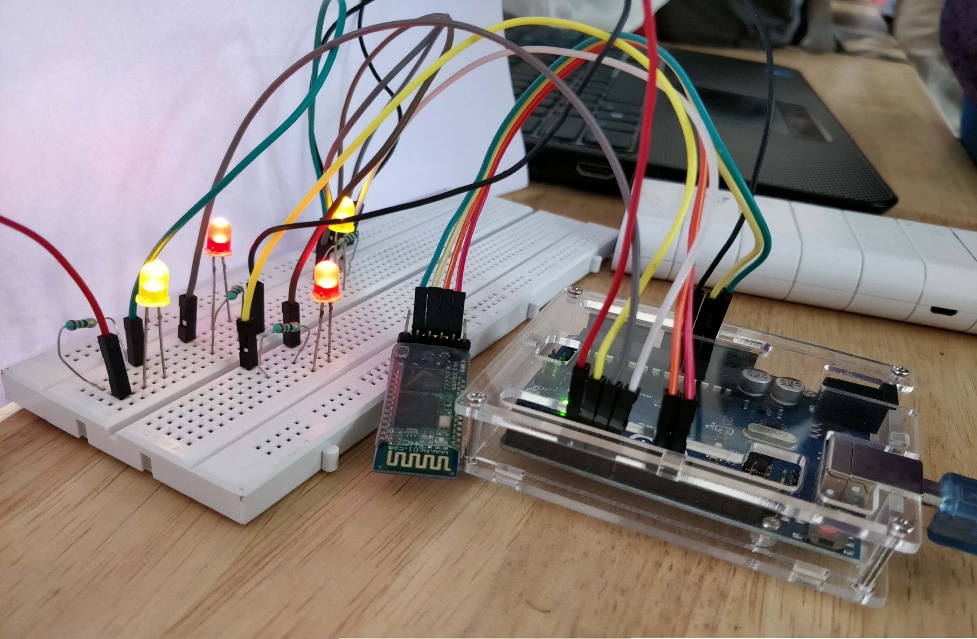


Figure 5.3.4: Output for simultaneous running of appliances

**5.4 Android app**

In this module, the result of user activity will be displayed so that the user can see his/her information regarding his/her activity performed on that day. And with the help of android app we are able to send emergency alert message to selected contacts. In this data received from Bluetooth is the input and output is also same as all the major processing is done on band. Pseudo code:

For receiving data from Arduino via Bluetooth:

*void beginListenForData() {*

*final Handler handler = new Handler();*

*stopThread = false;*

*buffer = new byte[1024];*

*Thread thread = new Thread(new Runnable() {*

*public void run() {*

*while (!Thread.currentThread().isInterrupted() &amp;&amp; !stopThread) {*

*try {*

*int byteCount = inputStream.available();*

*if (byteCount &gt; 0) {*

*byte[] rawBytes = new byte[byteCount];*

*inputStream.read(rawBytes);*

*final String string = new String(rawBytes, &quot;UTF-8&quot;);*

*handler.post(new Runnable() {*

*public void run() {*

*sendDataForSegregation(string);*

*// Log.d(TAG, &quot;run: Result: &quot; + string);*

*}*

*});*

*}*

*} catch (IOException ex) {*

*stopThread = true;*

*}*

*}*

*}*

*});*

*thread.start();*

*}*

For sending message from android app to saved contacts:

*private void sendEmergencyAlertMessage(String trigger) {*

*GPSTracker gpsTracker = new GPSTracker(this);*

*String latitude = String.valueOf(gpsTracker.getLatitude());*

*String longitude = String.valueOf(gpsTracker.getLongitude());*

*String[] stringsPhoneNumber = {&quot;9619337636&quot;, &quot;8693812076&quot;, &quot;9773340195&quot;};*

*String message = &quot;I am in an emergency. My location is (http://maps.google.com/maps?q=loc:&quot; + latitude + &quot;,&quot; + longitude*

*+ &quot;)&quot;;*

*Log.d(TAG, &quot;sendEmergencyAlertMessage: Location: &quot; + gpsTracker.getLatitude() + &quot;,&quot; + gpsTracker.getLongitude());*

*if (trigger.equals(&quot;sw&quot;)) {*

*SmsManager smsManager = SmsManager.getDefault();*

*for (String aStringsPhoneNumber : stringsPhoneNumber) {*

*smsManager.sendTextMessage(aStringsPhoneNumber, null, message, null, null);*

*}*

*} else {*

*Toast.makeText(this, &quot;Error while sending message&quot;, Toast.LENGTH\_SHORT).show();*

*}*

*}*

Output:

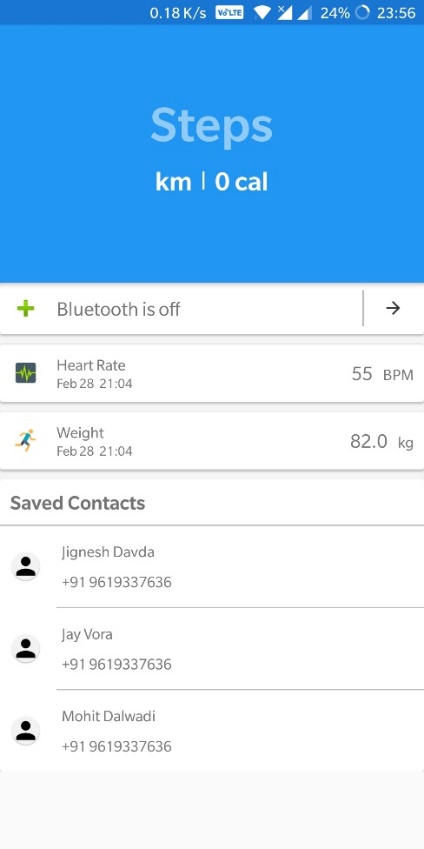
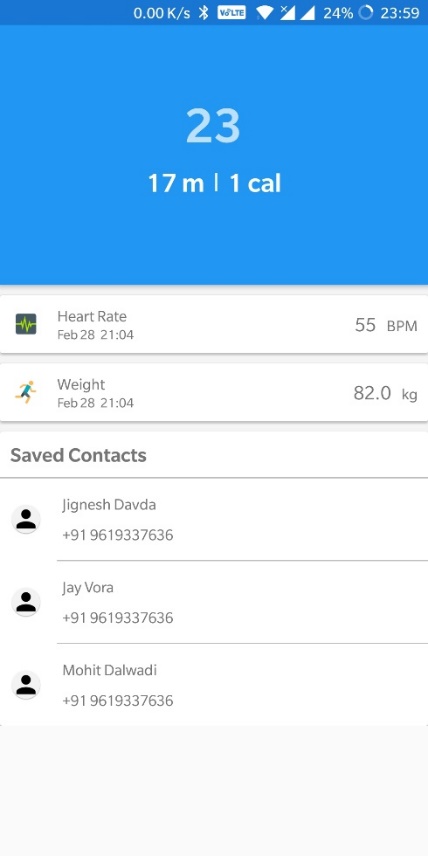
 

Figure 5.4.1: State when app not Figure 5.4.2: State when app

connected to band is connected to band