FINAL REPORT, GROUP 3, EE2 PROJECT, DEPARTMENT OF ELECTRICAL AND ELECTRONIC ENGINEERING, IMPERIAL COLLEGE LONDON

V. Cardazzi (CID:01351836), L. Goudelis (CID:01349472),
J. Grivas (CID:01359303), C. Gulli (CID:01352733),
D. Kazantzis (CID:01367700), C. Lu (CID:01337169),
E. Ozyilkan (CID:01358595)

Abstract. — Final design outcomes for *Over-Watch v1.0* are discussed. Implemented solutions for different functionalities are presented and the convenient decisions made are explained with their rationale. Group dynamics and further possible developments are provided.

Title of the Project. — Over-Watch v1.0

Submission Date. — 22/03/2019

Supervisor's Name. — Dr. Imad M. Jaimoukha

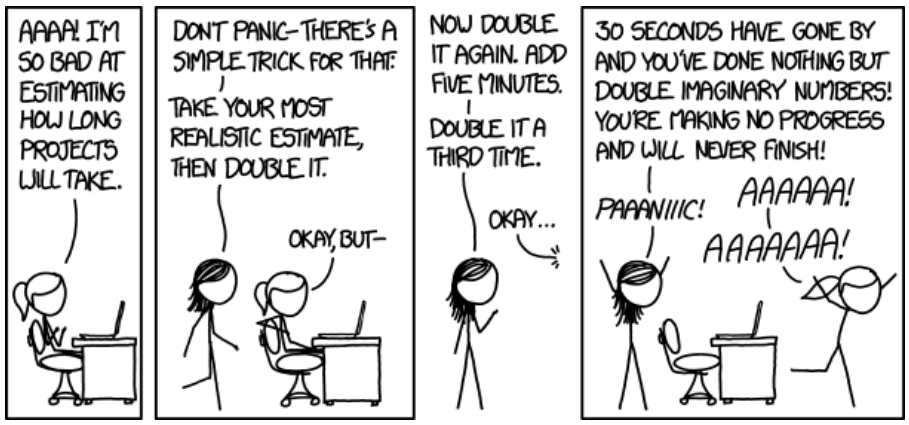


Figure 1. Estimating Time [1].

Contents

1. Abstract

This document is the description of the prototype of a smart watch aimed for elderlies. The product is implemented to improve old people's lives by monitoring their health conditions. It does so by communicating with a website which can be accessed by people that care for the elderly that is wearing the watch. Values for heart beat and oxygen concentration in the blood are displayed in this webpage, together with different levels of danger which give information on possible health risks that could arise when changes in these levels are experienced. Moreover, the watch detects sudden movements (i.e. falls) and the relevant information is, again, shown in the website. The smart device also has a medicine reminder which aims at making the wearer remember to take his daily pharmaceuticals. The time when this reminder goes off is set from the website. Another feature of the prototype is a GPS tracker which allows to know the current location of the wearer from the webpage.
These features all together allow people who care for the relative to be informed of the elder's possible health risks, as well as make the old person more independent since, wearing the watch, there is no need for him to be constantly accompanied by someone.

2. Introduction

Many elderlies live alone and, in cases of emergencies, it often happens that they are unable to access immediate help. It is proven that 28% of people above 65 years old live alone and feelings of loneliness can negatively affect both physical and mental health [8]. As a result, one study [9] found that older adults who were reported to feel lonely had 59 % higher risk of mental and physical decline along with a 45% increase in their risk of death. Moreover, another common encountered problem amongst them is falling and not being able to stand up, having no way to call for help. It has also been observed that the heart rate gives good guidance to detect health-related problems [5]; this is relevant because sudden variations of the heartbeat could lead to permanent damage. In fact, it has been shown that an increase in the heart rate by 10 beats per minute could be associated with an increase in the risk of cardiac death by at least 20% [10]. These heart related problems include heart attacks, convulsions and seizures. To further enhance the importance of monitoring the heart rate, in has been proven that in the UK heart diseases cause more than a quarter (26%) of all deaths, which is an average of 435 people each day or one death every three minutes [11]. In addition, the oxygen concentration level in the blood is another aspect that can be of help in monitoring one's health. Low levels can cause various severe problems, such as hypoxemia.

Another big problem that old people often encounter is having sudden memory loss. This can have a huge impact on their everyday life: It might result in hazardous situations where they lose their way around in the neighbourhood and thus are not able to get home safely. Furthermore, elderlies frequently need to take pharmaceuticals on a daily basis, and it is of vital importance that they remember to do so.

These needs of monitoring health levels and reducing hazardous situations that could arise due to memory loss can be accommodated with a smart device worn by the elderly. This smart watch detects sudden movements (i.e. falls) and heart rate and oxygen levels, in order to prevent serious and permanent physical damages.

Furthermore, a GPS keeps track of the elderly's location and a built-in medication reminder can be programmed from a website to remind them to take these. This device communicates with an easily accessible website so that pre-determined contacts (i.e. close relatives) would immediately be informed.

Overall, this smart watch should help elderlies become more independent and safer. This project has the potential to protect lives and prevent fatal diseases. Therefore, the market for this product is potentially very large since it could improve the health standards of the global society.

In the prototype developed, named as *Over-Watch v1.0*, this is achieved by:

- detecting sudden movements with the triaxial accelerometer ADXL345
- monitoring heartbeat and oxygen levels with heartrate sensor and oximeter MAX30100
- sending information via GSM connection to the website via SIM808
- having a medicine reminder via a DC motor vibrating
- having a GPS to keep track of the current location

Figure 2 shows the overall connection between the modules that constitute the watch.

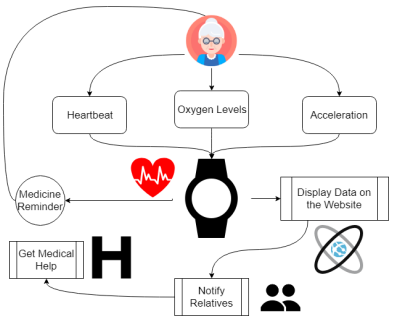


FIGURE 2. Overall functionalities of *Over-Watch v1.0*.

3. Concept Development - Technical Functionalities

3.1. Sensors. — The prototype of the smart watch uses a heart beat sensor and pulse oximeter and an accelerometer. These sensors are controlled using the Arduino board through I2C connection.

3.1.1. ***Heart rate Sensor and Pulse Oximeter.*** — The reason behind the choice of using a heartbeat sensor in designing the watch is that about 1 in 4 deaths occur because of heart related problems [6]. By measuring the heartbeat and the oxygen concentration, a number of these episodes can be detected promptly and thus prevented from having fatal consequences.

Hardware Choice. — As illustrated in the Preliminary Report, the group considered a number of sensors before choosing the appropriate one. The two sensors that were considered optimal and tested are:

- *Maxim Integrated MAX30101 High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health*
- *Maxim Integrated MAX30100 Pulse Oximeter and Heart-Rate Sensor IC for Wearable Health*

These sensors are very similar, MAX30101 being the new version of the MAX30100. However, for a number of reasons, the group opted for the second alternative. First of all, the MAX30100 was released in September 2014 while MAX30101 in October 2018. Because of the fact that the first has been in the market for much longer, there is a lot of accessible material available to be used as guidance for its programming and use. Moreover, for the purpose of the product, the additional sensitivity provided by the newer version is not essential. This is because this would have been virtually imperceptible to the user given the relatively low computational capability of the Arduino on which the data processing was implemented.
In order to achieve easy connection with the Arduino and since the final product is a prototype (where the space constraints are not strict) the sensor was bought as part of an integrated circuit, the *MikroElektronika MIKROE-2000 mikroBus Click Board*. Figure 3 shows the integrated circuit of the heart beat sensor and oximeter.

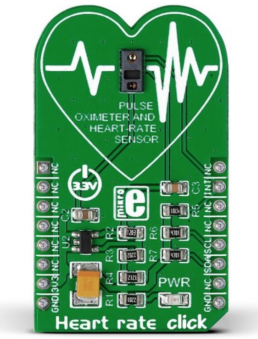


FIGURE 3. MAX30100 sensor with MikroElektronika MIKROE-2000 Click Board

Functioning. — The MAX30100 is a pulse oximetry and heart rate sensor. It is an optical sensor that derives its readings from emitting two wavelengths of light from two LEDs, then measuring the absorbency of pulsing blood through a photodetector. The signal is processed by a low-noise analog signal processing unit and communicated to the microprocessor through the I2C interface. It uses 3.3V power supply [7].

Arduino Code. — In the Arduino `setup()` the operations that need to be performed just once at the beginning of the program are executed:

1. the sensor in initialised with the desired settings;

2. the message "Place your finger on the sensor" is displayed on the serial monitor and on the display.

The operations performed in the `loop()` are constantly repeated until the program is stopped:

1. the necessary settings of the sensor are updated with `pox.update()`;
2. if the average calculation is not complete, the values of heart rate and oxygen concentration are taken every 0.5 seconds by calling the functions `pox.getHeartRate()` and `pox.getSpO2()`;
3. the average of ten realistic values is calculated by calling the function calculate_average;
4. if the average calculation is complete, the values are displayed on the screen placed on the top layer of the watch and the serial monitor, in addition to being sent to the website. The variables are then reinitialised.

The relevant Arduino code is shown below. Refer to the appendix (page 27) for the complete code.

```
void calculate_average(int beat, int SpO2) {
    if (not calculation_complete and beat>40 and beat<250) {
        average_beat = filterweight * (beat) + (1 - filterweight ) * previous_average_beat;
        previous_average_beat = average_beat;
        average_SpO2 = filterweight * (SpO2) + (1 - filterweight ) * previous_average_SpO2;
        previous_average_SpO2 = average_SpO2;
        readIndex++;
        filterweight = 1/(readIndex+1);
    }
    if (readIndex==numReadings) {
        calculation_complete=true;
        calculating=false;
        initialized=false;
        readIndex=0;
        filterweight=1;
        display_values(); //display the values & the levels of alert on the screen and the serial monitor
    }
}

void setup(){
    //[...other code...]
    Serial.begin(9600);
    pox.begin();
    pox.setOnBeatDetectedCallback(onBeatDetected);
    initial_display(); //display the "Place your finger on the sensor" on the screen and the serial monitor
    //[...other code...]
}

void loop(){
    //[...other code...]
    pox.update();
    if ((millis() - tsLastReport > REPORTING_PERIOD_MS) and (not calculation_complete)) {
        calculate_average(pox.getHeartRate(),pox.getSpO2());
        tsLastReport = millis();
    }
    if (calculation_complete){
        getRequestHeart(average_beat, average_SpO2); //send the data to the website via SIM808
        calculation_complete=false;
        average_beat=0;
        average_SpO2=0;
    }
    //[...other code...]
}
```

Processing. — The processing that takes place is aimed at obtaining measurements which are as accurate as possible:

- only values within a range of 40 to 250 BPM are considered. Any value below or above these thresholds is not realistic as a heart rate and is therefore rejected. Testing proved that neglecting out-of-range values considerably increases the noise rejection;
- an average between 10 realistic values is taken. A running average is used, meaning that it is constantly updated as more data points are collected rather than being calculated after all the data have been gathered. Only the resulting value is displayed and sent to the website.

On top of constantly displaying and transmitting to the webpage the current heart rate and oxygen concentration values, the device is aimed at producing levels of alert to advise if the values are outside a normal range and could potentially be a sign of danger.
As far the the heart rate is concerned, there are 3 levels of alert [12]:

1. heart rate between 100 and 150 BPM;
2. heart rate between 150 and 200 BPM: the first two ranges are values that a healthy young person encounters in situations of physical or emotional stress. In an old person they can be dangerous but are not necessarily a sign of medical emergency;
3. heart rate above 200 BPM: values above this threshold are always a sign of medical condition and should be treated immediately. Even in a young healthy individual such values would be atypical. By testing within the members of the group (that comprises of 7 people aged 19 to 20 in good health conditions) it was indeed impossible to get vales above 180 BPM. A screen of the serial monitor is shown in the appendix in page 45.

In the case of the oxygen concentration, there are 2 levels of danger:

1. oxygen concentration between 90 and 95%: this range of values is not ideal but still within the norm;
2. oxygen concentration below 90%: pulse oximeter readings below this threshold can be a sign of hypoxemia and are very risky if persistent. They should receive immediate medical attention.

Having different levels of alert allows to find a balance between detecting the critical situation and not having en excessive number of false positives. In fact, the low levels of alert will generate a lot of false positives but virtually always detect a dangerous situation, while the highest levels of alert are very unlikely to give false positives but risk to miss out on detecting a danger.

Testing. — Testing showed that the readings achieved in this way are overall reliable, given a set of conditions:

- The pulse oximeter settings need to be updated constantly. When incorporating the sensor's code into the overall Arduino code, this had to be taken into account. As further explained in the "Overall Arduino code" section (page 17), the `pox.update()` function was called before and after each computationally costly operation (which takes more time to run) and when introducing delays;
- Since the output relies on taking the average of ten consecutive values, one's finger tip has to touch the sensor for the whole duration of the process. If the user places the finger on the sensor after the average has started or removes it before the calculation has finished, the result is not accurate;
- The sensor has to be gently touched since much pressure can constrict capillary blood flow and therefore diminish the reliability of the data.

3.1.2. ***Accelerometer***. — The purpose of this sensor is to detect sudden movements of the elderly wearing the watch, which might be a sign of fall. The accelerometer is able to detect the device's acceleration in three directions through taking into account influence of gravity, angular movement and linear motion. The accelerometer is able to respond to the acceleration associated with movement in all three axis, x, y and z. The data of interest will therefore be a sudden increase in the velocity at which the person is moving.

Hardware Choice. — For the hardware implementation, the group decided to use the digital accelerometer ADXL345 (shown in figure 4) because it offers:

- Very low power: $23\mu A$ in measurement mode and $0.1\mu A$ in standby mode when using a 3.3V voltage input;
- User-selectable resolution;

FIGURE 4. ADXL345: triaxial accelerometer

Processing. — In order to detect a fall, the following steps are carried out at each iteration of the `loop()`:

1. the values of the current acceleration along the x, y and z axes are saved in the variables *newx, newy, newz*;
2. the difference between the acceleration value at the previous iteration of the loop and the current one is saved in *diffx, diffy, diffz*;
3. if the difference is above a certain threshold (further discussed in the "Testing" section below), a fall alert is sent;
4. the current acceleration is saved in *prevx, prevy, prevz* so that it becomes the previous value for the next iteration of the loop.

The reason why the difference is used to detect the fall is that this is an efficient and reliable way to calculate a variation that requires very low computational effort. Moreover, the raw data has an offset around which the values oscillate that is cancelled by performing the subtraction.

Testing. — After various procedures of testing, it was concluded that if the magnitude of the gradient variables (*diffx, diffy, diffz*) is bigger than $10m/s^2$, this indicates that the sudden acceleration of the wearer is high enough to indicate a fall.

It is indeed important to be able to distinguish among a wide range of values obtained with the sensor: for example, when the wearer walks vs. when the user falls down. The following graphs explain the reasoning behind the choice of the threshold value. Figure 5 indicates the range of values obtained when the user walks. As can be observed, when walking or waving the acceleration is found never to exceed the range -9.5 $<$ *diffx, diffy, diffz* $<$ 9.5 .

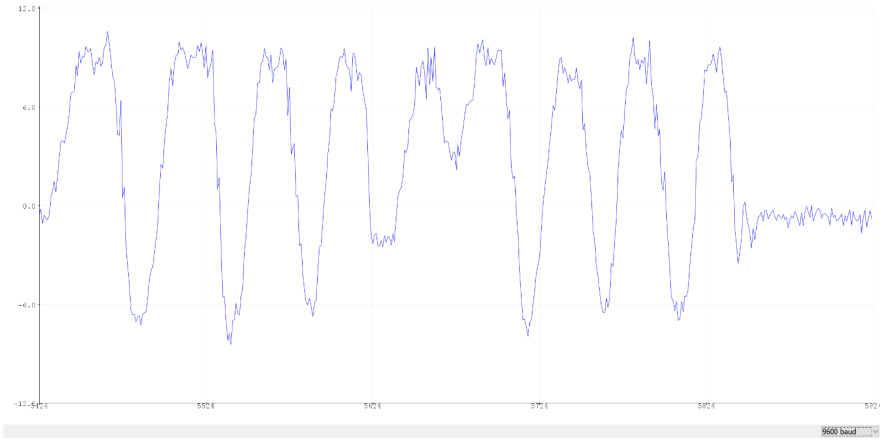


FIGURE 5. Acceleration in three axes: user walking

When testing the device during a fall (i.e. sudden acceleration), the graph in Figure 6 was obtained. The amplitude of the peak is around $12m/s^2$. Hence it is confirmed that if the user falls, at least one of the three new variables (*diffx, diffy, diffz*) will exceed the chosen threshold of $10m/s^2$.

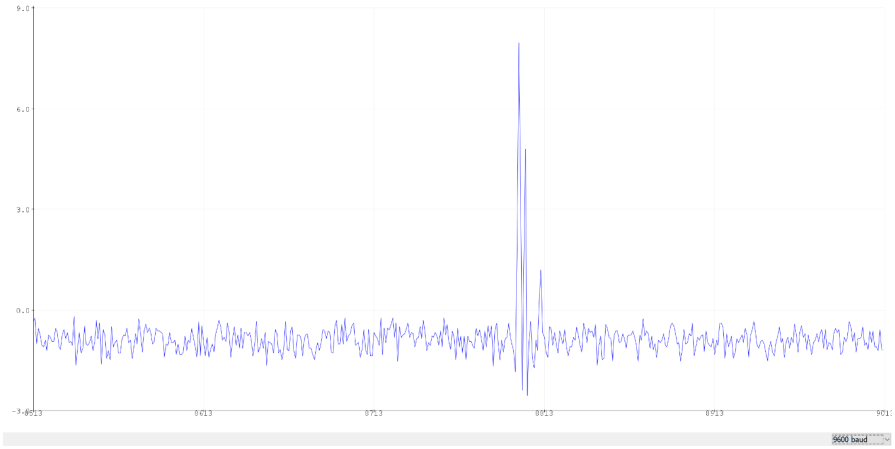


FIGURE 6. Acceleration in three axes: user falling

Coding. — In the `setup()` of the Arduino code, the following operations are performed:

- check that the sensor is connected, otherwise suggest to check the wiring;
- set the appropriate range of operation for the device;
- display general informaiton about the sensor.

In the `loop()`, the collection of data and processing described in the previous sections is implemented. The code is shown below.

```
void loop(){
    //[...other code...]
    sensors_event_t event; //Get a new sensor event

    newx = event.acceleration.x;
    newy = event.acceleration.y;
    newz = event.acceleration.z;
    diffx = newx - prevx;
    diffy = newy - prevy;
    diffz = newz - prevz;
```

```
    prevx = newx;
    prevy = newy;
    prevz = newz;

    //Processing: is it a fall?
    if (diffx > abs(10) || diffy > abs(10) || diffz > abs(10)) {
        Serial.println ("FALL DETECTED");
        last_fall = millis();
        getRequestAccel(11); //fall information is sent to the website
        fall_sent = false;
    }
    if(millis() - last_fall > 10000 && fall_sent == false){
        getRequestAccel(0); //the fall alert on the website lasts 10 seconds
        fall_sent = true;
    }
    //[...other code...]
}
```

3.2. Medicine Reminder. — Since most elderlies need to take pharmaceuticals on a daily basis to maintain their health, it is of vital importance that they remember this. To accommodate for this need, the concept of medicine reminder is developed in this product.

Overall concept. — The reminder is a feature that notifies the elderly when it is time to take medications. After considering different ways for notification (e.g. light, sound, vibration), vibration is finally chosen for implementation since it the most reliable. This is both because the vibration can't be missed and because it is very hard for the circuit to be damaged. The information (i.e. time) for this notification can be set from the website, which is easily accessible from any computer and phone. Hence, the relatives of the elderly can monitor and set the time for a medication to be taken.

Hardware choice. — Two different devices which produce vibration are considered: a DC motor and a vibrator. In reality, the vibration functionality could be achieved by simply supplying some power to the motor, but the vibration obtained wouldn't be strong enough.
Since the product is a prototype, the group decided to opt for the DC motor option, although knowing that its weight could affect negatively the wearability requirement of the watch. The vibration is achieved by attaching a weight to the motor's shaft. The mass of the whole device will therefore be distributed in an improper way so that the off-centre mounting will lead to vibration.

Circuit Principle. — The reminder feature is controlled by the Arduino so that the motor vibrates three times when it is the time to take medicines, set externally on the website. Because Arduino pins only allow 40 mA current to pass through, this motor can't be connected to the Arduino directly. Another possible inconvenience is that the motor is capable of generating a counter EMF, which could damage the Arduino Board. Thus, a driver circuit is implemented to protect the Arduino and to easily control the motor (figure 7).
The transistor (i.e. BC337 NPN) acts as an electrically operated switch to turn the motor on and off depending on the voltage applied to its base by the Arduino itself. The diode utilised is a fly-back diode: It forces the current to flow to the motor so that it protects the Arduino board. When the Arduino gives a 5V input, current will flow through the transistor so that the diode becomes reverse-biased. After this signal has stopped, the DC motor will generate reverse current so that the diode will be forward-biased, blocking the current from flowing back to the Arduino. A 1.5V battery is used as a voltage source to power the motor.

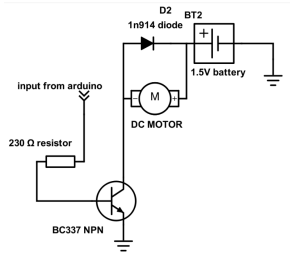


FIGURE 7. Circuit for Medicine Reminder

Coding. — In the `setup()` of the Arduino code, digital pin 22 is set as an output pin. In the `loop()`, the code makes the output pin go high (5V) for 1 second, and low (0V) for 1 second, repeating this alternation three times. This part of the Arduino code is shown below.

```
int motor_pulse = 0;
uint32_t last_motor = 0;
bool sick = true;

void setup(){
    //[...other code...]
     pinMode(22,OUTPUT);
    //[...other code...]
}

void loop(){
    //[...other code...]
    if (hour()==hour_med && minute()==minute_med && sick==true){
        while(motor_pulse < 3){
            digitalWrite(22,HIGH);
            last_motor = millis();
            while ( (millis() - last_motor) < 1000){
                pox.update(); //the settings of the heartbeat sensor need to be updated when a delay is
                //introduced in the code, as explained in the "Overall Arduino Code" section
            }
            digitalWrite(22,LOW);
            last_motor = millis();
            while ( (millis() - last_motor) < 1000){
                pox.update();
            }
            motor_pulse++;
        }
        sick=false;
        motor_pulse=0;
    }
    if (minute()==(minute_med+1)){
        sick=true;
    }
    //[...other code...]
}
```

3.3. Website User Interface. — In order to monitor elderlies' health, a website is built to display the relevant data obtained via the watch. This shows the GPS location of the wearer, levels of heart rate and oxygen concentration, falls and provides the opportunity for the relatives to set the time parameter for the medicine reminder. The website is divided into the four parts, referring to four different functionalities of the product. Each data segment is refreshed at an interval of 2 to 5 seconds. Figures 8 and 9 are screen shots of the website.

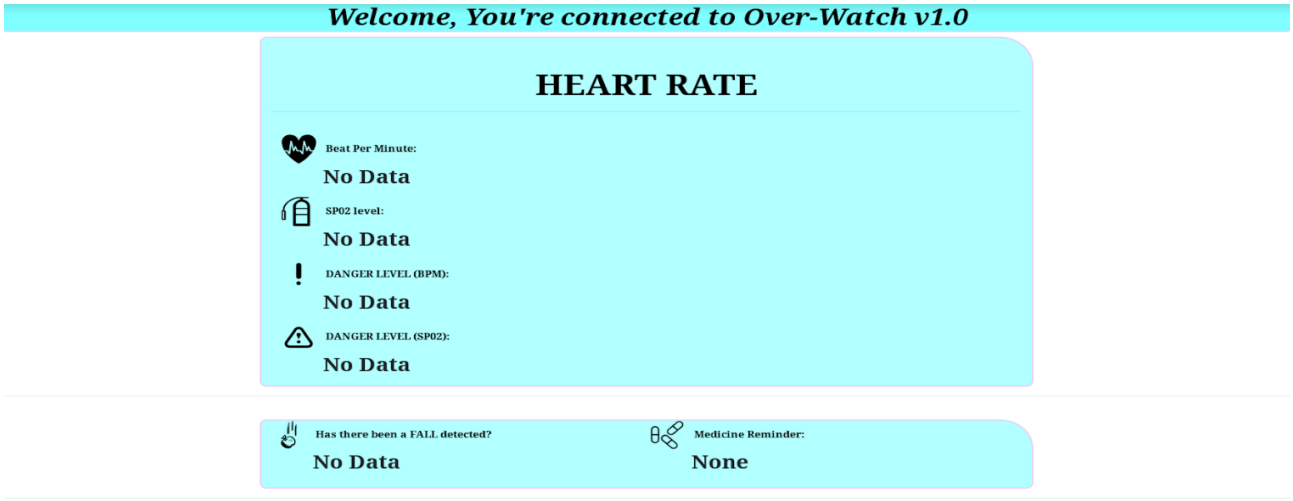


FIGURE 8. Web User Interface (1)

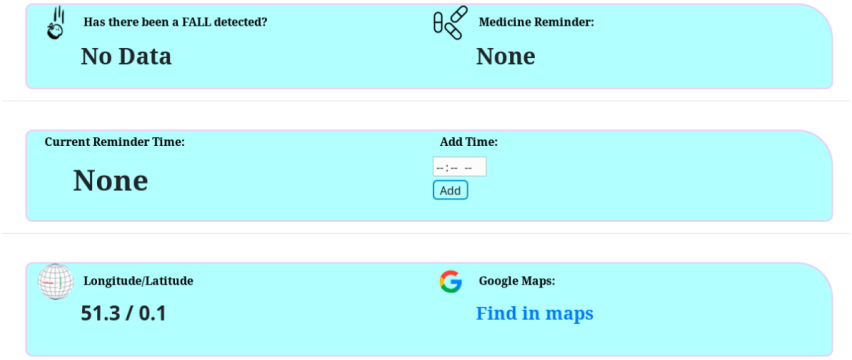


FIGURE 9. Web User Interface (2)

- The first section is dedicated to readings form the pulse oximeter and heart rate sensor. Four different sets of data are displayed in this part: Beat Per Minute, SP02 level, Danger Level (BPM), and Danger Level (SP02). Refer to the "Heart rate sensor and pulse oximeter" section (page 5) for a more accurate explanation on the levels of danger.
- The section about accelerometer's data follows. If there is a fall, *NO DATA* is replaced with *DANGER* in this segment (figure 9) for an interval of 10 seconds. After this interval, *NO DATA* is displayed again so that the field is emptied to receive a new alert. However, this is not the optimal implementation to display a fall, therefore improvements are discussed in the "Connection & Website user interface" section in "Further Developments", page 20.
- For the medicine reminder, the time for alarms to go off could be set in the form of (*hour : minute pm/am*). All notifications set up can be seen in the *Current Reminder Time* section.
- The last section is GPS tracking. The longitude and the latitude of the GPS location is demonstrated on the left hand side. Via clicking on *Find in maps*, the relatives could access the location of the elderly on the Google Maps.

Coding. — The website was made using HTML as its markup language, CSS as its style sheet and JavaScript as its programming language. As the HTML and CSS are solely for the contents and styling of the page, this part will not be explained but can be found in Appendix section in page 40. Besides dealing with the connection between modules, the JavaScript code contains several functions which

add responsiveness and functionality to the UI. The *calculateDanger* function takes the sensor values as input and displays them in the website. It also compares the values with the thresholds in order to set the relevant levels of danger.
In addition, there are several functions regarding the medicine reminder, which enable it to function in a user-friendly way. The *displayMedicineReminder* function loops through all the current reminders and selects the one that is closest to the current time. The other functions are used for adding and removing reminder times. Finally, *setInterval* updates the website every 3 seconds by making continuous requests to the server in order to return the most recent sensor values. Figures 11 and 12 show the main parts of the JavaScript code. Refer to the section "Website Javascript code" in the Appendix for the complete code (page 36).

```
function calculateDanger(heart, oxy, acc){
    appendString = '<h2 class="box_output">' + heart + '</h2>';
    $( 'div.beatHolder' ).html(appendString);
    appendString = '<h2 class="box_output">' + oxy + '</h2>';
    $( 'div.oxygenHolder' ).html(appendString);
    if(heart >= 100 && heart <= 120){
        $( 'div.beatDanger' ).html('<h2 class="box_output">DANGER Level 1!</
        h2>');
    }
    else if(heart > 120 && heart <= 150){
```

FIGURE 10. calculateDanger function

```
function displayMedicineReminder(){

function printNone(){

function addTime(){

function actualRemoveTime(index){

function removeTime(){
```

FIGURE 11. Functions employed for website UI

```
setInterval(function(){
    let user_hour = medicineHours[correctIndex];
    let user_minute = medicineMinutes[correctIndex];
    socket.emit( 'my event', {
            minutes : user_minute,
        hour : user_hour
  })
}, 3000);
```

FIGURE 12. setInterval timing event

3.4. Connection between modules. — In order for individual module components to communicate with one another, several methods of communication are set up. The result of the processing from the Arduino has to be sent over such that all of the health data can be displayed on the user interface (i.e. website). This is done through an HTTP request from the SIM808. The SIM808 makes a request to Python Flask server, which receives the given parameters and saves them in its database. The very same request retrieves data regarding the medicine reminder and, this time, returns it to the watch. On the other hand, the back-end of the user interface also makes HTTP requests to the server in order to retrieve the information stored so that it can display them on the website.

SIM808 connection. — The Arduino is connected to the SIM808 through *Software Serial Connection* and controls it through AT commands [1]. These commands are built-in on the SIM808 and either enable some configuration features (e.g. initialise its ability to make HTTP requests, send power to the GPS or put it to sleep mode etc.) or return data about SIM808 to the Arduino. [2] In the initial setup, the modem tries to get a connection automatically and the AT+CREG command returns either 0 or 1, where 0 implies GSM connection and 1 implies *registration* (meaning it has cellular connection). After that, APN (i.e. Access Point Name) parameters should be set up. For the SIM card, the parameter *pp.vodafone.co.uk* was used for the APN and *wap* for the username and password.

Following these steps, 3G connection was enabled and the modem was made ready to make HTTP requests. Every time data is to be transferred, parameters are passed to the URL with the AT+HTTPPARA command before making the request. The modem is able to setup the GSM connection with its GPRS antenna. The data is transferred in JSON format through which it is easy to work in both the transmitting and receiving end. The commands AT+HTTPACTION and AT+HTTPREAD make the request and read the value returned from it. Every AT command gives its response to the GPRS buffer. This is then retrieved by the *toSerial* function. Additionally, the *toSerial* function checks whether it finds the value returned by the *GET request* and stores it in the *constant* array which is used for the medicine reminder.

These parts of the program are shown in figures 13 and 14.

```
#include <SoftwareSerial.h>
SoftwareSerial GPRS(10,11);
```

(A)
Initialisation between SIM808 and Arduino

```
void sendGSM(const char* msg, int waitMs = 500) {
 GPRS.println(msg);
 delay(waitMs);
}
```

(B)
sendGSM function

```
sendGSM("AT+CREG?");
sendGSM("AT+SAPBR=3,1,\"APN\",\"pp.vodafone.co.uk\"");
sendGSM("AT+SAPBR=3,1,\"USER\",\"wap\"");
sendGSM("AT+SAPBR=3,1,\"PWD\",\"wap\"");
sendGSM("AT+HTTPINIT",1000);
sendGSM("AT+SAPBR=1,1",3000);
```

(C)
SIM808 set-up function (to connect to the Internet and make requests)

FIGURE 13. Arduino Header/Functions for the SIM808 (1)

1. AT commands are instructions that are used to control a modem. AT is the abbreviation of ATtention. Every command line starts with *AT* or *at*. That is why modem commands are called AT commands.

2. The manual explaining each of those commands can be found in the link: https://www.elecrow.com/wiki/images/2/20/SIM800$_{S}eries_{A}T_{C}ommand_{M}anual_{V}$1.09.*pdf*

```
sendOver = "AT+HTTPPARA=\"URL\",\"http://30ebc087.ngrok.io/post\"";
sendGSM(sendOver.c_str());
delay(1000);
toSerial();
sendGSM("AT+HTTPACTION=0");
delay(4000);
toSerial();
delay(500);
sendGSM("AT+HTTPREAD");
toSerial();
```

(A) Function for sending data from sensors to the server

```
void toSerial() {
 int i = 0;
 while(GPRS.available()!=0) {
  if( GPRS.peek() == '['){
   flag=2;
   constant[i] = GPRS.read();
  }
  else if(flag == 2 && i  != 5)
   while(GPRS.peek() != ']'){
    constant[i] = GPRS.read();
    i++;
   }
  else if(flag == 0){
  Serial.write(GPRS.read());
  }
  else{
   flag = 0;
  }
```

(B) toSerial function: reads the values returned by AT commands and HTTP requests

FIGURE 14. Arduino Functions for the SIM808 (2)

Server. — Another important factor in the connection is the server. It is a Python Flask server with a SQLite database. It has several routes for different requests from the SIM808 and the UI. Three of those routes are */Heart*, */Accel* and */Time*. The */Heart* updates the values for the heartrate and the oxygen levels. */Accel* updates the accelerometer value. However, */Time* does not update any value as it is only used when the time is wanted to be retrieved without updating any sensor values. On the other hand, the server also has a *socketio* route for the connection with the UI. It receives information for the medicine reminder which is given as a parameter to the *render_template* function, which will update the notification time which will be sent to the SIM808. Additionally, it sends back the most recent data which had been sent by the SIM808. For testing purposes, the server was hosted using *serveo*, which allows port forwarding to the server and thus, also hosts the server with a temporary URL. Figures 15 and 16 show this parts of the coding.

```
def report_heart():
    jsoninit["heartbeat"] = request.args.get('heartbeat')
    jsoninit["oxygen"] = request.args.get('oxygen')
    print(jsoninit)
    return render_template("post.html")

@app.route('/Accel',methods=['GET', 'POST'])
def report_accel():
    jsoninit["accelerometer"] = request.args.get('accelerometer')
    print(jsoninit)
    return render_template("post.html")

@app.route('/Time',methods=['GET', 'POST'])
def report_time():
    print(jsoninit)
    return render_template("post.html")
```

FIGURE 15. Routes for the Sim808 GET requests. Specific values are updated depending on the route

```python
@socketio.on('my event')
def handle_my_custom_event(jsonrecv, methods=['GET', 'POST']):
    currentReminder = jsonrecv
    print(currentReminder)
    socketio.emit('my response', jsoninit, callback=messageReceived)
    return render_template('post.html', data = currentReminder)
```

FIGURE 16. SocketIO route for the User Interface

Webpage Javascript. — Finally, the third part of the connection is the Javascript code for the UI (figures 17, 18 and 19). This has functions not only to give functionality to the interface, but also to allow communication with the server and, hence, with the Arduino and sensors. Firstly, it is necessary to initialise a socket with *socketio*, which will communicate with the server through its URL. Secondly, in order to send the medicine reminder data to the server, an *event* is made when the time was set on the UI. This data is then sent over in a JSON format. Whenever data is received in the server, *my response* event is done. After checking that the data transmission was done successfully, *displayMedicineReminder* and *calculateDanger* functions are called, which update the parameters on the UI. As a consequence, a complete connection is established from the sensors to the user of the website, through the UI.

```javascript
var socket = io.connect('https://itis.serveo.net');
```

FIGURE 17. SocketIO connection to the server initialization

```javascript
//To send Data (Medicine reminder)
socket.on( 'connect', function() {
  socket.emit( 'my event', {
    data: 'User Connected'
  })
  var form = $( 'form' ).on( 'submit', function(e) {
    e.preventDefault()
    let user_hour = medicineHours[correctIndex];
    let user_minute = medicineMinutes[correctIndex];
    socket.emit( 'my event', {
        minutes : user_minute,
        hour : user_hour
    })
  })
})
```

FIGURE 18. Data transmission to server *event*

```javascript
//When receiving Data
socket.on( 'my response', function( msg ) {
    console.log( msg );
    if( typeof msg.heartbeat !== 'undefined' ) {
        displayMedicineReminder();
        calculateDanger(msg.heartbeat, msg.oxygen, msg.accelerometer);
    }
})
```

FIGURE 19. Data reception from server *event*

3.5. GPS. — SIM808 also comprises of a GPS module. For the purpose of this design, this module is used in two different applications and *Siretta ECHO26/0 - GPS internal active antenna* is chosen to fulfil the need for a GPS antenna.
First of all, the user that accesses the website can request to get the current location of the watch

wearer, so that a relative can find the position of the elderly in case he gets lost or there is an emergency situation. This feature was implemented in our code by using an existing library. The result in the current implementation is, however, not nearly the optimal one. In fact, the coordinates obtained are truncated to the first decimal digit, making the location far from being accurate. For example, when requesting the location inside the Electrical Engineering building at Imperial College London, the obtained coordinates place the user in the southern suburbs of London, due to lack of precision occurred in coordinate digits obtained. This is obviously not accurate enough to be useful in the final implementation, but increasing the precision would be enough to make the feature very relevant in the overall design of the product.

A second use for GPS connection relates to the medicine reminder. The time at which the vibration has to go off is set from the website. However, there must be a way to keep track of the current time on the Arduino board so that the vibrator can remind the wearer to take medicines at the right moment of the day. This is ideally achieved via GPS connection. However, there is a number of reasons why this option is not usable at the moment. Firstly, it takes about 10 minutes for GPS to sync, which is far too long in a device for everyday use. Moreover, it often fails to connect to satellites when used in inside spaces, which is a huge drawback considered that the operating environment of the device is very likely to be indoors. It follows that a great amount of work is to be put to further enhance GPS reliability.

3.6. Overall Arduino Code. — As mentioned in the previous sections about sensors and connection, the program was written on the Arduino IDE and uploaded on an Arduino Mega board. Refer to the appendix (page 27) for the complete Arduino code.

`setup()`. — In the setup, the following operations are performed:

1. the accelerometer sensor is initialised by setting the desired range of operation, then the most important information about the sensor is displayed on the serial monitor;
2. the connection module is initialised and the times at which the medicine has to be taken are received as an input;
3. the screen is initialised;
4. digital pin 22 is set as output in order for it to be used as an input in the motor circuit;
5. the heart rate sensor and pulse oximeter is initialised;
6. the current time is given as input. From the moment in which the program is uploaded to the board onwards, the Arduino will keep track of the current time.

`loop()`. — In the loop, the following operations are performed:

1. the heart rate sensor and pulse oximeter collects its data and processes it, as further explained in the "Heart rate sensor and pulse oximeter" section in page 5;
2. the accelerometer collects its data and processes it, as described in the "Accelerometer" section in page 7;
3. the data from the sensor is displayed on the serial monitor and on the screen, as well as being sent to the website.
4. if the medicine reminder has to go off, the necessary outputs to make the motor vibrate are sent.

Delays. — A vital aspect to be taken into account throughout the program is that the settings of the heart rate and pulse oximeter sensor need to be constantly updated by calling the `pox.update()` function. For this reason, the needed delays throughout the program can not be implemented by calling *delay(milliseconds)*, but are to be performed using a *while* loop inside which the function `pox.update()` is called. `pox.update()` is to be called also before and after all the computationally costly operations, that take long to run.

Arduino board. — The overall Arduino program takes 11% of the board's storage space and 39% of its dynamic memory. This means that none of the smaller Arduino boards could be used to run our

program. The available memories for the main boards that the group considered using are shown in the table below.

	Arduino Uno and Arduino Nano	Arduino Mega
Flash Memory	32 KB	128 KB
SRAM	2 KB	8 KB
EEPROM	1 KB	4 KB

TABLE 1. Available memories for Arduino boards considered

3.7. Power. — As mentioned in the "Design" section (page 18), the prototype of the smart watch demands three $9V$ batteries in order to supply enough power to the Arduino board. After carrying out some research, the group decided that *Panasonic Pro Power* $9V$ *Battery Alkaline* was the most suitable and reliable option for the prototype. In fact, the two sensors (i.e. ADXL345 and MAX30100) and the motor circuit for the medicine reminder would have worked just including one 9V battery in the design. However, this was not the case for the overall circuit design of the prototype. After numerous tests and measurements, the group came to the conclusion that the connection module (i.e. SIM808), when not transmitting or receiving any data, draws around $0.1A$ of background current. One 9V battery is able to supply around $0.35A$, which results in being enough for this background current. However, when the connection module is transmitting or receiving, it draws up to $1A$ of current, which is the reason why three batteries are needed. The 9V batteries are connected in parallel so that the currents add up but the voltage is at the right level. This solution for providing enough power supply is not optimal since including all these batteries makes the watch quite heavy and, therefore, it directly effects the wearability of the watch.

The estimated lifetime for the batteries in the current implementation is 8.25 hours. This is because one battery has capacity of 550mAh. Three batteries will therefore have a capacity of 1650mAh. The average total current is estimated to be 200mA. $1650/200 = 8.25$ hours of life.

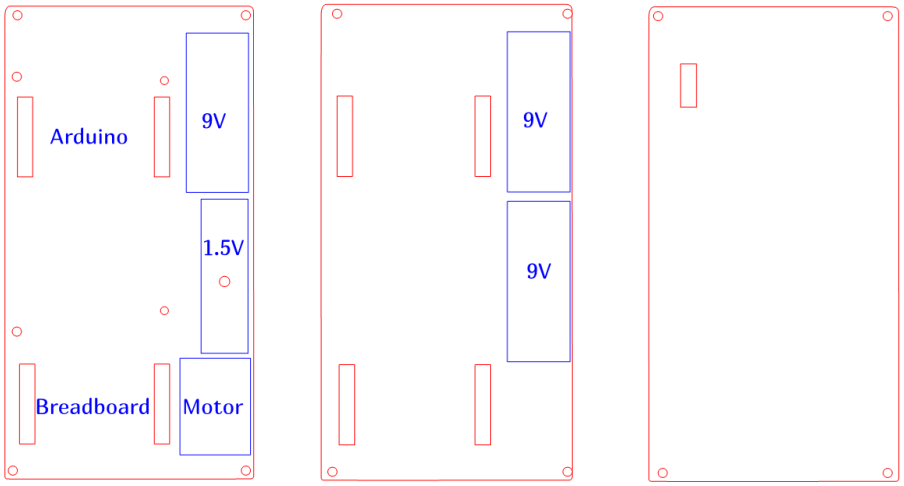
One feasible solution for to extend batter life is to use the SIM808's *power saver mode*. It temporarily shuts off its GSM connection while keeping the parameters that initialized from the setup. During sleep mode, the SIM808 draws only 20mA of ambient current. Hence, this implementation could increase the lifetime of the watch significantly.

The motor requires a voltage 1.5V and a current 0.7A to run, hence we used a *Duracell Plus Power* AA battery to power it. Since the motor turns on rarely, the lifetime of this battery will not be an issue when considering the maintenance of the product.

In conclusion, the group is aware of the fact that it is necessary to find a better solution for the power supply in order to make the device more wearable. Possible improvements are addressed in the "Further Developments" section.

4. Design

The physical design of the prototype of the smart watch is made using Gravit Designer, which is an online vector graphic design app and image editor. The structure is then cut using a laser cut machine. The design consists of two layers: the first contains all the components of the product, and the second, which functions as a covering layer on top of the first, is placed with using four standoffs of height 3 cm, at a perpendicular angle with respect to the other layer. The top layer has two main purposes: protecting the bottom one and making the prototype as close as possible to a final product, where the user can't see the inside of the watch. In addition, the prototype can be worn using two straps which go around the wrist and arm.



(A) Bottom Layer Upwards

(B) Bottom Layer Downwards

(C) Top Layer

FIGURE 20. Layers of the Watch

As shown figure 20A, there are 4 rectangular holes for the straps. Moreover, on this layer an Arduino Mega, one 9V batter, one 1.5V battery and its battery holder, the motor and a small breadboard are placed. Then, the connection module is placed on top of the Arduino. The breadboard contains all the necessary connections, the circuit for the motor and the actual sensors. The accelerometer is placed on the breadboard, while the heartbeat sensor and oximeter is connected to the Arduino through the breadboard but is accompanied with long wires which enable the wearer's finger to be placed on the sensor itself. Figure 20B shows the bottom layer of the prototype from another prospective: this side is placed in contact with the wrist. Two additional 9V batteries are placed here due to the high power consumption of the connection module (see "Power" section in page 18).

The top layer (figure 20C) also has 4 holes at the edges to support the standoffs. There is a small rectangular hole in the top left corner of the covering layer that has the purpose of fitting the LED screen which displays the measurements from heart rate and oxygen concentration in the blood, allowing the user to see these values while wearing the watch.

Ergonomics and manufacturing considerations. — The device needs to be adaptable such that the wearer can adjust it to his wrist. This is achieved by having straps attached to the bottom layer of the watch. A LED screen is also placed on the top layer so that the elderly person can track his heartbeat and oxygen levels himself. In order to insulate batteries from a direct contact with the user, which might lead to hazardous incidents, battery holder is utilised in the design.

5. Further Developments

This section deals with the further developments which could be applied to the prototype of the smart watch in order to make it more reliable and wearable, making it a competitive product in the market.

5.1. Heartrate sensor and oximeter. — As for the heart rate sensor and oximeter, in the final product the sensor would be placed on the wrist and not on the finger as in the current prototype so that the elderly's heatbeat and oxygen level data can be accessed at all times: It is understandable that it is unlikely that the wearer always keeps the finger on the sensor, applying the right amount

of pressure for the sensor to be precise. Therefore, moving this sensor to the wrist would imply that readings could be taken constantly and, hence, be more accurate (currently the sensor takes around 10 seconds until it starts getting the correct readings). Moreover, the only part needed for the heart rate and oxygen level readings is MAX30100 and not the whole integrated circuit (see figure 3), making the structure smaller and thus more wearable.

Another improvement would be to have more computational power, with which a mean of more values (e.g. 100) could be taken, making the average more accurate since some readings tend to not be very accurate.

In addition, an alternative faster way to process the data from this sensor is to use the built-in programmable interrupt of the sensor. In this way, there would not be the need to take different readings. Whenever a reading which is above the *DANGER* threshold is obtained, *interrupt* would be asserted so that the danger level is directly sent to the website.
Another health condition that could be monitored is the blood pressure. In future implementations of the smart watch, this feature would be added in order to improve the monitoring of the elder's health, further preventing severe physical damages.

5.2. Accelerometer. — Once again, as for the heart rate sensor and oximeter, interrupts could be used: Instead of doing a subtraction of two consecutive readings and check if the result exceeds a predetermined value, an interrupt could be asserted when a reading exceeds this value, meaning that a fall was detected. In this way, the response from the sensor would be faster and would result in less computational cost for the Arduino.

5.3. Medicine Reminder. — A smaller vibrator could be chosen instead of the DC motor, which would make the overall design of the prototype significantly smaller.
Moreover, a sound feature could be added jointly with vibration so that the wearer will be less likely to miss the warnings.
In the current design, one AA battery is used to provide power to the motor. This eventually runs out in the long term use. To avoid this, the power will be obtained thorough a rechargeable battery.

5.4. Connection & Website User Interface. — In the current implementation, data is sent from the device to the website, which displays values from the sensors and classifies them into different levels of danger. As for the heart rate sensor, *Level 3 Danger* is a severe one. Due to the gravity of this danger level, an improvement would be for the device to directly call for medical help. This means calling 999 (for United Kingdom) whenever a *Level 3 Danger* is obtained (refer to "Heart rate sensor and pulse oximeter" section in page 5 for further information about the danger levels).
Although the current website interface design is well organised and includes all the fundamental functions that are needed, there are still some aspects that could be improved further. An additional feature would be to inform relatives directly through a notification as soon as the highest levels of danger arise, in order to facilitate access to immediate medical care. Moreover, basic information and history of emergency should be recorded in the user profile as a reference for the future.

5.5. GPS. — As mentioned in the GPS section (see page 16), a vital upgrade would be to get more precise coordinates for the location of the wearer, not just to the first decimal digit as it is done now. Moreover, as the time parameter for the medicine reminder could be taken using the GPS, another improvement is making the set up time of the GPS significantly shorter as in the current implementation it takes around 10 minutes, as well as making the GPS usable in inside spaces where it often fails to connect.

5.6. Power. — A major weakness of the prototype is its power consumption. The current implementation uses three $9V$ batteries to supply power to the Arduino board and one $1.5V$ battery for the motor (refer to "Power" section in page 18 for more detailed information about the power consumption). As a consequence, the prototype is quite large in size and most importantly heavy. These characteristics

make the prototype watch not nearly optimal to be used on daily basis. For these reasons, a further development would be to replace batteries used with the ones that provide same amount of power, but which are also smaller in size and weight. This would make the watch significantly more wearable.

A further upgrade is to replace the batteries with a rechargeable one, in order to make the daily maintenance of the watch easier and considerably less expensive.

In addition, since the connection board is the component that draws most of the power, an upgrade would be to replace it with a counterpart which demands less power supply and, thus, draws less current.

6. Budget

The total cost for the design and implementation of the watch prototype is represented in table 2.

Product	Cost (in £)	Supplier
Heartrate sensor and oximeter – MAX30100	15.27	RS components
Accelerometer - ADXL345	14.88	Rapid components
RS PRO DC Motor	4.55	RS components
GSM Antenna - ECHO1A/0.1M/IPEX/S/S/11 Siretta	9.65	Rapid components
Siretta ECHO26/0 - GPS internal active antenna	5.24	RS components
Screen	9.99	Amazon
Breadboard	4.40	Rapid components
9V battery (x3)	17.00	Waitrose
1.5V battery	1.50	Waitrose
1.5V battery holder	0.77	RS components
Vishay 50V 1A – Diode	2.00	RS components
Transistor	2.00	RS components
Arduino Mega	28.41	RS components
Stand-offs (pack of 10)	5.38	RS components
SIM808 Module GPS GSM Quad Band Development Board	15.45	banggood.com
Coloured acrylic for laser cut	6.00	Imperial Union Shop
Straps	4.81	Amazon
Total cost:	**£147.30**	

TABLE 2. Budget for *Over-Watch v1.0*

7. Project Management

Throughout the project the work has been divided into smaller subsections so that every member could specialise in one area. This gave a deep insight in specific aspects of the development of the watch, which allowed members to drastically improve their technical understanding of these. Figure 21 is a diagram that represents the division of work.

The group regularly met every Tuesday and Friday during Spring Term so that every member was informed about the progresses made regarding all aspects of the project. During the last three weeks before the submission of the project, the group met four times per week: on Mondays, Wednesdays,

Fridays, and Sundays. Through Google Drive file, weekly accomplishments were constantly published so that each member was aware of the developments and new tasks were also published in this file. Furthermore, the group has a Facebook chat for urgent questions and clarifications.

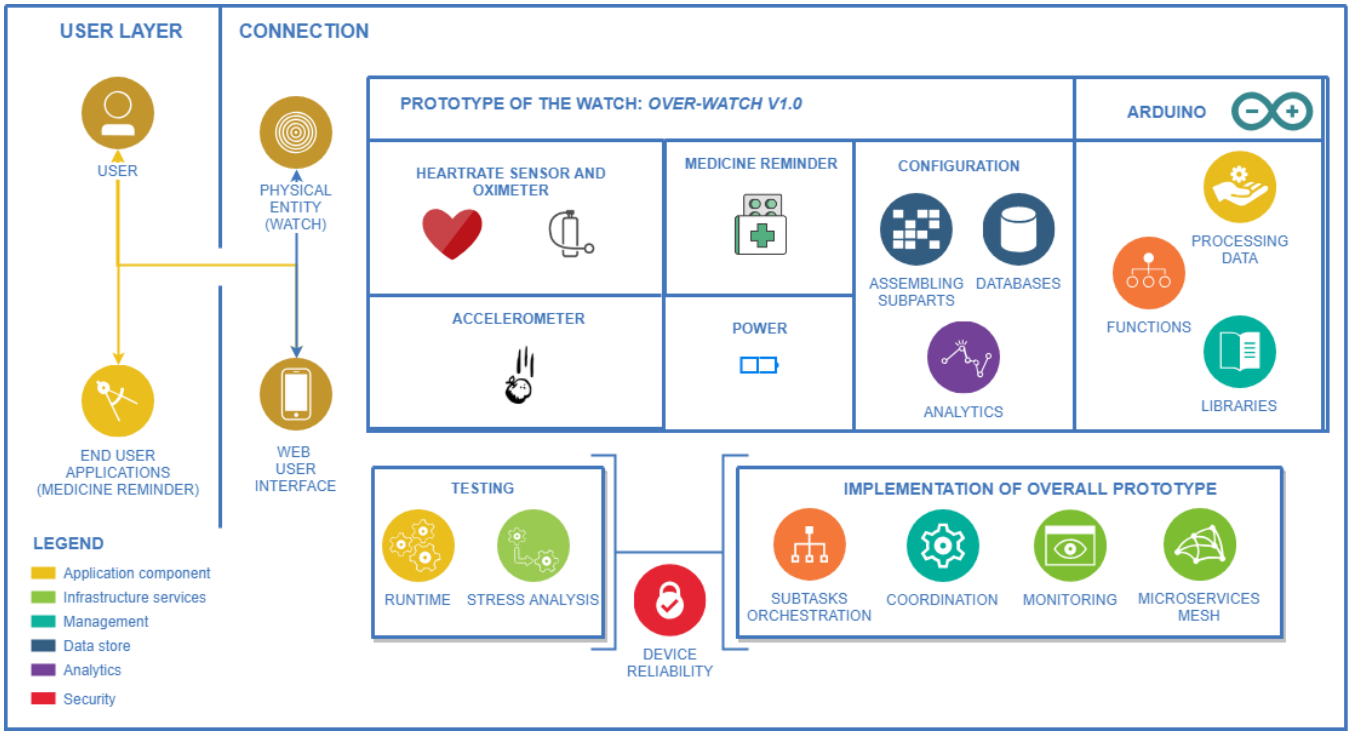


FIGURE 21. Division of Work.

8. Over-watch v1.0

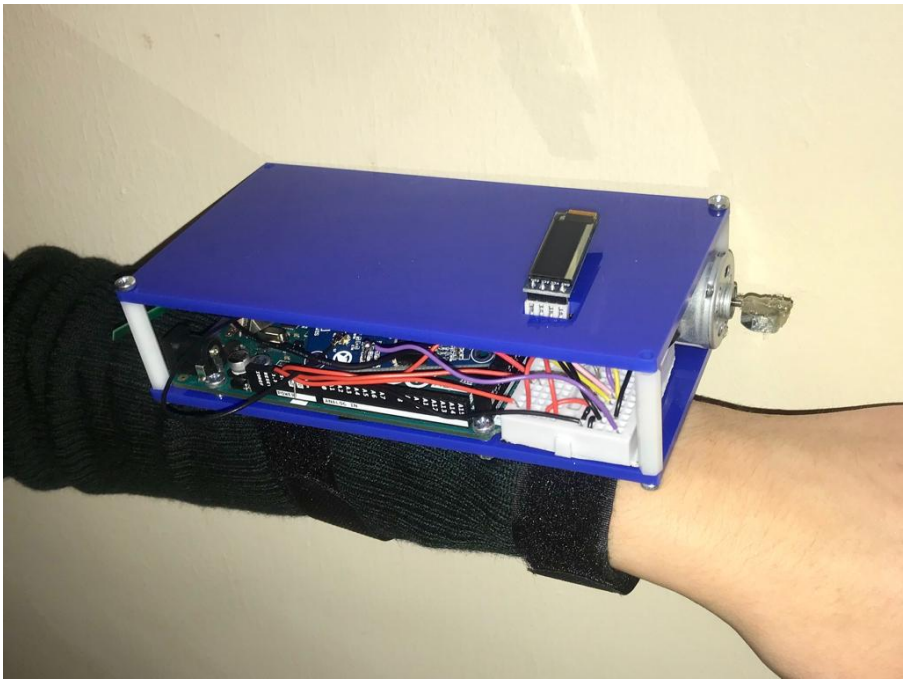


FIGURE 22. The final design of the prototype

9. Conclusion

Overall, the purpose of the smart watch has been achieved. This device aims at improving elderlies' lives by monitoring their health and sending the relevant information to a website accessed by the people that care for them (i.e. relatives or doctors). This would result in allowing the elder to be more independent and safe.

The oximeter and heart rate sensor measures the heartbeat and oxygen level in the blood. Three different levels of danger for the heart rate are set: *DANGER 1* and *DANGER 2* give an indication of possible unwanted heath situations, while *DANGER 3* is always a sign for needing immediate medical help. This information is sent to the website using a GSM connection which displays current levels of heart rate as well as the different dangers that could be encountered. The same procedure is applied to the oxygen concentration, however this only has two different levels of danger. Moreover, current heart rate and SPO2 levels are displayed on the screen on the top layer of the watch prototype. By having this features, possible health related problems could be prevented.

The accelerometer detects sudden movements which imply falls. If a fall is detected, this warning is sent to the website. This would solve many inconveniences that arise when elderlies are alone and can't reach for help.

The medicine reminder aspect of the smart device is meant to warn the wearer to take medications. This is implemented using a motor which vibrates and the time for this reminder to go off is set from the website. This component is useful since elderlies have to take pharmaceuticals on a daily basis and often forget to, which could lead lo severe consequences.

Another aspect of the device which aims at improving old people's lives is having a GPS which gives information about the wearer's location through the website. This is of great help since it is very common between old people to have sudden memory loss and, therefore not being able to find their way home.

Many further developments which could improve the reliability and ta

10. Appendix

10.1. Heart rate sensor and pulse oximeter Flowchart. —

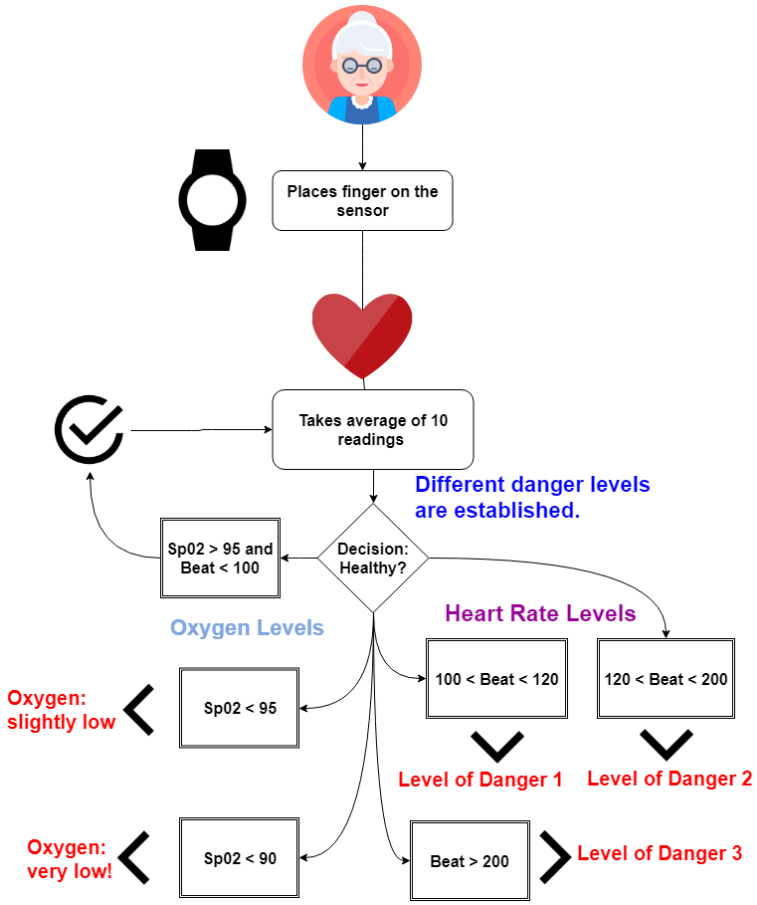


FIGURE 23. Sensor used: MIKROE-200 Heart Rate Click with MAX30100 integrated pulse oximetry and heart rate sensor.

10.2. Accelerometer sensor Flowchart. —

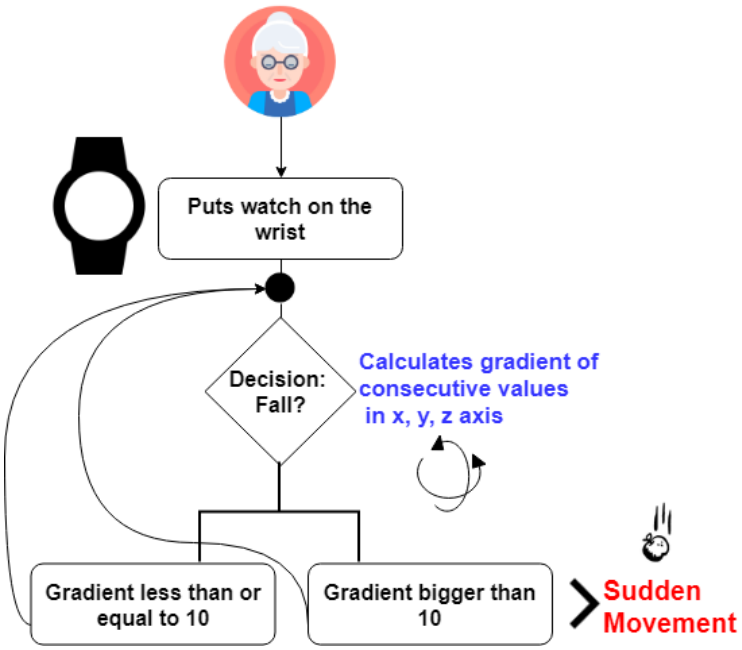


FIGURE 24. Sensor used: Adafruit 1231 ADXL345 Triple-Axis Accelerometer.

10.3. Connection Flowchart. —

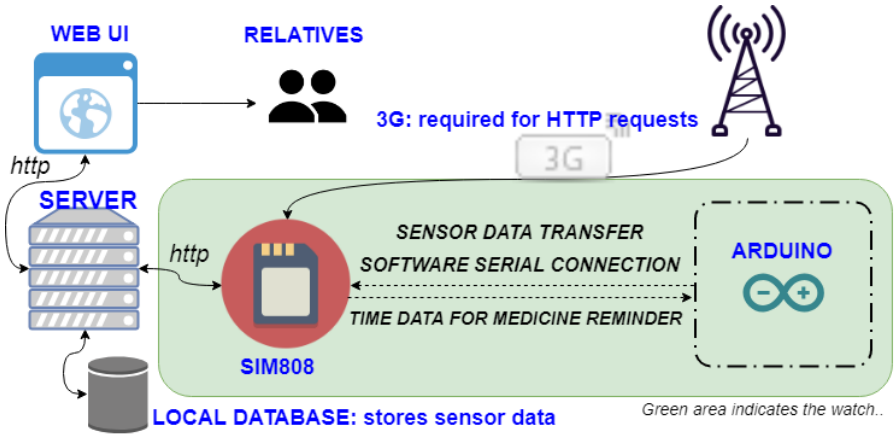


FIGURE 25. Module used: SIM808 GSM/GPRS/GPS.

10.4. Medicine reminder Flowchart. —

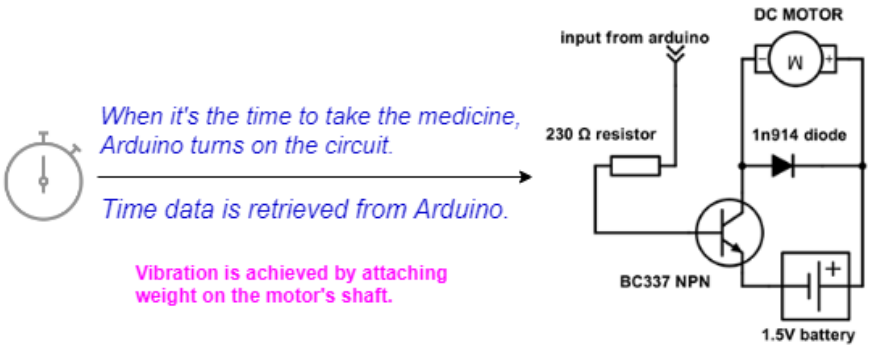


FIGURE 26. Module used: SIM808 GSM/GPRS/GPS.

10.5. Overall Arduino Code. —

```
////LIBRARIES
#include <Wire.h>

//HEART
#include <MAX30100_Filters.h>
#include <CircularBuffer.h>
#include <MAX30100.h>
#include <MAX30100_SpO2Calculator.h>
#include <MAX30100_BeatDetector.h>
#include <MAX30100_PulseOximeter.h>
#include <MAX30100_Registers.h>

//ACC
#include <Servo.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_ADXL345_U.h>

//SCREEN
#include <U8g2lib.h>
#include <U8x8lib.h>

//TIME
#include <Time.h>
#include <TimeLib.h>

//CONNECTION
#include <SoftwareSerial.h>


////VARIABLES

//HEART
#define REPORTING_PERIOD_MS     500
PulseOximeter pox;
const int numReadings=10;
float filterweight=1;
uint32_t tsLastReport = 0;
uint32_t last_beat=0;
float readIndex=0;
int average_beat=0;
int average_SpO2=0;
int previous_average_beat=0;
int previous_average_SpO2=0;
bool calculation_complete=false;
bool calculating=false;
bool initialized=false;
byte beat=0;

//ACCELEROMETER
float x,y,z;
float prevz, diffz, prevy, diffy, prevx, diffx;
Adafruit_ADXL345_Unified accel = Adafruit_ADXL345_Unified(12345);
uint32_t last_fall = 0;
bool fall_sent=true;

//SCREEN
U8G2_SSD1306_128X32_UNIVISION_F_HW_I2C u8g2(U8G2_R0);
```

```
//MOTOR
int motor_pulse = 0;
uint32_t last_motor = 0;
bool sick = true;


//CONNECTION
SoftwareSerial GPRS(10,11);
//phone: 18917644766
unsigned char buffer[64];
unsigned char constant[64];
unsigned char strhour[4];
unsigned char strminute [4];
int hour_med;
int minute_med;
int count = 0;
int concount = 0;
String sendOver;
int flag = 0;

////FUNCTION DEFINITION

//HEART
void onBeatDetected()
{
    last_beat=millis();
}

void initial_display() {
    if (not initialized)
    {
        Serial.println("Place finger on the sensor.");
        u8g2.clearBuffer();
        u8g2.setCursor(24,12);
        u8g2.setFont(u8g2_font_smart_patrol_nbp_tf);
        u8g2.print("Place finger");
        u8g2.setCursor(0,30);
        u8g2.print("on the sensor");
        u8g2.sendBuffer();
        initialized=true;
    }
}

void display_values() {
    u8g2.clearBuffer();
    u8g2.setFont(u8g2_font_smart_patrol_nbp_tf);

    u8g2.setCursor(65,12);
    u8g2.print(average_beat);
    u8g2.print(" Bpm");
    u8g2.setCursor(0,30);
    u8g2.print("SpO2 ");
    u8g2.setCursor(65,30);
    u8g2.print(average_SpO2);
    u8g2.print("%");
    u8g2.sendBuffer();

    Serial.print("Average beat: ");
    Serial.print(average_beat);
```

```
    Serial.println(" Bpm.");
    if (average_beat>100 && average_beat<120){
        Serial.println("HEARTBEAT: LEVEL OF DANGER 1");
    }
    else if (average_beat>120 && average_beat<200){
        Serial.println("HEARTBEAT: LEVEL OF DANGER 2");
    }
    else if (average_beat>200){
        Serial.println("HEARTBEAT: LEVEL OF DANGER 3 - SEE A DOCTOR!");
    }

    Serial.print("Average SpO2: ");
    Serial.print(average_SpO2);
    Serial.println(" %");

    if (average_SpO2>=90 && average_SpO2<95){
        Serial.println("SpO2: your oxigen concentration is slightly low :(");
    }
    else if (average_SpO2<90){
        Serial.println("SpO2: your oxigen concentration is very low. DANGER: SEE A DOCTOR.");
    }
    Serial.println();

}

void calculate_average(int beat, int SpO2) {

    if (not calculation_complete and beat>40 and beat<220) {
        average_beat = filterweight * (beat) + (1 - filterweight ) * previous_average_beat;
        previous_average_beat = average_beat;
        average_SpO2 = filterweight * (SpO2) + (1 - filterweight ) * previous_average_SpO2;
        previous_average_SpO2 = average_SpO2;
        readIndex++;
        filterweight = 1/(readIndex+1);
    }
    if (readIndex==numReadings) {
        calculation_complete=true;
        calculating=false;
        initialized=false;
        readIndex=0;
        filterweight=1;
        display_values();
    }
}


//ACCELEROMETER
void displaySensorDetails(void)
{
    sensor_t sensor;
    accel.getSensor(&sensor);
    Serial.println("------------------------------------");
    Serial.print  ("Sensor:       "); Serial.println(sensor.name);
    Serial.print  ("Driver Ver:   "); Serial.println(sensor.version);
    Serial.print  ("Unique ID:    "); Serial.println(sensor.sensor_id);
    Serial.print  ("Max Value:    "); Serial.print(sensor.max_value); Serial.println(" m/s^2");
    Serial.print  ("Min Value:    "); Serial.print(sensor.min_value); Serial.println(" m/s^2");
    Serial.print  ("Resolution:   "); Serial.print(sensor.resolution); Serial.println(" m/s^2");
    Serial.println("------------------------------------");
```

```
    Serial.println("");
    delay(500);
}

void displayDataRate(void)
{
    Serial.print  ("Data Rate:    ");

    switch(accel.getDataRate())
    {
        case ADXL345_DATARATE_3200_HZ:
            Serial.print  ("3200 ");
            break;
        case ADXL345_DATARATE_1600_HZ:
            Serial.print  ("1600 ");
            break;
        case ADXL345_DATARATE_800_HZ:
            Serial.print  ("800 ");
            break;
        case ADXL345_DATARATE_400_HZ:
            Serial.print  ("400 ");
            break;
        case ADXL345_DATARATE_200_HZ:
            Serial.print  ("200 ");
            break;
        case ADXL345_DATARATE_100_HZ:
            Serial.print  ("100 ");
            break;
        case ADXL345_DATARATE_50_HZ:
            Serial.print  ("50 ");
            break;
        case ADXL345_DATARATE_25_HZ:
            Serial.print  ("25 ");
            break;
        case ADXL345_DATARATE_12_5_HZ:
            Serial.print  ("12.5 ");
            break;
        case ADXL345_DATARATE_6_25HZ:
            Serial.print  ("6.25 ");
            break;
        case ADXL345_DATARATE_3_13_HZ:
            Serial.print  ("3.13 ");
            break;
        case ADXL345_DATARATE_1_56_HZ:
            Serial.print  ("1.56 ");
            break;
        case ADXL345_DATARATE_0_78_HZ:
            Serial.print  ("0.78 ");
            break;
        case ADXL345_DATARATE_0_39_HZ:
            Serial.print  ("0.39 ");
            break;
        case ADXL345_DATARATE_0_20_HZ:
            Serial.print  ("0.20 ");
            break;
        case ADXL345_DATARATE_0_10_HZ:
            Serial.print  ("0.10 ");
            break;
        default:
```

```
            Serial.print  ("???? ");
            break;
    }
    Serial.println(" Hz");
}

void displayRange(void)
{
    Serial.print  ("Range:         +/- ");

    switch(accel.getRange())
    {
        case ADXL345_RANGE_16_G:
            Serial.print  ("16 ");
            break;
        case ADXL345_RANGE_8_G:
            Serial.print  ("8 ");
            break;
        case ADXL345_RANGE_4_G:
            Serial.print  ("4 ");
            break;
        case ADXL345_RANGE_2_G:
            Serial.print  ("2 ");
            break;
        default:
            Serial.print  ("?? ");
            break;
    }
    Serial.println(" g");
}


//CONNECTION
void sendGSMsetup(const char* msg, int waitMs = 500) {
    GPRS.println(msg);
    delay(waitMs);
}
void sendGSM(const char* msg, int waitMs = 500) {
    pox.update();
    GPRS.println(msg);
    pox.update();
    //delay(waitMs);
}

void getRequestHeart(int heart, int oxy){
    pox.update();
    sendOver = "AT+HTTPPARA=\"URL\",\"http://3c0b35fa.ngrok.io/postHeart?heartbeat="
    + String(heart) + "&oxygen=" + String(oxy) + "\"";
    sendGSM(sendOver.c_str());
    sendGSM("AT+HTTPACTION=0");
    pox.update();
}

void getRequestAccel(int acceldata){
    pox.update();
    sendOver = "AT+HTTPPARA=\"URL\",\"http://3c0b35fa.ngrok.io/postAccel?accelerometer="
    + String(acceldata) + "\"";
    sendGSM(sendOver.c_str());
    sendGSM("AT+HTTPACTION=0");
```

```
    pox.update();
}

void getRequestMed(int meddata){
    sendOver = "AT+HTTPPARA=\"URL\",\"http://3c0b35fa.ngrok.io/postTime?accelerometer="
    + String(meddata) + "\"";
    sendGSMsetup(sendOver.c_str());
    delay(1000);
    toSerial();
    sendGSMsetup("AT+HTTPACTION=0");
    delay(4000);
    toSerial();
    delay(500);
    sendGSMsetup("AT+HTTPREAD");
    toSerial();
}

void sim808_setup() {
    sendOver = "AT+HTTPPARA=\"URL\",\"http://3c0b35fa.ngrok.io/post?heartbeat=" + String(10)
    +"&accelerometer=" + String(10) + "&oxygen=" + String(10) + "\"";
    delay(500);
    GPRS.println("AT+CGNSPWR=1\n");

    delay(500);

    // wait ten seconds for GSM module to connect to mobile network
    Serial.println( "Waiting for SIM startup..." );
    delay(10000);

    Serial.println("Before startup");
    sendGSMsetup("AT+CREG?");
    sendGSMsetup("AT+SAPBR=3,1,\"APN\",\"pp.vodafone.co.uk\"");
    sendGSMsetup("AT+SAPBR=3,1,\"USER\",\"wap\"");
    sendGSMsetup("AT+SAPBR=3,1,\"PWD\",\"wap\"");
    sendGSMsetup("AT+HTTPINIT",1000);
    sendGSMsetup("AT+SAPBR=1,1",3000);
    sendGSMsetup(sendOver.c_str());

    Serial.println("After Startup");

    GPRS.println("AT+CGNSPWR=1\n");

    delay(500);
}

void clearBufferArray(){
    for (int i = 0; i < count; i++) {
        buffer[i] = NULL;
    }
}

void splitString(){
    strhour[0] = constant[0];
    strhour[1] = constant[1];
    strminute[0] = constant[3];
    strminute[1] = constant[4];
    minute_med = atoi(strminute);
    hour_med = atoi(strhour);
}
```

```
void toSerial() {
    int i = 0;
    while(GPRS.available()!=0) {
        if( GPRS.peek() == '['){
            flag=2;
            constant[i] = GPRS.read();
        }
        else if(flag == 2 && i  != 5)
            while(GPRS.peek() != ']'){
                constant[i] = GPRS.read();
                i++;
            }
        else if(flag == 0){
            Serial.write(GPRS.read());
        }
        else{
            flag = 0;
        }
    }
}

void setup(){
    //Serial.begin(115200);
    Serial.begin(9600);

    //ACCELEROMETER
    #ifndef ESP8266
    while (!Serial);
    #endif
    Serial.println("Accelerometer Test"); Serial.println("");

    //Initialise the sensor
    if(!accel.begin()) {
        Serial.println("Ooops, no ADXL345 detected ... Check your wiring!");
        while(1);
    }

    //Set the range
    //accel.setRange(ADXL345_RANGE_16_G);
    //accel.setRange(ADXL345_RANGE_8_G);
    //accel.setRange(ADXL345_RANGE_4_G);
    accel.setRange(ADXL345_RANGE_2_G);

    // Display some basic information on this sensor
    displaySensorDetails();

    // Display additional settings (outside the scope of sensor_t)
    displayDataRate();
    displayRange();
    Serial.println("");


    //CONNECTION
    GPRS.begin(9600);
    sim808_setup();
    GPRS.end();
    GPRS.begin(9600);
    getRequestMed(5);
```

```
    splitString();
    Serial.print("MEDICINE HOUR: ");
    Serial.println(hour_med);
    Serial.print("MEDICINE MINUTE: ");
    Serial.println(minute_med);


    //SCREEN
    u8g2.begin();

    //MOTOR
    pinMode(22,OUTPUT);

    //HEART
    pox.begin();
    pox.setOnBeatDetectedCallback(onBeatDetected);
    initial_display();

    //TIME
    //setTime(hr,min,sec,day,month,yr);
    setTime(12,20,00,19,03,2019);

    Serial.println("end setup");
}


void loop(){

    //HEART
    pox.update();
    if ((millis() - tsLastReport > REPORTING_PERIOD_MS) and (not calculation_complete)) {
        /*Serial.print("get beat ");
         int b = pox.getHeartRate();
         Serial.println(b);*/
        calculate_average(pox.getHeartRate(),pox.getSpO2());
        tsLastReport = millis();
    }
    if (calculation_complete){
        getRequestHeart(average_beat, average_SpO2);
        calculation_complete=false;
        average_beat=0;
        average_SpO2=0;
    }

    //ACCELEROMETER
    //Get a new sensor event
    sensors_event_t event;
    x = 0;
    y = 0;
    z = 0;
    for(int i = 0; i < 30; i++){
        accel.getEvent(&event);
        x += event.acceleration.x;
        y += event.acceleration.y;
        z += event.acceleration.z;
    }
    x = x/30;
    y = y/30;
    z = z/30;
```

```
    diffz = event.acceleration.z - prevz;
    prevz = event.acceleration.z;
    diffy = event.acceleration.y - prevy;
    prevy = event.acceleration.y;
    diffx = event.acceleration.x - prevx;
    prevx = event.acceleration.x;

    //detection of fall
    if (diffx > abs(10) || diffy > abs(10) || diffz > abs(10)) {
        Serial.println ("FALL DETECTED");
        last_fall=millis();
        getRequestAccel(11); //11 is sent because any value above 10 is deen as a fall.
        fall_sent=false;
    }
    if(millis()-last_fall>10000 && fall_sent==false){
        getRequestAccel(0);
        fall_sent=true;
    }

    //MOTOR
    if (hour()==hour_med && minute()==minute_med && sick==true){
        while(motor_pulse < 3){
            Serial.print("motor pulse");
            Serial.println(motor_pulse);
            digitalWrite(22,HIGH);
            last_motor = millis();
            while ( (millis() - last_motor) < 1000){
                pox.update();
            }
            digitalWrite(22,LOW);
            last_motor = millis();
            while ( (millis() - last_motor) < 1000){
                pox.update();
            }
            motor_pulse++;
        }
        sick=false;
        motor_pulse=0;
    }
    if (minute()==(minute_med+1)){
        sick=true;
    }
}
```

10.6. Website Javascript Code. —

```
var socket = io.connect('http://localhost:5000');

var medicineHours = [];

var medicineMinutes = [];

var sortedMedicineHours = [];

var sortedMedicineMinutes = [];

var correctIndex;

var numberOfReminders = 0;

var minutes_t = 1000 * 60;
var hours_t = minutes_t * 60;
var days = hours_t * 24;
var years = days * 365;
var d = new Date();
var t = d.getTime();
var currentYear = Math.round(t / years) + 1970;
var currentDay = Math.round(t / days);
var currentHour = Math.round(t / hours_t) - (currentDay)*24;
var currentMinute = Math.round(t / minutes_t) - (currentDay)*24*60 - (currentHour)*60;

function calculateDanger(heart, oxy, acc){
    appendString = '<h2 class="box_output">' + heart + '</h2>';
    $( 'div.beatHolder' ).html(appendString);

    appendString = '<h2 class="box_output">' + oxy + '</h2>';
    $( 'div.oxygenHolder' ).html(appendString);

    if(heart >= 100 && heart <= 120){
        $( 'div.beatDanger' ).html('<h2 class="box_output">DANGER Level 1!</h2>');
    }
    else if(heart > 120 && heart <= 150){
        $( 'div.beatDanger' ).html('<h2 class="box_output">DANGER Level 2!</h2>');
    }
    else if(heart > 150){
        $( 'div.beatDanger' ).html('<h2 class="box_output">DANGER Level 3!</h2>');
    }
    else if(heart < 10){
        $( 'div.beatDanger' ).html('<h2 class="box_output">No Data</h2>');
    }
    else{
        $( 'div.beatDanger' ).html('<h2 class="box_output">Normal</h2>');
    }

    if(oxy < 95 && oxy > 90){
        $( 'div.oxyDanger' ).html('<h2 class="box_output">Your Oxygen Is Slightly Low</h2>');
    }
    else if( oxy <= 90 && oxy > 20){
        $( 'div.oxyDanger' ).html('<h2 class="box_output">Danger, see a doctor!</h2>');
    }
    else if(oxy < 10){
        $( 'div.oxyDanger' ).html('<h2 class="box_output">No Data</h2>');
    }
    else{
```

```
        $( 'div.oxyDanger' ).html('<h2 class="box_output">Normal</h2>');
    }

    if(acc > 1){
        $( 'div.fallHolder' ).html('<h2 class="box_output">DANGER!</h2>');
    }
    else{
        $( 'div.fallHolder' ).html('<h2 class="box_output">No</h2>');
    }
}

function displayMedicineReminder(){
    console.log("displayStart");
    if(currentMinute < 0){
        currentMinute = currentMinute + 60;
        currentHour = currentHour - 1;
    }
    if(currentHour < 0){
        currentHour = currentHour + 24;
    }
    hourvalue = 30;
    for(var i = 0; i <= medicineHours.length - 1; i++){
        if((medicineHours[i] - currentHour) < hourvalue){
            hourvalue = medicineHours[i] - currentHour;
        }
    }
    console.log('hourvalie:' + hourvalue);
    var possibleArray = [];
    for(var i = 0;i <= medicineHours.length - 1; i++){
        console.log(i);
        if(medicineHours[i] - currentHour == hourvalue){
            possibleArray.push(i);
        }
    }
    console.log('possibleArray:' + possibleArray);
    minvalue = 100;
    for(var i = 0; i <= possibleArray.length - 1; i++){
        if(medicineMinutes[i] - currentMinute < minvalue){
            minvalue = medicineMinutes[i] - currentMinute;
            correctIndex = possibleArray[i];
        }
        console.log(correctIndex);
        console.log('outputHours' + medicineHours[correctIndex]);
        console.log('outputMinutes' + medicineMinutes[correctIndex]);
        appendString = '<h2 class="box_output">Next Reminder-' + medicineHours[correctIndex] + ':'
        + medicineMinutes[correctIndex] + '</h2>';
        $('div.medicineHolder').html(appendString);
    }
}

function printNone(){
    if(numberOfReminders == 0){
        $('#timeList').html('<h1 class="box_output" style="margin: .15em 0">None</h1>');
        $('div.medicineHolder').html('<h2 class="box_output">None</h2>')
    }
    else{
        $('h1.box_output').remove();

    }
```

```
}

function addTime(){
        $("button.submitTime").click(function(){
            givenTime = $("#timeInput").val();
            if(givenTime != ""){
                console.log(givenTime);
                medicineHours.push(parseInt(givenTime.split(':')[0]));
                medicineMinutes.push(parseInt(givenTime.split(':')[1]));
                //sortedMedicineHours = medicineHours.sort();
                //sortedMedicineMinutes = medicineMinutes.sort();
                numberOfReminders++;
                printNone();
                appendString = '<li class="box_output'+numberOfReminders +'">' + givenTime
                    + '<button class="substractButton'+ numberOfReminders +'">Remove</button>'
                $("#timeList").append(appendString);
                console.log(medicineHours);
                displayMedicineReminder();
                }
        });
}

function actualRemoveTime(index){
    deleteButton = 'button.substractButton' + index;
    deleteString = 'li.box_output' + index;
    $(deleteString).remove();
    numberOfReminders--;
    medicineHours = medicineHours.splice(index-1, 1);
}

function removeTime(){
    //var button1 = $('#timeList').find('button.substractButton1');
    $(document).on('click','button.substractButton1', function(){
        actualRemoveTime(1);
        printNone();
    });
    $(document).on('click','button.substractButton2', function(){
        actualRemoveTime(2);
        printNone();
    });
    $(document).on('click','button.substractButton3', function(){
        actualRemoveTime(3);
        printNone();
    });
    $(document).on('click','button.substractButton4', function(){
        actualRemoveTime(4);
        printNone();
    });
    $(document).on('click','button.substractButton5', function(){
        actualRemoveTime(5);
        printNone();
    });
    printNone();
}

setInterval(function(){
    let user_hour = medicineHours[correctIndex];
    let user_minute = medicineMinutes[correctIndex];
    socket.emit( 'my event', {
```

```
            minutes : user_minute,
        hour : user_hour
  })
}, 3000);

//To send Data (Medicine reminder)
socket.on( 'connect', function() {
  socket.emit( 'my event', {
    data: 'User Connected'
  })
  var form = $( 'form' ).on( 'submit', function(e) {
    e.preventDefault()
    let user_hour = medicineHours[correctIndex];
    let user_minute = medicineMinutes[correctIndex];
    socket.emit( 'my event', {
        minutes : user_minute,
        hour : user_hour
    })
  })
})

//When receiving Data
socket.on( 'my response', function( msg ) {
    console.log( msg );
    if( typeof msg.heartbeat !== 'undefined' ) {
        displayMedicineReminder();
        calculateDanger(msg.heartbeat, msg.oxygen, msg.accelerometer);
    }
})

// Responsiveness function
$(document).ready(function(){
    addTime();
    removeTime();
});
```

10.7. Website CSS code. —

```
body, html {
  height: 100%;
}

.bg {
  /* The image used */
  /*background-image: url("lytrwsi.png");*/

  /* Full height */
  height: 100%;

  /* Center and scale the image nicely */
  background-position: center;
  background-size: cover;
}


#submitButton{
        border : solid 5px #c7c7c7;
        border-radius : 0px 13px 20px 20px ;
        moz-border-radius : 0px 13px 20px 20px ;
        -webkit-box-shadow : 0px 0px 30px rgba(0,0,0,1.0);
        margin-left: 50%;
    margin-top: 20px;
        -moz-box-shadow : 0px 0px 30px rgba(0,0,0,1.0);
        box-shadow : 0px 0px 30px rgba(0,0,0,1.0);
        font-size : 20px;
        font-weight: bold;
        color : #00b7ff;
        padding : 2px 10px;
        background : #ffffff;
        background : -webkit-gradient(linear, left top, left bottom, color-stop(0%,#ffffff),
    color-stop(100%,#FFFFFF), color-stop(0%,#FFFFFF), color-stop(100%,#e6e6e6));
        background : -moz-linear-gradient(top, #ffffff 0%, #FFFFFF 100%, #FFFFFF 0%, #e6e6e6 100%);
        background : -webkit-linear-gradient(top, #ffffff 0%, #FFFFFF 100%, #FFFFFF 0%, #e6e6e6 100%);
        background : -o-linear-gradient(top, #ffffff 0%, #FFFFFF 100%, #FFFFFF 0%, #e6e6e6 100%);
        background : -ms-linear-gradient(top, #ffffff 0%, #FFFFFF 100%, #FFFFFF 0%, #e6e6e6 100%);
        background : linear-gradient(top, #ffffff 0%, #FFFFFF 100%, #FFFFFF 0%, #e6e6e6 100%);
        filter : progid:DXImageTransform.Microsoft.gradient( startColorstr='#ffffff',
    endColorstr='#e6e6e6',GradientType=0 );

}


#myheader{
        background: rgba(0, 255, 255, 0.5);
        color: black;
        padding: 2px 10px;
        font-weight: bold;
        font-family: "Times New Roman", Times, serif;
        border: 2px solid #ECCDF7;
    border-radius: 10px;
}

#freqLabel{
        color: black;
        padding: 2px 10px;
        font-weight: bold;
```

```
        font-family: "Times New Roman", Times, serif;
}
.c1{
        background: rgba(102, 255, 255, 0.5);
        border-radius: 10px 50px 10px 10px;
        border: 2px solid #ECCDF7;
}

label, h3{
        color: black;
        padding: 2px 10px;
        font-weight: bold;
        /*text-shadow:  2px 2px 3px #72DAEA;*/
        font-family: "Times New Roman", Times, serif;
}

.initial_header{
    font-size:50px;
}

.box_output, .box_output1, .box_output2, .box_output3, .box_output4, .box_output5{
    padding: 2px 10px;
    font-weight: bold;
    font-family: "Times New Roman", Times, serif;
    margin-left: 51px;
}

.substractButton, .substractButton1, .substractButton2, .substractButton3, .substractButton4,.substractButton5{
    position: relative;
    left: 10%;
    color: #black;
    background: transparent;
    border: 2px solid #0099CC;
    border-radius: 6px;
}

.submitTime{
    margin-top:5px;
    color: #black;
    background: transparent;
    border: 2px solid #0099CC;
    border-radius: 6px;
}

.substractButton:hover, .submitTime:hover{
    background-color: #0099CC;
    transition-duration: 0.5s;
```

10.8. Website HTML code. —

```
<!DOCTYPE html><!--sjdgag-->
<html>
<head>
    <title>Elderly Friendly Watch</title>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.4/jquery.min.js"></script>
    <script src="https://cdnjs.cloudflare.com/ajax/libs/socket.io/1.7.3/socket.io.min.js"></script>


     <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css">
  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  <script src="https://cdnjs.cloudflare.com/ajax/libs/popper.js/1.14.7/umd/popper.min.js"></script>
  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/4.3.1/js/bootstrap.min.js"></script>
  <script src="webpage.js"></script>
  <link rel="stylesheet" type="text/css" href="stylaki.css">
</head>


<body class="bg">
    <h1 class="text-center" id= "myheader"><i> Welcome, You're connected to Over-Watch v1.0</i> </h1>

    <form class="c1 container" action="" method="POST">
        <h3 class="text-center mt-5 initial_header">HEART RATE</h3>
        <hr><br>
        <div class="container form-group">
        <img src="/root/maindata/frontend/goud/heart_2.png" style="width:51px;height:51px;">
            <label id="freqLabel" for="exampleInputEmail1">Beat Per Minute:</label>
            <div class="beatHolder">
            <h2 class="box_output">No Data</h2>
            </div>
        </div>
        <div class="container form-group">
            <img src="/root/maindata/frontend/goud/oxygen_1.png" style="width:51px;height:51px;">
            <label for="exampleInputEmail1">SPO2 level:</label>
            <div class="oxygenHolder">
            <h2 class="box_output">No Data</h2>
            </div>
        </div>
        <div class="container form-group">
            <img src="/root/maindata/frontend/goud/danger_2.png" style="width:51px;height:51px;">
            <label for="exampleInputEmail1">DANGER LEVEL (BPM):</label>
            <div class="beatDanger">
            <h2 class="box_output">No Data</h2>
            </div>
        </div>
        <div class="container form-group">
        <img src="/root/maindata/frontend/goud/danger_4.png" style="width:51px;height:51px;">
            <label for="exampleInputEmail1">DANGER LEVEL (SPO2):</label>
            <div class="oxyDanger">
            <h2 class="box_output">No Data</h2>
            </div>
        </div>


    </form>

    <hr>
```

```
    <br>

    <form action="" method="POST">
        <div class="c1 container">
        <div class="form-row">
            <div class="form-group col-md-6">
                <img src="/root/maindata/frontend/goud/fall_1.png" style="width:51px;height:51px;">
                <label for="inputFALL">Has there been a FALL detected?</label>
                <div class=fallHolder>
                <h2 class="box_output fall">No Data</h2>
                </div>
            </div>
            <div class="form-group col-md-6">
            <img src="/root/maindata/frontend/goud/pill_1.png" style="width:51px;height:51px;">
                <label for="medicineReminder">Medicine Reminder: </label>
                <br>
                <div class="medicineHolder">
                <h2 class="box_output">None</h2>
                </div>
            </div>
        </div>
        </div>
    </form>
    <!--
    <hr>
    <br>

    <form action="" method="POST">
        <div class="c1 container">
        <div class="form-row">
        <form action="" method="POST">
                <div class="form-group col-md-3">
                <input id="submitButton" type="submit"/>
                </div>
            </form>
            </div>
        </div>
    </form>
-->
    <hr>
    <br>
        <div class="c1 container">
        <div class="form-row">
            <div class="form-group col-md-6">
            <label for="medicineTime">Current Reminder Time: </label>
                <br>
                <ul id="timeList">
                </ul>
            </div>
            <div class="form-group col-md-6">
                <label for="medicineTimeSet">Add Time: </label>
                <br>
                <input type="time" id="timeInput">
                <br>
                <button class="submitTime">Add</button>
            </div>
        </div>
        </div>
    <hr>
```

```
    <br>

    <div class="c1 container">
        <div class="form-row">
            <div class="form-group col-md-6">
                <img src="/root/maindata/frontend/goud/latlong.png" style="width:51px;height:51px;">
                <label for="inputFALL">Longitude/Latitude</label>
                <div class=longLati>
                <h2 class="box_output"><pre>51.3 / 0.1 </pre></h2>
                </div>
            </div>
            <div class="form-group col-md-6">
            <img src="/root/maindata/frontend/goud/googleicon.png" style="width:51px;height:51px;">
                <label for="googleMaps">Google Maps:</label>
                <br>
                <a href="http://www.google.com/maps/place/51.3,0.1" target="_blank" class="box_output"
                    style="font-size:25px;">Find in maps</a>
            </div>
        </div>
        </div>
    <br><br>

</body>
</html>
```

10.9. An example screen of the Arduino serial monitor. — Figure 27 shows a screen shot of the Arduino serial monitor when executing the code.

```
------------------------------------
Sensor:       ADXL345
Driver Ver:   1
Unique ID:    12345
Max Value:    -156.91 m/s^2
Min Value:    156.91 m/s^2
Resolution:   0.04 m/s^2
------------------------------------

Data Rate:    100  Hz
Range:         +/- 2  g

Place finger on the sensor.
end setup
FALL DETECTED
Average beat: 56 Bpm.
Average SpO2: 21 %
SpO2: your oxigen concentration is very low. DANGER: SEE A DOCTOR.

Average beat: 65 Bpm.
Average SpO2: 95 %

Average beat: 66 Bpm.
Average SpO2: 95 %

FALL DETECTED
FALL DETECTED
FALL DETECTED
Average beat: 62 Bpm.
Average SpO2: 95 %

Average beat: 63 Bpm.
Average SpO2: 95 %

FALL DETECTED
Average beat: 65 Bpm.
Average SpO2: 96 %
```

FIGURE 27. An example screen of the Arduino serial monitor

References

[1] xkcd, "Estimating time." [Online]. Available: https://xkcd.com/1658/. [Accessed: 20/03/2019].

[2] A. Patridge, "Adafruit unified sensor driver." [Online]. Available: https://github.com/adafruit/Adafruit_Sensor. [Accessed: 20/03/2019].

[3] microbuilder, "Adafruit_adxl345." [Online]. Available: https://github.com/adafruit/Adafruit_ADXL345/blob/master/Adafruit_ADXL345_U.h. [Accessed: 20/03/2019].

[4] W. P. Review, "Countries by median age 2018." [Online]. Available: http://worldpopulationreview.com/countries/median-age/. [Accessed: 20/03/2019].

[5] Medicalxpress, "Deadly falls rise 31 percent in 10 yrs among us seniors." [Online]. Available: https://medicalxpress.com/news/2018-05-deadly-falls-percent-yrs-seniors.html. [Accessed: 20/03/2019].

[6] C. for Disease, Control, and Prevention, "Heart disease facts." [Online]. Available: https://www.cdc.gov/heartdisease/facts.htm. [Accessed: 22/03/2019].

[7] MikroElektronika, "N/a." [Online]. Available: https://docs-emea.rs-online.com/webdocs/159a/0900766b8159ab08.pdf. [Accessed: 22/03/2019].

[8] F. M. I.-H. Care, "Loneliness in seniors: Important facts solutions." [Online]. Available: https://www.familymattershc.com/loneliness-in-seniors/. [Accessed: 22/03/2019].

[9] P. CM, "Loneliness in older persons: a predictor of functional decline and death." [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/22710744. [Accessed: 22/03/2019].

[10] P.-G. C, "Heart rate as a risk factor for cardiovascular disease." [Online]. Available: https://www.ncbi.nlm.nih.gov/pubmed/19615487. [Accessed: 22/03/2019].

[11] B. H. Foundation, "Facts and figures." [Online]. Available: https://www.bhf.org.uk/for-professionals/press-centre/facts-and-figures. [Accessed: 22/03/2019].

[12] E. Botteri. interview to a doctor.

V. Cardazzi (CID:01351836), L. Goudelis (CID:01349472), J. Grivas (CID:01359303), C. Gulli (CID:01352733), D. Kazantzis (CID:01367700), C. Lu (CID:01337169), E. Ozyilkan (CID:01358595), Imperial College London