BACHELOR THESIS
COMPUTING SCIENCE

RADBOUD UNIVERSITY

Information Technology Support for the Arterial Thoracic Outlet Syndrome

Author:
Dave Artz
s4475712

First supervisor/First assessor:
dr. P.W.M. (Pieter) Koopman
pieter@cs.ru.nl

Second assessor:
M. (Mart) Lubbers
m.lubbers@cs.ru.nl

June 26, 2020

Abstract

A person is diagnosed with the arterial thoracic outlet syndrome when an arterial compression occurs in the space between the first rib and the collarbone called the thoracic outlet. A first rib resection can relieve the pressure in the thoracic outlet, but this surgery is not suitable for every patient. In this thesis we research how information technology can support patients with ATOS who suffer from a complete compression of the subclavian artery in certain physical positions. With a microcontroller, a photoplethysmography sensor and an accelerometer we develop a prototype with three features. First the prototype detects a compression and activates an alarm so that the patient can change position for decompression. Secondly the prototype captures the rotation of the upper arm to give the patient some insight in critical physical positions. Thirdly the prototype sends a distress signal to a server over Wi-Fi when the patient does not respond to the alarm within an extended period of time.

Contents

Chapter 1

Introduction

The thoracic outlet is the space between the collarbone and the first rib. With the Thoracic Outlet Syndrome (TOS) this space can get narrowed which places pressure on nerves and blood vessels as is illustrated in figure 1.1. 95% of all cases of TOS concerns the compression of nerves, only 3 to 5 percent the compression of veins and only 1 to 2 percent concerns arterial compression.[6] In the latter case the subclavian artery is compressed which is known as the Arterial Thoracic Outlet Syndrome (ATOS).

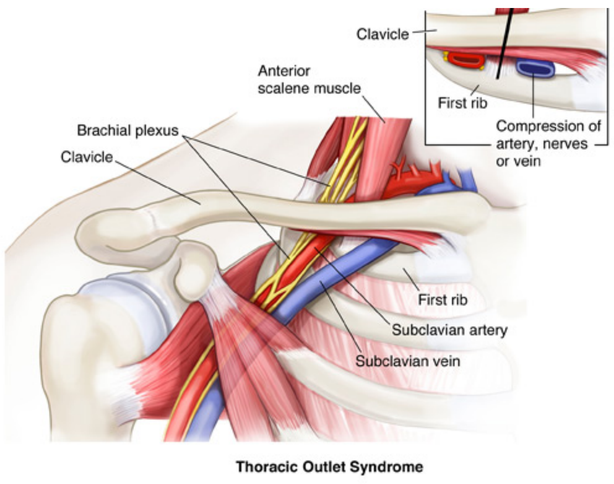


Figure 1.1: Thoracic Outlet Syndrome[1]

The compression of the subclavian artery can have serious complications. It can cause thrombosis where the entire limb may be diffusely swollen.[9] In the worst case the clot dislodges and causes a pulmonary embolism, a stroke or a heart attack. In other cases ATOS can lead to Critical Limb Ischaemia

[1]Thoracic Outlet Syndrome [Online image]. Sport Med School.
http://sportmedschool.com/thoracic-outlet-syndrome/

(CLI) as blood flow is markedly reduced to the hands.[3] Left untreated, the complications of CLI may result in the amputation of the affected limb.

A patient can be diagnosed with Doppler ultrasonography, X-ray or CTA.[2] Provocation tests are also used for diagnosing. Here physical positions are tested with the Roos, Elvey and Adson's test to find compressions of the subclavian artery.[8] A patient with ATOS can be treated with a first rib resection, which relieves the pressure in the thoracic outlet.[7]

Due to the lack of consensus for diagnostic testing the incidence of TOS is unknown. Some authors claim that 3 to 80 out of 1000 people can suffer from TOS.[5] Approximately of all TOS patients only 1 percent suffers from ATOS. Not all patients qualify for the first rib resection as it is a drastic surgery. The patients who do undergo the surgery would not be operated immediately. Thus at least at some point all patients with ATOS are at risk for complications after they are diagnosed.

In this thesis we research how information technology can support a patient with ATOS. As with other syndromes the causal factors and symptoms may vary per patient. For this research we assume there is only an arterial compression in certain physical positions and we only focus on a complete compression of the subclavian artery. The information technology support in this research is a prototype in the form of a microcontroller. With this microcontroller, a photoplethysmography sensor and an accelerometer we develop three features.

The first feature is an alarm that informs the patient that the subclavian artery is completely compressed. The moment of compression is of most interest because then the patient is at risk for complications. If the prototype detects the compression and activates the alarm then the patient can change position for decompression. Avoiding positions in which complete compression takes place will minimize some risks on complications. The second feature is the orientation capture of the upper arm. If the prototype captures the orientation at the time of compression then this information helps the patient recognize critical physical positions to avoid. The third feature is a distress signal to a server over a Wi-Fi. This message can alarm others when the patient does not respond to the compression alarm.

The features that we develop in this thesis are not remarkable on their own. However, combined they can result in a product that solves an unique problem. Others have used similar technologies to solve different problems. Thus in this thesis we do not develop a new technology, but we demonstrate how one can use existing technologies to tackle a specific problem.

In the preliminaries we describe the techniques and components that we use for the prototype. Then the research section of chapters 3, 4, and 5 contain the features of compression alarm, position capture and the distress signal respectfully. In chapter 6 we combine the features into a single prototype. Chapter 7 covers the related work and in chapter 8 we describe our conclusions.

Chapter 2

Preliminaries

This chapter introduces the basic concepts that are necessary to comprehend other components of the research. First we explain the technique photoplethysmography. Then we list the components of the prototype and finally we show which outputs we are use and how we initialize them.

2.1 Photoplethysmography

Photoplethysmography (PPG) is an optical technique that measures light absorption. A simple PPG sensor will shine some light on a surface and find out how much light this surface has absorbed by monitoring the amount of reflected light. In this research we place an infrared light-emitting diode (LED) on tissue as is illustrated in figure 2.1. Concurrently the photodiode monitors how much light is reflected from the tissue and converts it to an electric current. We can use this technique to monitor the pulse because the infrared light is absorbed by the blood. Therefore the voltage signal from the sensor is proportional to the quantity of blood flowing through the blood vessels.

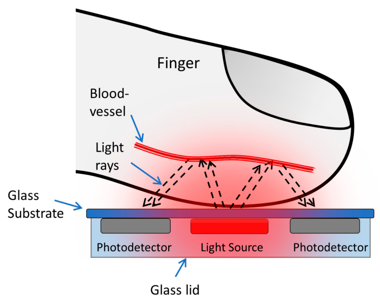
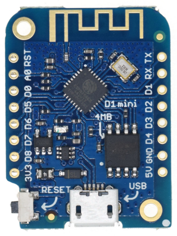


Figure 2.1: Photoplethysmography[4]

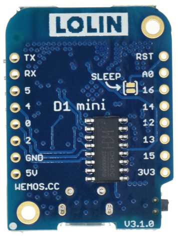
2.2 Components

For the prototype we use the following 5 components.

1. Wemos D1 Mini V3
   This is a mini breakout board with 4MB flash memory and a ESP8266 microchip. The ESP8266 is a Wi-Fi microchip with a full TCP/IP stack. It is compatible with MicroPython, Arduino and Nodemcu.



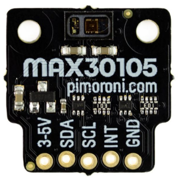
(a) front



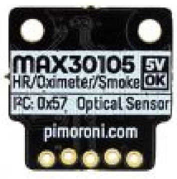
(b) back

Figure 2.2: Wemos D1 Mini V3

2. MAX30105 Breakout
   The MAX30105 breakout from Pimoroni is a heart rate, oximeter, and particle sensor. The sensor has photodetectors and a green, red and infrared LED. We use the sensor to monitor the heart rate with the infrared LED and a photodiode. Unfortunately this PPG sensor does not meet the requirements to be used for medical diagnosing. Therefore it is important to keep in mind that we only use the sensor for educational purposes.



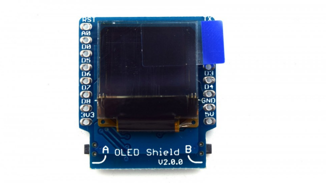
(a) front



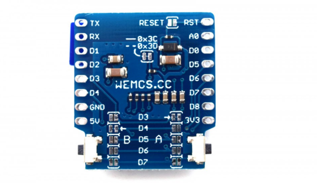
(b) back

Figure 2.3: MAX30105 Breakout

3. Wemos D1 Mini OLED shield
   This 0.66 inch OLED has a screen size of 64 x 48 pixels. We use the OLED to display the state of the program, the average beats per minute and the rotation dimensions of the upper arm.



(a) front

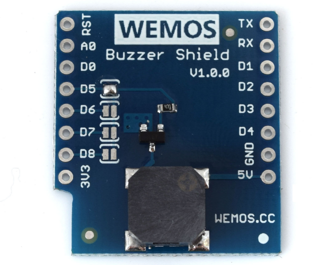


(b) back

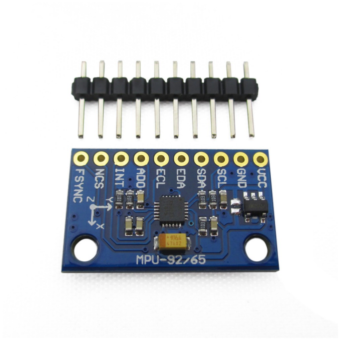
Figure 2.4: Wemos D1 Mini OLED shield

4. Wemos D1 Mini Buzzer shield
   This shield has a passive buzzer. We use the buzzer for the compression alarm which is set at the time of a complete compression of the subclavian artery.

5. MPU-9265
   The MPU-9265 contains a MPU-6500 (accelerometer and a gyroscope) and a HMC5883L (magnetometer) chip. We use the accelerometer to determine two dimensions of the upper arm rotation in chapter 4.



(a) Wemos D1 Mini Buzzer shield



(b) MPU 9265

Figure 2.5: Wemos D1 Mini Buzzer shield and MPU 9265

Figure 2.6 contains an image of the assembled prototype. The ESP8266 is not visible on the image as it is below the OLED shield. The prototype is sewn on a black wrist brace so that it can be carried around easily.

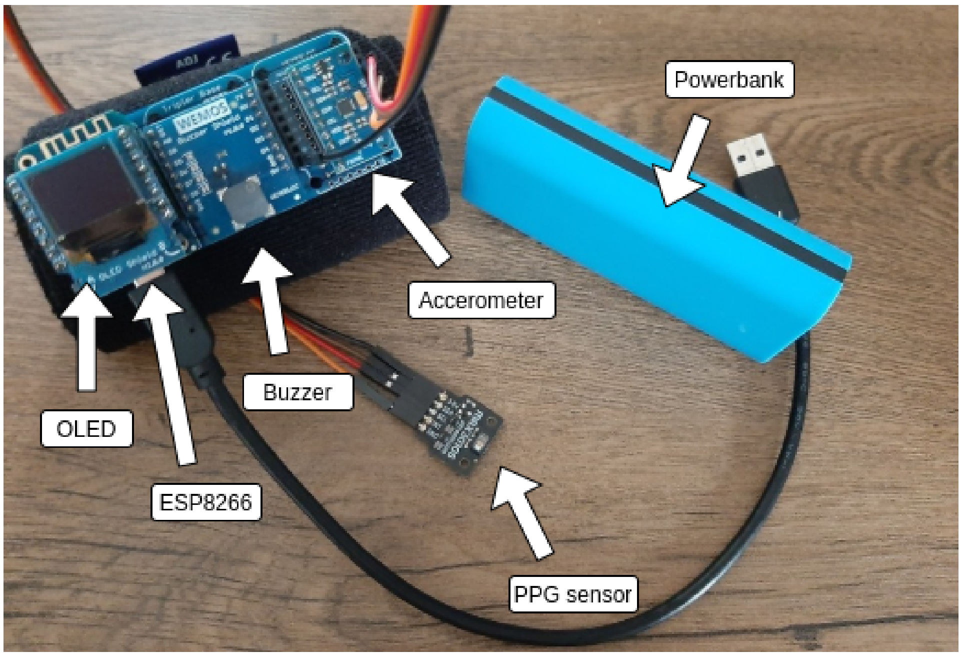


Figure 2.6: Prototype

2.3 Output

In this research we use the Arduino programming language, which is C++ with some domain-specific libraries.[2] The prototype uses three kinds of output. We use the OLED, the buzzer and Wi-Fi.

To demonstrate the use of the OLED we include an "Hello World!" example in listing 2.1. We use the `Adafruit_SSD1306.h` library for a 128x64 OLED(1).[3] This is not a problem for our 66 inch OLED as we take the difference into account. We use the `Adafruit_GFX.h` library to display letters and numbers(2).[4] In order to display something we initialize the display with the width, height and the reset pin(3). Before we print, we clear the display and set the text color and size(5-7). Then we set the cursor and define the string to be printed(8-9). Finally we set the display to print the string "Hello World!"(10).

[2]Arduino. 2020. *Language Reference*. https://www.arduino.cc/reference/en
[3]Adafruit. 2019. *Adafruit_SSD1306*. https://github.com/adafruit/Adafruit$_SSD$1306
[4]Adafruit. 2020. *Adafruit_GFX*. https://github.com/adafruit/Adafruit-GFX-Library

Listing 2.1: OLED

```
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
Adafruit_SSD1306 display(128, 48, &Wire, -1);
display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
display.clearDisplay();
display.setTextColor(WHITE);
display.setTextSize(1);
display.setCursor(32, 0);
display.println("Hello World!");
display.display();
```

We use the buzzer to alarm the user. An example of the buzzer can be found in listing 2.2. On the Wemos D1 Mini the default buzzer pin is D5(1). To activate the buzzer we use the function `tone()` with a frequency of 1000 to our pin(2). This frequency is chosen for no particular reason. We use the function `noTone()` to deactivate the buzzer(3).

Listing 2.2: buzzer

```
1  const int buzzerPin = D5;
2  tone(buzzerPin, 1000);
3  noTone(buzzerPin);
```

In chapter 5 we send a distress signal over Wi-Fi. In listing 2.3 is an example of how we connect with an access point. We use the `ESP8266WiFi.h` library to connect with a network that requires authentication(1).[5] We define the local SSID and PSK and use them in `Wifi.begin()`(2-4). In the setup the program waits for the connection to be completed before entering the main loop(5-6).

Listing 2.3: Wi-fi

```
1  #include <ESP8266WiFi.h>
2  #define SSID "network-name"
3  #define PSK "network-password"
4  WiFi.begin(SSID, PSK);
5  while (WiFi.status() != WL_CONNECTED)
6       delay(500);
```

[5]Arduino. 2019. *ESP8266WiFi.*
https://github.com/esp8266/Arduino/blob/master/libraries/ESP8266WiFi/src/ESP8266WiFi.h

Chapter 3

Compression alarm

In this research we assume that the patient with ATOS is only at risk for complications when there is a complete compression in certain physical positions. This rules out the patients who suffer from a partial compression or a full-time compression of the subclavian artery. One of the complications we like to prevent is thrombosis, which can be the result of a long lasting complete compression. Therefore a patient with ATOS can get thrombosis overnight by sleeping in the wrong position. In this chapter we want to alarm the patient when there is a complete compression, so that the patient can change position for decompression.

At the moment of a complete compression the blood can not flow to the arm through the artery, which means that there is no pulse. We choose to focus on the complete compression because we can detect the heartbeat with a PPG sensor. If the sensor doesn't detect a heartbeat for an extended period of time then we alarm the patient.

In section 3.1 we explain how we detect a heartbeat, in section 3.2 we verify the correctness of this method and in section 3.3 we set the alarm upon compression.

3.1 Heartbeat detection

PPG can be used to detect a heartbeat as we explained in section 2.1. Of the sensor we only use the infrared LED and the photodiode. Tissue scatters and absorbs infrared wavelengths so that, in order to have a measurable signal, a thin part of the body must be used. Therefore we place the sensor on the index finger.

To find a heartbeat we retrieve the immediate infrared value from the PPG sensor. This values represents the amount of reflected infrared light by a surface. The code snippet for retrieving the value can be found in listing 3.1. For the sensor we use the `MAX30105.h` library of

SparkFun(1).[6] Some parts of the code in this chapter are directly extracted from the libraries heart rate example.[7] The initialization takes places in the function `initPPGSensor()`. Here we turn the red LED to low to indicate that the sensor is running(7) and the green LED is turned off because it has no use in this research(8). After the setup is done we retrieve the immediate infrared value with the function `getIR()`(10).

Listing 3.1: PPG sensor

```
1  #include "MAX30105.h"
2  MAX30105 ppgSensor;
3  void initPPGSensor()
4  {
5    ppgSensor.begin(Wire, I2C_SPEED_FAST);   // 400KHz speed
6    ppgSensor.setup();
7    ppgSensor.setPulseAmplitudeRed(0x0A);    // turn red LED to low
8    ppgSensor.setPulseAmplitudeGreen(0);     // turn off green LED
9  }
10 long irValue = ppgSensor.getIR();
```

We can use the immediate infrared value to detect a heartbeat with the function `checkForBeat()` from Sparkfun's `heartRate.h` library.[8] The function contains an implementation of the peripheral beat amplitude algorithm of Maxim Integrated Products Inc., which detects a heartbeat based on zero crossing(5).[9] The code snippet for this can be found in listing 3.2. As we need to monitor the heart rate constantly we retrieve the infrared value at the start of the loop(4).

With the functionality to recognize a heartbeat we have the potential to set an alarm upon complete compression if we add the element of time. Before we do that in section 3.3 we verify the correctness of the function `checkForBeat()` with the input of the immediate infrared value in section 3.2.

Listing 3.2: heartbeat

```
1  #include "heartRate.h"
2  void loop()
3  {
4      long irValue = ppgSensor.getIR();
5      if (checkForBeat(irValue))
6           ...
7  }
```

[6]Sparkfun. 2018. *MAX30105 library.* https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library/blob/master/src/MAX30105.h

[7] Sparkfun. 2018. *Heart rate example.* https://github.com/sparkfun/SparkFun_MAX3010x_Sensor_Library/tree/master/examples/Example5_HeartRate

[8]Sparkfun. 2016. *Heart rate library.* https://github.com/sparkfun/MAX30105_Particle_Sensor_Breakout/blob/master/Libraries/Arduino/src/heartRate.h

[9]Maxim Integrated Products, Inc. 2016. *MAX30105 Evaluation Kit.* https://datasheets.maximintegrated.com/en/ds/MAX30105ACCEVKIT.pdf

3.2 Verification

We need to verify that the function `checkForBeat()` with the input of the immediate infrared can detect a heartbeat correctly before we use it in the prototype. We do this by calculating the beats per minute (BPM) and comparing the results with another heart rate device.

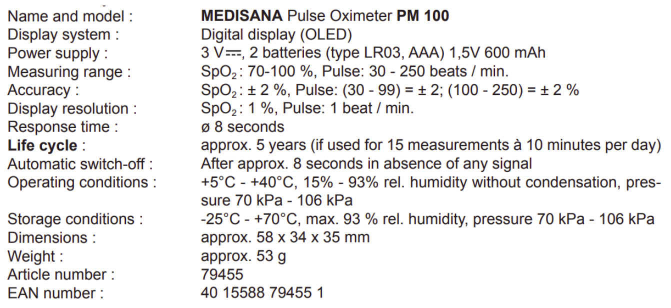
We calculate the BPM by keeping track of the interval between heartbeats. Most of the code in this section is copied from Sparkfun's heart rate example and can be found in listing 3.3.[7] We calculate the average BPM in the function `bpm()` after a heartbeat is detected(5). The variable `wave` represents the interval since the last heartbeat. To calculate the beats per minute we simply divide 60 seconds by the value of `wave`. We store that value in the variable `beatsPerMinute`(10).

The patient must apply constant pressure to receive reliable data from the PPG sensor. When the patient moves around the sensor may give divergent readings. To handle this we calculate the average BPM out of the last 8 beats. To filter out erroneous values we check if the `beatsPerMinute` value is within the possible range of 30 to 255(13). Then the value is stored in an array and from that array the average BPM is calculate(16-23).

Listing 3.3: beats per minute

```
 1 const byte RATE_SIZE = 8;    // number of bpm for average
 2 byte rates[RATE_SIZE];       // array of last 8 bpm
 3 byte rateSpot = 0;           // index
 4 long checkpoint = millis();
 5 void bpm()
 6 {
 7   // save checkpoint
 8   long wave = millis() - checkpoint;
 9   reset();
10   beatsPerMinute = 60 / (wave / 1000.0);
11 
12   // filter erroneous values
13   if (beatsPerMinute < 255 && beatsPerMinute > 30)
14   {
15     // store bpm in array
16     rates[rateSpot++] = (byte)beatsPerMinute;
17     rateSpot %= RATE_SIZE;
18 
19     // calculate average
20     beatAvg = 0;
21     for (byte x = 0; x < RATE_SIZE; x++)
22       beatAvg += rates[x];
23     beatAvg /= RATE_SIZE;
24   }
25 }
```

To compare the average BPM of the prototype with another device we display the value on the OLED. This other heart rate device is the Medisana pulse oximeter PM 100.[10] In figure 3.1 the technical details of this device are included. The accuracy of the pulse between 30 and 99 BPM is approximately 2.



Name and model :	**MEDISANA** Pulse Oximeter **PM 100**
Display system :	Digital display (OLED)
Power supply :	3 V⎓, 2 batteries (type LR03, AAA) 1,5V 600 mAh
Measuring range :	SpO_2 : 70-100 %, Pulse: 30 - 250 beats / min.
Accuracy :	SpO_2 : ± 2 %, Pulse: (30 - 99) = ± 2; (100 - 250) = ± 2 %
Display resolution :	SpO_2 : 1 %, Pulse: 1 beat / min.
Response time :	ø 8 seconds
Life cycle :	approx. 5 years (if used for 15 measurements à 10 minutes per day)
Automatic switch-off :	After approx. 8 seconds in absence of any signal
Operating conditions :	+5°C - +40°C, 15% - 93% rel. humidity without condensation, pressure 70 kPa - 106 kPa
Storage conditions :	-25°C - +70°C, max. 93 % rel. humidity, pressure 70 kPa - 106 kPa
Dimensions :	approx. 58 x 34 x 35 mm
Weight :	approx. 53 g
Article number :	79455
EAN number :	40 15588 79455 1

Figure 3.1: Medisana Pulse Oximeter PM 100

To compare the average BPM we measure the pulse of the same arm on both devices at the same time. The setup can be seen in figure 3.2. The index finger is placed on the PPG sensor and the middlefinger in placed in the Medisana pulse oximeter. The middlefinger is placed inside the device as the Medisana pulse oximeter is a transmissive instead of reflective sensor. This means that the the LED is on one side and the photodiode is on the other.

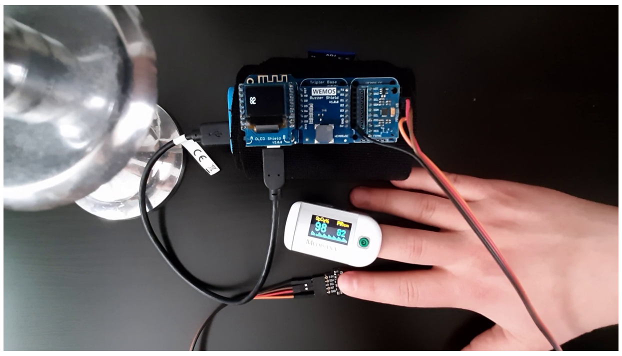


Figure 3.2: BPM test setup

To collect the values we record a video of 100 seconds. As all recorded values are between the 70 and 86 BPM we convert the video to images by 2 frames per second. This way we do not miss any changes because $\frac{86}{60} < 2$. The results are plotted below in figure 3.3 and 3.4.

[10]Medisana. 2019. *Pulse Oximeter PM 100.* https://www.medisana.com/en/Health-control/Pulsoximeter/PM-100-Pulseoximeter.html?force_sid=0ikdo4jt9jm8m18cd97bfusj96

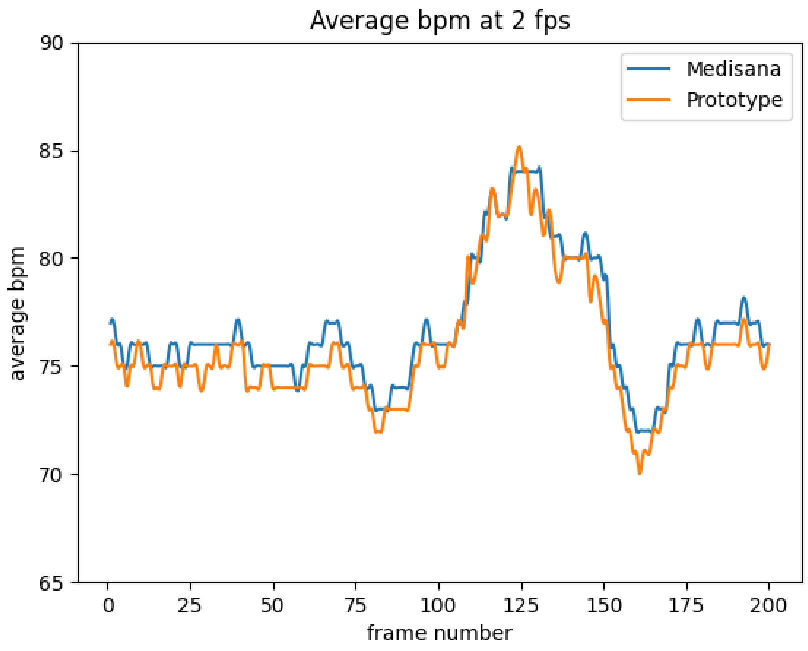


Figure 3.3: Average BPM at 2 fps

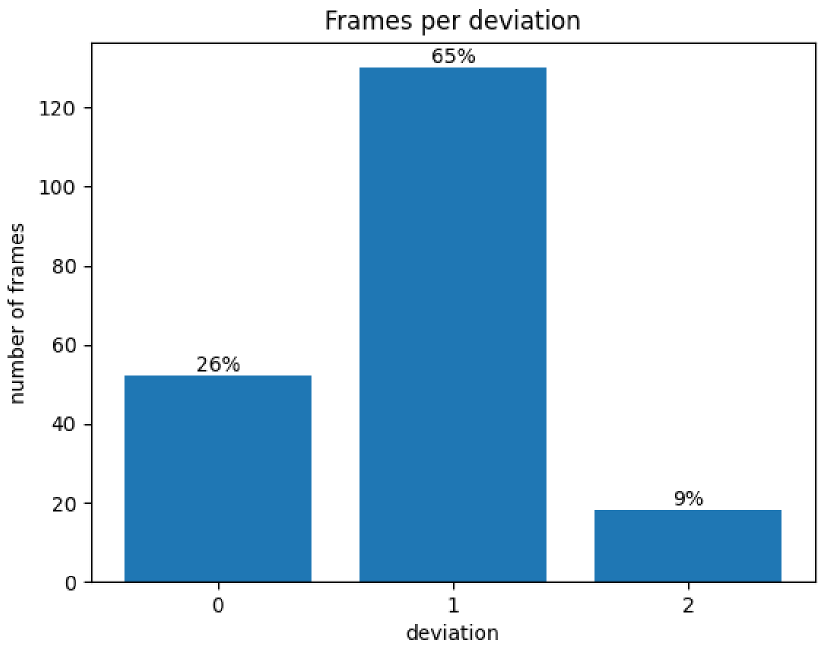


Figure 3.4: Frames per deviation

In figure 3.3 the average BPM per frame is plotted of both the prototype and the Medisana pulse oximeter. Around 50 seconds in, at frame 100, we move the upper arm around to test if both devices respond equivalently to the rise of the pulse. From the graph we can clearly see that the average BPM of the prototype is nearly equivalent to the BPM of the Medisana pulse oximeter.

For figure 3.4 the deviation per frame is calculated in order to find out how accurate the prototype is in comparison to the Medisana pulse oximeter. We find that 26% of the frames have exact the same BPM, 65% of the frames have a deviation of 1 BPM and only 9% of the frames have a deviation of 2 BPM. As the Medisana pulse oximeter has an accuracy of approximately 2 between 30 and 99 BPM we can conclude that the prototype has an accuracy of approximately 4 between 30 and 99 BPM.

With an accuracy of approximately 4 between 30 and 99 BPM we can conclude that the function `checkForBeat()` with the immediate infrared value is a reliable source to detect a heartbeat.

3.3 Alarm

As we have verified the correctness of the heartbeat detection we can use it to set an alarm. The prototype sets the buzzer as alarm when it does not receive a heartbeat within an extended period of time. In this thesis we set this threshold to 30 seconds.

The alarm will be set wrongfully when there is not a finger on the sensor. We use the PPG sensor to prevent these false positives. As not all objects absorb the same amount of infrared light as tissue, the PPG sensor can detect tissue like surface. This helps with the distinction between the state in which a patient is not using the device and the state in which the patient is using the device, but there is no heartbeat detected. Therefore we divide the program into four states, which are illustrated in figure 3.5.

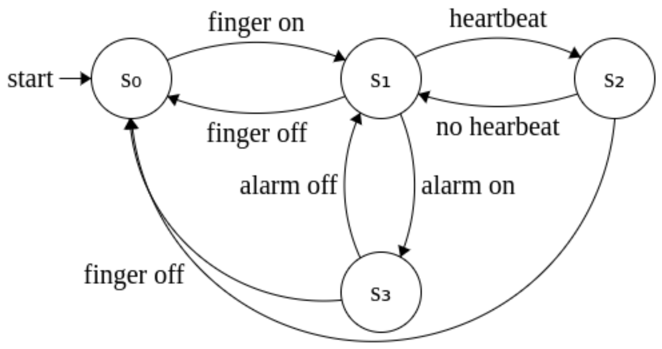


Figure 3.5: State diagram compression alarm

When a patient is not using the device the sensor has no contact with tissue and the program will be in state 0. When the patient places a finger on the sensor, the sensor will recognize tissue and the program will jump to state 1. When the program recognizes a heartbeat it goes into state 2. If the sensor recognizes tissue, but not a heartbeat within the given threshold, the program is in state 3. In this state a compression is detected and the alarm is activated. The alarm will be deactivated by returning to either state 0 or state 1.

The four states of the program are mostly determined by the functions `alarm()` and `oled()` which are included in listing 3.4. In order to recognize tissue the infrared value needs to be at least greater than 50000(13). Thus if the value is lower than 50000 the program knows it is in state 0. In this state the program prints "no contact" on the OLED and the checkpoint resets(15-16). We reset the checkpoint because it represents the time of the last detected heartbeat. If the infrared value is greater than 50000 then the program is in state 1. As the prototype might not recognize a heartbeat during movement of the user, the average BPM is displayed on the OLED in both state 1 and 2(21). To check if the program is in state 3 we calculate the downtime(3). The downtime is the interval since the last checkpoint. As the checkpoint is reset in both state 0 and 2, the downtime represents the interval in which the sensor recognizes tissue, but not a heartbeat. If the downtime is greater than the threshold then the alarm will be set to true and the buzzer turns on(6-7).

Listing 3.4: alarm and oled

```
1  void alarm()
2  {
3    downTime = (millis() - checkpoint) / 1000.0;
4      if (downTime > ALARM_THRESHOLD)
5      {
6        alarmOn = true;
7        tone(BUZZER_PIN, 1000);
8      }
9  }
10 void oled()
11 {
12   // set message
13   if (irValue < 50000)              // state 0
14   {
15     reset();
16     output = "no contact";
17   }
18   else if (alarmOn)                 // state 3
19     output = "no pulse";
20   else                              // state 1 or 2
21     output = String(beatAvg);
22   display.println(output);
23 }
```

In this chapter we managed to detect a heartbeat with an accuracy of approximately 4 between 30 and 90 BPM. We used the heartbeat detection to keep track of the interval since the last heartbeat. If this interval is greater than the threshold of 30 seconds then a compression of the subclavian artery is detected. In this state the alarm is set to warn the patient to change position. We managed to create a program flow that resets the alarm and the heartbeat interval, when a heartbeat is detected or when the device is not actively used.

The complete source code of this feature is included in appendix A. Note that the calculation of the average BPM will be excluded in the final version of the prototype. The BPM is included in this appendix to verify the correctness of the function `checkForBeat()` with the immediate infrared value.

Chapter 4

Orientation capture

In this chapter we aid the patient with knowledge about critical upper arm orientations. In chapter 6 the prototype needs to log the orientation of the upper arm at the time of compression. This information gives the patient insight on dangerous orientations to avoid. Therefore in this chapter we capture two dimensions of the upper arm orientation using the accelerometer. To measure the orientation the prototype is attached to the outside of the upper arm. This is the right side for the right arm and the left side for the left arm.

For the three dimensional orientation of the upper arm we use the Euler angles. To visualize this we imagine that the tail of an airplane is attached to the shoulder and the nose to the elbow. We then need three rotations to determine the orientation of the upper arm. As illustrated in figure 4.1 these rotations are also known as the roll, pitch and yaw.

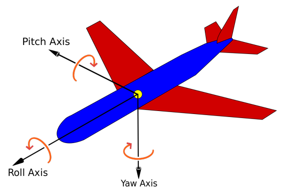


Figure 4.1: Rotations[11]

With the accelerometer we can only calculate the roll and pitch. The yaw can't be calculate with the accelerometer due to the lack of change in gravity in the z-axis. One can try to calculate the pitch, roll and yaw with a sensor fusion algorithm for 3D orientation using the accelerometer, gyroscope and magnetometer, but in this research we will limit ourselves with the roll and pitch.[1]

[11]An image showing all three axes [Online image]. Wikipedia. 2010. https://en.wikipedia.org/wiki/File:Yaw_Axis_Corrected.svg

In section 4.1 we describe how we retrieve the values from the accelerometer. Section 4.2 then covers how we use these values to calculate the roll and pitch.

4.1 Accelerometer

For the computation of the roll and pitch we need the three axes units from the accelerometer. With a bit of imagination figure 4.2 contains a three dimensional sketch to visualize what results the accelerometer returns. To keep it simple we image that the red balls are attached to a string. When the accelerometer moves in a direction the red balls move forward or backward. A string notices if the ball moves forward or backward and saves this acceleration in two registers. An accelerometer is good for calculating the roll and pitch because there is enough gravity involved as both movements are vertical. The accelerometer is not suitable to calculate the yaw as this movement is only horizontal.

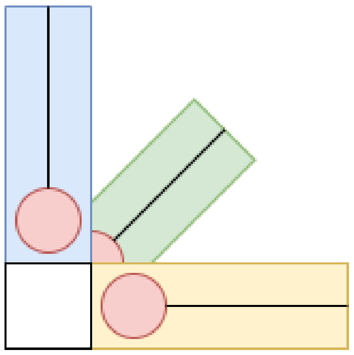


Figure 4.2: 3-axis accelerometer sketch

We activate the accelerometer and read the values of the three axes x, y and z. To do this we implement two functions with the help of the MPU-6500 register map and the Arduino Playground.[12] [13]The first function `void initPositionSensor()` starts the accelerometer of the MPU(3). From the register map follows that the values of x, y and z are saved in register 0x3B up to 0x40 respectively. With that knowledge the second function `getAcceleration()` reads the values from the registers and stores these is in variables `ax`, `ay` and `az`(4-13).

[12]InvenSense Inc. 2013. MPU-6500 Register Map and Descriptions Revision 2.1 https://invensense.tdk.com/wp-content/uploads/2015/02/MPU-6500-Register-Map2.pdf

[13]Arduino Playground. 2018. *MPU-6050 Accelerometer + Gyro.* https://playground.arduino.cc/Main/MPU-6050/

Listing 4.1: accelerometer

```
#define MPU_ADDR 0x68
#define ACC_ADDR 0x3B
void initPositionSensor()

void getAcceleration(int16_t &ax, int16_t &ay, int16_t &az)
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(ACC_ADDR);
  Wire.endTransmission();
  Wire.requestFrom(MPU_ADDR, 6);
  ax = Wire.read() << 8 | Wire.read(); // 0x3B & 0x3C
  ay = Wire.read() << 8 | Wire.read(); // 0x3D & 0x3E
  az = Wire.read() << 8 | Wire.read(); // 0x3F & 0x40
}
```

We noticed some high frequency values that could lead to a deceptive outcome. To prevent these high values corrupt the result we add a filter which is included in listing 4.2. The filter calculates the x, y and z values with 99 percent of their previous value. The filtering factor of 0.01 cuts out all high frequency values as it can only influence the current value with a factor of 0.01.

Listing 4.2: filtering

```
const float F_CONST = 0.99;
const float F_DEVIATION  = 0.01;
void loop()
{
    getAcceleration(xi, yi, zi);
    x = F_CONST * x + F_DEVIATION * float(xi);
    y = F_CONST * y + F_DEVIATION * float(yi);
    z = F_CONST * z + F_DEVIATION * float(zi);
}
```

With the filter factor of 1 percent we need multiple retrievals and updates of the values for a good representation of the acceleration. Therefore we use a counter with threshold 100 and a short delay of 100 milliseconds. The code for this is included below in listing 4.3.

Listing 4.3: counter and delay

```
void loop()
{
    counter++;
    if (counter == 100)
    {
        counter = 0;
        // calculate roll and pitch
    }
    delay(100);
}
```

4.2 Calculation

Now we have gathered filtered acceleration values x, y and z we can calculate the roll and pitch. The roll is the rotation about the x-axis between -180 and 180 degrees. The pitch is the rotation about the y-axis between -90 and 90 degrees. We calculate the roll and pitch in the function `setPosition()` which is included in listing 4.1. The formula is based on rotation matrices, which transform a vector under a rotation by angels in the roll and pitch.[14]

Listing 4.4: calculation roll and pitch

```
const float CONVERT = 180.0 / M_PI; // radians to degree
void setPosition()
{
  roll = int(atan2(y, z) * CONVERT);
  pitch = int(atan2(x, sqrt(y * y + z * z)) * CONVERT);
}
```

We test the accuracy of the orientation by attaching the prototype to a rectangular object. As can be seen in figure 4.3 we use a 4 by 4 rubix cube on a table. The results of the rotation can be found below in table 4.1. The first row represents the expected degrees. We see that the roll and pitch have a maximum deviation of 5 degrees. The deviation might originate from poor calibration.

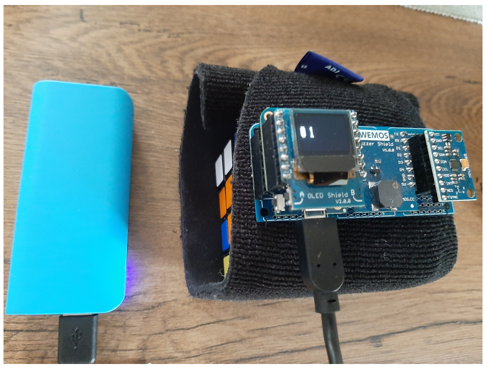


Figure 4.3: Orientation test setup

Table 4.1: Results roll and pitch

Degrees	0	90	-90
Roll	0	84	-89
Pitch	1	85	-86

[14]https://www.nxp.com/docs/en/application-note/AN3461.pdf, roll = equation 25, pitch = euqation 26

In this chapter we managed to use the x, y and z values from the accelerometer to calculate the roll and pitch with a maximum deviation of 5 degrees. This feature can be used to calculate two dimensions of the upper arm rotation. The complete source code of this feature can be found in appendix B. Note that we only calculate the roll and pitch at the time of compression in the final version of the prototype. In this version we display the roll and pitch continually to verify the correctness of the calculation.

Chapter 5

Distress signal

In chapter 3 we implemented the compression alarm. This alarm turns off by itself when a heartbeat is detected. The patient can also disable the alarm by removing the finger from the sensor. If there is no response to the alarm within an extended period of time we assume that the patient is in danger. Therefore in this chapter the prototype sends a signal to a server over Wi-Fi. This can function as a distress signal which will be send when the patient does not respond to the alarm within an extended period of time. The server could then potentially warn others to aid the patient. However, in this thesis we limit the server by only receiving data. Processing and redirecting the data is outside the scope.

In section 5.1 we describe how the prototype sends a signal, in section 5.2 we describe how the server receives this signal and we verify this process in section 5.3.

5.1 Client

The prototype has a ESP8266 Wi-Fi microchip with a full TCP/IP stack. This means that it can connect to a Wi-Fi network and use the Hyper Text Transfer Protocol (HTTP) for requests. The microchip supports IEEE 802.11 standards b, g and n. Therefore it it can operate in both the 5 GHz and the 2.4 GHz band. The microchip can connect to open networks and networks that require WEP or WPA/WPA2 authentication. The upside of using Wi-Fi is that it is easy to implement and it is hard to find a building without Wi-Fi anno 2020. The downside is that the prototype needs to be connected to a network to be able to send a signal. With this prototype the user can only connect to 1 access point as there is no interface for authentication. Therefore in this thesis we assume that the prototype is connected to the same Wi-Fi network at all times.

The signal we send to the server is a HTTP post request. HTTP is used for communication between a client and a server. A post request is used to submit data to a server to be processed. The post data contains a message in JSON format. An example is included in listing 5.1. In this thesis the message contains a user identification number, the timestamp and the two dimensions of the upper arm orientation. We include the orientation because the server could be used to send a message to the patient with a visualization of critical upper arm orientations.

Listing 5.1: JSON message example

```
{
    "user_id":"1",
    "timestamp":"2017-04-04 19:28:23",
    "position":"65-21"
}
```

In section 2.3 we showed how we connect to a network with authentication. We use that connection to send a HTTP request. For this we use the function `send_signal()` which is included in listing 5.2. We craft the request with the help of the `ESP8266HTTPClient.h` library(1).[15] A post requests requires a destination(6). This destination is a domain name or IP address. In section 5.2 we describe how we craft the destination of our local server. In the header we specify what kind of content we send along. The content is of type `"application/json"` as we want to send JSON data(7). To send JSON in a string format we include the backslashes that function as escape characters for the quotes(8). As we do not have any data collected in this version we only send the empty string.

After we send the POST request we save the response value in a variable `httpCode`(9). We use this value in chapter 6 to check if the post request was successful. If the request is unsuccessful then we try again.

Listing 5.2: HTTP post request

```
1  WiFiClient client;
2  HTTPClient http;
3  void sendSignal()
4  {
5    if ((WiFi.status() == WL_CONNECTED))
6    {
7      // Send data with http post request
8      http.begin(client, "http://" SERVER_IP); //HTTP
9      http.addHeader("Content-Type", "application/json");
10     String data = "{\"user_id\":\"1\",\"timestamp\":\"\",\"
           position\":\"\"}";
11     int httpCode = http.POST(data);
12   }
13 }
```

[15]Arduino. 2020. *ESP8266 HTTP Client library*.
https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266HTTPClient

5.2 Server

We use a web server to receive the prototypes HTTP post request data. In this thesis we use the built-in web server of PHP, version 7.5.3.[16] We choose to use this server because it saves the trouble of installing and configuring a full-featured web server like Apache or Nginx. Note that the PHP built-in web server is useful for testing purposes, but it should not be used on a public network. There we run this server on a computer a local network.

To start the server we use the command `php -S 0.0.0.0:8000`. With the address 0.0.0.0 the host is reachable at all it's IP addresses. The server is in the same local network as the prototype operates. Therefore the destination of the client will be the local IP of the server with the specified port, in our case this is port 8000.

When the server receives a post requests it processes the data in order to store it. We process the post request in a PHP script named `process.php`, which is included in listing 5.3. This script store the data in a JSON file named `data.json`. First the data from the request is stored in a variable `post`(2). Then the data from the JSON file is stored in another variable `data`(3). Next we set the timestamp of the request(4). We do this on the server because the client would need a real time clock unit to determine the time. Finally we add the data from the POST request and save it in `data.json`(6-7). This way the JSON file contains all messages send by the prototype.

Listing 5.3: process.php

```
1  <?php
2      $post = json_decode(file_get_contents('php://input'), true);
3      $data = json_decode(file_get_contents("data.json"));
4      $post['timestamp'] = date("Y-m-d H:i:s");
5      array_push($data, $post);
6      $update = json_encode($data);
7      file_put_contents('data.json', $update);
8  ?>
```

[16]PHP. (n.d.). *built-in web server*
https://www.php.net/manual/en/features.commandline.webserver.php

5.3 Verification

We verify that the prototype sends the HTTP post request and that the server stores the data correctly. For this we run a test in which the prototype sends 5 messages with a 5 second interval. As can be seen in listing 5.5 the server received the requests successfully. The `process.php` stores the values correctly in `data.json`, of which the content is included in listing 5.6. Therefore we can conclude that the prototype successfully sends a signal to a server over Wi-Fi.

Listing 5.4: server

```
[Jun 22 15:50:11 2020] x.x.x.x:49392 Accepted
[Jun 22 15:50:12 2020] x.x.x.x:49392 [200]: POST /process.php
[Jun 22 15:50:12 2020] x.x.x.x:49392 Closing
[Jun 22 15:50:17 2020] x.x.x.x:61023 Accepted
[Jun 22 15:50:17 2020] x.x.x.x:61023 [200]: POST /process.php
[Jun 22 15:50:17 2020] x.x.x.x:61023 Closing
[Jun 22 15:50:22 2020] x.x.x.x:52575 Accepted
[Jun 22 15:50:22 2020] x.x.x.x:52575 [200]: POST /process.php
[Jun 22 15:50:22 2020] x.x.x.x:52575 Closing
[Jun 22 15:50:27 2020] x.x.x.x:54005 Accepted
[Jun 22 15:50:27 2020] x.x.x.x:54005 [200]: POST /process.php
[Jun 22 15:50:27 2020] x.x.x.x:54005 Closing
[Jun 22 15:50:32 2020] x.x.x.x:52523 Accepted
[Jun 22 15:50:32 2020] x.x.x.x:52523 [200]: POST /process.php
[Jun 22 15:50:32 2020] x.x.x.x:52523 Closing
```

Listing 5.5: data.json

```
[
{"user_id":"1","timestamp":"2020-06-22 15:50:11","position":""},
{"user_id":"1","timestamp":"2020-06-22 15:50:17","position":""},
{"user_id":"1","timestamp":"2020-06-22 15:50:22","position":""},
{"user_id":"1","timestamp":"2020-06-22 15:50:27","position":""},
{"user_id":"1","timestamp":"2020-06-22 15:50:32","position":""}
]
```

In this chapter we managed to send a signal over Wi-Fi to a server with the prototype. The signal is the form of a HTTP post request and carries data in JSON format. The server ran on a local network and processed the request successfully. The complete source code of this feature is included in appendix C. Note that a request is send every 5 seconds and the file requires a service set identifier, password and the destination of the server.

Chapter 6

Prototype

In this chapter we merge the compression alarm, orientation capture and distress signal into a single prototype. The combination of these feature result into a prototype that can support a patient with ATOS in three ways. First the prototype sets the buzzer alarm when there is no heartbeat detected within the threshold of 30 seconds. This alarm informs the patient to change position for decompression. After a compression is detected the position data of two dimensions of the upper arm orientation are displayed on the OLED. The patient can use this information to gain insight on critical orientations to avoid. When a patient does not respond to the alarm for 60 seconds a distress signal is send to the server. The server could then potentially alarm others.

There are two conditions for the prototype in order to work correctly. First to monitor the heart rate we assume that the patient places constant pressure on the PPG sensor with the index finger. Secondly to send a message to a server we assume the prototype is connected to a Wi-Fi network and the destination of the server is defined.

In section 6.1 we merge the functionality of the compression alarm with the orientation capture. In section 6.2 the functionality of the distress signal is added and in section 6.3 we reflect on the prototype as a whole.

6.1 Orientation information at compression

In this section we merge the functionality of the compression alarm with the orientation capture. We can omit the functionality that we needed to verify the correctness. Therefore the code related to the BPM calculation of chapter 3 and the constant calculation and displaying of the roll and pitch of chapter 4 will be omitted.

From the alarm compression we use the functionality for heart rate detection and for setting the alarm. From the orientation capture we use the functionality that keeps track of the acceleration and the roll and pitch calculation. For this we initialize the PPG sensor, accelerometer, display and buzzer. The code snippet for this is included in listing 6.1.

Listing 6.1: setup

```
void setup()
{
    initPPGSensor();
    initPositionSensor();
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
    pinMode(BUZZER_PIN, OUTPUT);
    // Initalize checkpoint and alarm
    reset();
}
```

The program flow can be found in listing 6.2. Each iteration we request the infrared value and we use it to detect a heartbeat as in chapter 3(3-4). When a heartbeat is detected we reset all values as the heartbeat interval needs to be reset and all alarms can be turned off(5). We also retrieve the x, y and z acceleration values each iteration and filter these as in chapter 4(7-9). In the function `alarm()` we check if the alarm must be activated. If that is the case then we also call the function `setPosition()` of chapter 4, which calculates the roll and pitch.

Listing 6.2: loop

```
 1  void loop()
 2  {
 3    irValue = ppgSensor.getIR();
 4    if (checkForBeat(irValue))
 5       reset();
 6    getAcceleration(xi, yi, zi);
 7    x = F_CONST * x + F_DEVIATION * float(xi);
 8    y = F_CONST * y + F_DEVIATION * float(yi);
 9    z = F_CONST * z + F_DEVIATION * float(zi);
10    alarm();
11    oled();
12  }
```

In listing 6.3 we include the new *set message* part of the function `oled()`. We print "no contact" when the sensor does not recognize a tissue like surface as in chapter 3(4). In any other state we print the values of the last captured roll and pitch, which are zero initially(7). This way the patient can view the data of the critical orientation on the OLED after a compression is detected.

Listing 6.3: set message

```
if (irValue < 50000)
    output = "no contact";
else
    output = String(roll) + " " + String(pitch);
display.println(output);
```

6.2 Distress signal at compression

In this section we merge the functionality of the previous section with the distress signal of chapter 5. When a patient does not respond to the alarm for 60 seconds a distress signal will be send to the server. We only send one distress signal per compression. To assure this we add a boolean `distressSignal` which represents whether the signal has been send or not. We also add a constant `SIGNAL_THRESHOLD` that represents the threshold of the amount of allowed downtime before a distress signal is send. In this thesis this threshold is 60 seconds.

We use these additions in the function `alarm()`. This function is included in listing 6.4 together with the updated function `sendSignal()`. We send the distress signal as soon as the threshold is met(10-11). The boolean `distressSignal` is set to true in the function `sendSignal()` when the signal is send successfully(22). Therefore the signal can only be send once per detected compression. We add the roll and pitch to the string `data` to include the position data in the message(20).

Listing 6.4: alarm and signal

```
 1 void alarm()
 2 {
 3   downTime = (millis() - checkpoint) / 1000.0;
 4   if (downTime > ALARM_THRESHOLD)
 5   {
 6     if (!alarmOn)
 7       setPosition();
 8     alarmOn = true;
 9     tone(BUZZER_PIN, 1000);
10     if (downTime > SIGNAL_THRESHOLD && !distressSignal)
11       sendSignal();
12   }
13 }
14 void sendSignal()
15 {
16   if ((WiFi.status() == WL_CONNECTED))
17   {
18     http.begin(client, "http://" SERVER_IP);
19     http.addHeader("Content-Type", "application/json");
20     String data = "{\"user_id\":\"1\",\"timestamp\":\"\",\"
         position\":\"" + String(roll) + "-" + String(pitch) + "
         \"}";
21     int httpCode = http.POST(data);
22     distressSignal = httpCode == HTTP_CODE_OK;
23   }
24 }
```

The complete source code of the prototype can be found in appendix D. Note that the file requires a service set identifier, password and the destination of the server.

6.3 Verification

We verified the correctness of the features in sections 3.3, 4.2 and 5.3. Therefore we do not repeat the verification process for the combined features in this section. Instead we describe our remarks on the limitations of the prototype.

First of all we like to note that, although it is not a goal of this thesis, in general the prototype is not user friendly for a patient with ATOS. As can be seen in figure 2.6 the assembled prototype is uncomfortable to carry around due to the wires, the powerbank and all the shields.

Secondly the PPG sensor works as expected when the index finger places constant pressure, but with the prototype this is hard to do while moving. An elastic can be used to keep the prototype in place, but we can only assure correct heartbeat detection when the user is inactive.

Another remark on the PPG sensor has to do with the threshold of 50000 we chose for the infrared value to detect a tissue like surface in chapter 3. With this method the prototype is able to detect whether a finger is on the sensor or not. However, the use of the value 50000 can give false positives. This is due to the fact that the PPG sensor can receive values greater than 50000 on other surfaces as well. If the sensor is placed on such a surface then the program can not detect a heartbeat, which results in the activation of the alarm. This can only occur when the patient is not actively using the device. Therefore in order to prevent this, the user must either deactivate the prototype or make sure the sensor is not placed close to a surface.

One of the limitations of the prototype is that it only captures 2 out of 3 dimensions of the upper arm rotation. Solely the information about the roll and pitch will not help the patient with identifying the upper arm positions. The improve this the yaw must be calculate, preferably with a sensor fusion algorithm using an accelerometer, gyroscope and magnetometer. In this thesis we limited ourselves to the roll and pitch due to time limitation.

Lastly, the use of Wi-Fi has great possibilities, but also important limitations. The most important limitation is that the prototype needs to be connected to a network at all times to be able to send a distress signal. This means that the patient can not leave the network. If the prototype is started without an available network then the program will not even leave the setup. At this point the prototype is limited to connect with only one network. Although this can be extended manually, the device would need an interface for flexibility.

Chapter 7

Related Work

In this research we developed a prototype that detects a heartbeat, activates an alarm, captures two orientation dimensions and sends a distress signal. On its own none of these feature are remarkable, but combined they result in a product that solves a specific problem. Others have used similar technologies to solve different problems. There are different ways to relate to other work. As we did not develop a new technology, but rather combined existing ones, we will compare how others have used similar technologies and implemented similar features. In this chapter we will discuss some of these products and compare them with our prototype.

The first set of products we encountered that uses PPG sensors to monitor the heart rate are the watches from the Apple Watch Series. The Apple WatchOS 6 is the operating system designed to run on the Apple Watch Series 1 and later. The system checks the users heart rate with a PPG sensor and sends an alert if it notices anything abnormal.[17] The sensor can be seen on the inside of the watch in figure 7.1. A user can also set its own beats per minute upper and lower bounds for an alert.

The Apple Watch has similar features as the prototype: detect a heartbeat and activate an alarm. The main difference is that the watch uses a green LED to measure the heart rate instead of an infrared LED. This is due to the fact that green light does not travel as deep into tissue as infrared light. Therefore green light is more suitable for a reflective PPG sensor on the wrist.

Figure 7.1: Apple Watch

[17]Apple Support. 2020. *Heart health notifications on your Apple Watch.* https://support.apple.com/en-us/HT208931

Another device that monitors the heart rate is the Pulse Companion heart rate monitor from Epilepsy Alarms UK.[18] This device is developed to detect epileptic seizures and sends a distress signal to a pager so that others can come to aid. The pulse companion has similar features as the prototype: detect a heartbeat and send a distress signal. The main difference with this device is the protocol for the distress signal. As can be seen in figure 7.2 the Pulse Companion comes with three devices: the sensor, a smartbox and the pager. The sensor is connected to the smartbox within a range of 10 meters. If a seizure is detected the smart box sends an alert to the pager with a maximum range of 450 meters. The benefit of this approach is that there is no need for an internet connection. The downside is that the user needs to carry around an extra object and the distress signal is limited to 450 meters.

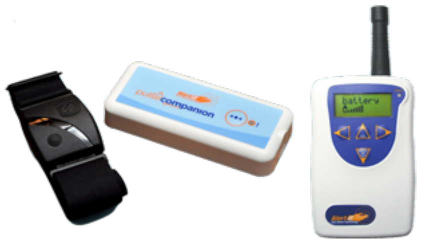


Figure 7.2: Pulse Companion

The last device we discuss is the emergency watch of 100plus.[19] The emergency watch monitors the heart rate and when it detects a problem it asks the user if it should call for help as can be seen in figure 7.3. Two notable differences with respect to the prototype are the option for a distress signal and the cell network connectivity. Asking the user if a distress signal must be send can reduce the amount of false positives, but it can also be fatal when the user is not able to request help. The use of cell network connectivity is a better option than connection by Wi-Fi, as the latter requires an access point.

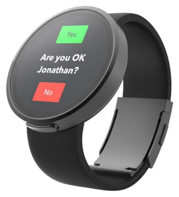


Figure 7.3: Emergency watch

[18]Epilepsy Alarms UK. 2019. *Pulse Companion.* https://www.epilepsyalarms.co.uk/product/pulse-companion/

[19]100Plus. 2019. *Emergency Watch* https://www.100plus.com/emergency-watch/

Although these products have similar functionality as the prototype, the methods vary. The usability of the prototype can be improved with a PPG sensor on the wrist with a green LED like the Apple Watch. This way the patient would not need to actively place constant pressure on the sensor. Also the cell network connectivity of the emergency watch can improve the usability as Wi-Fi connection would not be a requirement anymore.

What makes the prototype unique is the merge of different features. The combination of heart rate monitoring and orientation capture is very rare. Therefore products that capture a orientation upon compression and send this data to a web server are very hard to find.

Chapter 8

Conclusions

In this thesis we researched how information technology can support patients with ATOS who suffer from a complete compression of the subclavian artery in certain physical positions. With a microcontroller, a photoplethysmography sensor and an accelerometer we managed to develop a prototype with three features that can support a patient with ATOS.

The first feature supports a patient by minimizing compression time. With a photoplethysmography sensor we detected a heartbeat with an accuracy of approximately 4 between 30 and 90 beats per minute. We assume there is a compression of the subclavian artery when no heartbeat is detected for 30 seconds. The buzzer is then activated to alarm the patient to change positions for decompression.

The second feature captures two dimensions of the upper arm orientation. With an accelerometer we managed to capture the orientation with a maximum deviation of 5 degrees. The capture of the upper arm rotation supports the patient by giving knowledge about critical physical positions.

The final feature supports a patient in critical situations. When a patient does not respond to the compression alarm we assume the patient is in danger. In that case the prototype sends a signal over Wi-Fi to a server. The signal is the form of a HTTP post request and carries data about the patient in JSON format.

All the features we implemented can be improved and extended. The first feature can be improved by using a sensor that can be carried on the wrist using a green LED to increase usability. The second feature can be improved by using an accelerometer, gyroscope and magnetometer to calculate 3 dimensions of the upper arm orientation instead of 2. The third feature can be improved by proving an interface for network connection, so that the patient is not limited to one network.

Bibliography

[1] Fatemeh Abyarjoo, Armando Barreto, Jonathan Cofino, and Francisco R. Ortega. Implementing a sensor fusion algorithm for 3d orientation detection with inertial/magnetic sensors. In Tarek Sobh and Khaled Elleithy, editors, *Innovations and Advances in Computing, Informatics, Systems Sciences, Networking and Engineering*, pages 305–310. Springer International Publishing, Cham, 2015.

[2] Catharina Hospital. *Thoracic Outlet Syndroom*, Eindhoven, Netherlands, 2019.

[3] Łukasz Dzieciuchowicz, Wojciech Włodarczyk, and Grzegorz Oszkinis. Critical upper limb ischemia caused by initially undiagnosed thoracic outlet syndrome - case report. *Polski przeglad chirurgiczny*, 84:158–62, 2012.

[4] *Figure 2.1.* Photoplethysmography. Reprinted from Bilgaiyan A. Affiq M. Shim C. H. Ishidai H. Hattori R. Elsamnah, F. *Biosensors*, 9(3):87., 2019. Copyright 2019 by the authors.

[5] Jason H. Huang and Eric L. Zager. Thoracic outlet syndrome. *Neurosurgery*, 55(4):897–902, 2004.

[6] Mark R. Jones, Amit Prabhakar, Omar Viswanath, Ivan Urits, Jeremy B. Green, Julia B. Kendrick, Andrew J. Brunk, Matthew R. Eng, Vwaire Orhurhu, Elyse M. Cornett, and et al. Thoracic outlet syndrome: A comprehensive review of pathophysiology, diagnosis, and treatment. *Pain and therapy*, 8(1):5–18, 2019.

[7] Sofoklis Mitsos, Davide Patrini, Sara Velo, Achilleas Antonopoulos, Martin Hayward, Robert S George, David Lawrence, and Nikolaos Panagiotopoulos. Arterial thoracic outlet syndrome treated successfully with totally endoscopic first rib resection. *Case reports in pulmonology*, 2017.

[8] Sebastian Povlsen and Bo Povlsen. Diagnosing thoracic outlet syndrome: Current approaches and future directions. *Diagnostics (Basel, Switzerland)*, 8(1):21., 2018.

[9] Asa J. Wilbourn. Thoracic outlet syndromes. *Neurologic Clinics*, 17(3):477–497, 1999.

Appendix A

Compression alarm

Listing A.1: Compression detection

```
#include "MAX30105.h"              // PPG sensor
#include "heartRate.h"             // PBA algortihm
#include <Adafruit_SSD1306.h>      // OLED 128x48
#include <Adafruit_GFX.h>          // OLED chars

// PPG sensor
MAX30105 ppgSensor;
long irValue;

// OLED
Adafruit_SSD1306 display(128, 48, &Wire, -1);
String output;

// buzzer
const int BUZZER_PIN = D5;

// beats per minute
const byte RATE_SIZE = 8;          // number of bpm for average
byte rates[RATE_SIZE];             // array of last 8 bpm
byte rateSpot = 0;                 // index
float beatsPerMinute;
int beatAvg;

// alarm
bool alarmOn;
long checkpoint = 0;                // timestamp of last beat
long downTime;                      // on tissue without heartbeat
const int ALARM_THRESHOLD = 30;     // amount of allowed downTime

void setup()
{
  initPPGSensor();

  // Initialize OLED display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  // Initialize buzzer
  pinMode(BUZZER_PIN, OUTPUT);
}

void loop()
{
  irValue = ppgSensor.getIR();
  if (checkForBeat(irValue))
    bpm();
  alarm();
  oled();
}

// Extracted from Sparkfun's MAX30105 heartRate example
void initPPGSensor()
{
  ppgSensor.begin(Wire, I2C_SPEED_FAST);   // 400KHz speed
  ppgSensor.setup();
  ppgSensor.setPulseAmplitudeRed(0x0A);    // turn red LED to low
  ppgSensor.setPulseAmplitudeGreen(0);     // turn off green LED
}

// Extracted from Sparkfun's MAX30105 heartRate example
void bpm()
{
  // save checkpoint
  long wave = millis() - checkpoint;
  reset();
  beatsPerMinute = 60 / (wave / 1000.0);

  // filter erroneous values
  if (beatsPerMinute < 255 && beatsPerMinute > 30)
  {
    // store bpm in array
    rates[rateSpot++] = (byte)beatsPerMinute;
    rateSpot %= RATE_SIZE;

    // calculate average
    beatAvg = 0;
    for (byte x = 0; x < RATE_SIZE; x++)
      beatAvg += rates[x];
    beatAvg /= RATE_SIZE;
  }
}

void reset()
{
  checkpoint = millis();
  noTone(BUZZER_PIN);
  alarmOn = false;
}
```

```
void alarm()
{
  downTime = (millis() - checkpoint) / 1000.0;
    if (downTime > ALARM_THRESHOLD)
    {
      alarmOn = true;
      tone(BUZZER_PIN, 1000);
    }
}

void oled()
{
  // set display
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(32, 0);

  // set message
  if (irValue < 50000)              // state 0
  {
    reset();
    output = "no contact";
  }
  else if (alarmOn)                 // state 3
    output = "no pulse";
  else                              // state 1 or 2
    output = String(beatAvg);
  display.println(output);

  // display message
  display.display();
}
```

Appendix B

Position capture

Listing B.1: Position capture

```
#include <Adafruit_SSD1306.h>        // OLED 128x48
#include <Adafruit_GFX.h>            // OLED chars

// position sensor
#define MPU_ADDR 0x68
#define ACC_ADDR 0x3B
const float CONVERT = 180.0 / M_PI; // radians to degree
const float F_CONST = 0.99;
const float F_DEVIATION  = 0.01;
float x, y, z;
int16_t xi, yi, zi;
int roll, pitch;
int counter = 0;

// OLED
Adafruit_SSD1306 display(128, 48, &Wire, -1);
String output;

void setup()
{
  initPositionSensor();

  // Initialize OLED display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
}

void loop()
{
  getAcceleration(xi, yi, zi);
  x = F_CONST * x + F_DEVIATION * float(xi);
  y = F_CONST * y + F_DEVIATION * float(yi);
  z = F_CONST * z + F_DEVIATION * float(zi);
  counter++;
  if (counter == 100)
  {
    counter = 0;
    setPosition();
    oled();
  }
  delay(100);
}

void initPositionSensor()
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission();
}

void getAcceleration(int16_t &ax, int16_t &ay, int16_t &az)
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(ACC_ADDR);
  Wire.endTransmission();
  Wire.requestFrom(MPU_ADDR, 6);
  ax = Wire.read() << 8 | Wire.read(); // 0x3B & 0x3C
  ay = Wire.read() << 8 | Wire.read(); // 0x3D & 0x3E
  az = Wire.read() << 8 | Wire.read(); // 0x3F & 0x40
}

void setPosition()
{
  roll = int(atan2(y, z) * CONVERT);
  pitch = int(atan2(x, sqrt(y * y + z * z)) * CONVERT);
}

void oled()
{
  // set display
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(32, 0);

  // set message
  output = String(roll) + " " + String(pitch);
  display.println(output);

  // display message
  display.display();
}
```

Appendix C

Distress signal

Listing C.1: Client

```
#include <ESP8266WiFi.h>        // Wi-Fi
#include <ESP8266HTTPClient.h>  // HTTP

// Wi-Fi
#define SERVER_IP "x.x.x.x:xxxx/process.php"
#define SSID "ssid"
#define PSK "password"

// distress signal
WiFiClient client;
HTTPClient http;

void setup()
{
  // Initialize Wi-Fi
  initWifi();
}

void loop()
{
  sendSignal();
  delay(5000);
}

void initWifi()
{
  WiFi.begin(SSID, PSK);
  while (WiFi.status() != WL_CONNECTED)
    delay(500);
}
```

```
void sendSignal()
{
  if ((WiFi.status() == WL_CONNECTED))
  {
    // Send data with http post request
    http.begin(client, "http://" SERVER_IP);
    http.addHeader("Content-Type", "application/json");
    String data = "{\"user_id\":\"1\",\"timestamp\":\"\",\"
        position\":\"\"}";
    int httpCode = http.POST(data);
  }
}
```

Listing C.2: Server: process.php

```
<?php
    $post = json_decode(file_get_contents('php://input'), true);
    $data = json_decode(file_get_contents("data.json"));
    $post['timestamp'] = date("Y-m-d H:i:s");
    array_push($data, $post);
    $update = json_encode($data);
    file_put_contents('data.json', $update);
?>
```

Listing C.3: Server: data.json

```
[
    {
        "user_id":"1",
        "timestamp":"2017-04-04 19:28:23",
        "position":"65-21"
    }
]
```

Appendix D

Prototype

Listing D.1: Prototype

```
#include "MAX30105.h"           // PPG sensor
#include "heartRate.h"          // PBA algortihm
#include <Adafruit_SSD1306.h>   // OLED 128x48
#include <Adafruit_GFX.h>       // OLED chars
#include <ESP8266WiFi.h>        // Wi-Fi
#include <ESP8266HTTPClient.h>  // HTTP

// PPG sensor
MAX30105 ppgSensor;
long irValue;

// OLED display
Adafruit_SSD1306 display(128, 48, &Wire, -1);
String output;

// buzzer
const int BUZZER_PIN = D5;

// alarm
bool alarmOn;
long checkpoint = 0;            // timestamp of last beat
long downTime;                  // on tissue without heartbeat
const int ALARM_THRESHOLD = 30; // amount of allowed downTime

// position sensor
#define MPU_ADDR 0x68
#define ACC_ADDR 0x3B
const float CONVERT = 180.0 / M_PI; // radians to degree
const float F_CONST = 0.99;
const float F_DEVIATION  = 0.01;
float x, y, z;
int16_t xi, yi, zi;
int roll, pitch;
```

```
// Wi-Fi
#define SERVER_IP "x.x.x.x:xxxx/process.php"
#define SSID "ssid"
#define PSK "password"

// distress signal
WiFiClient client;
HTTPClient http;
bool distressSignal;
const int SIGNAL_THRESHOLD = 60;

void setup()
{
  // Initialize PPG sensor
  initPPGSensor();

  // Initialize position sensor
  initPositionSensor();

  // Initialize Wi-Fi
  initWifi();

  // Initialize OLED display
  display.begin(SSD1306_SWITCHCAPVCC, 0x3C);

  // Initialize buzzer
  pinMode(BUZZER_PIN, OUTPUT);

  // Initalize values
  reset();
}

void loop()
{
  irValue = ppgSensor.getIR();
  if (checkForBeat(irValue))
     reset();

  getAcceleration(xi, yi, zi);
  x = F_CONST * x + F_DEVIATION * float(xi);
  y = F_CONST * y + F_DEVIATION * float(yi);
  z = F_CONST * z + F_DEVIATION * float(zi);

  alarm();
  oled();
}
```

```
// Extracted from Sparkfun's MAX30105 heartRate example
void initPPGSensor()
{
  ppgSensor.begin(Wire, I2C_SPEED_FAST);  // 400KHz speed
  ppgSensor.setup();
  ppgSensor.setPulseAmplitudeRed(0x0A);   // turn red LED to low
  ppgSensor.setPulseAmplitudeGreen(0);    // turn off green LED
}

void initPositionSensor()
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(0x6B);
  Wire.write(0);
  Wire.endTransmission();
}

void initWifi()
{
  WiFi.begin(SSID, PSK);
  while (WiFi.status() != WL_CONNECTED)
    delay(500);
}

void getAcceleration(int16_t &ax, int16_t &ay, int16_t &az)
{
  Wire.beginTransmission(MPU_ADDR);
  Wire.write(ACC_ADDR);
  Wire.endTransmission();
  Wire.requestFrom(MPU_ADDR, 6);
  ax = Wire.read() << 8 | Wire.read(); // 0x3B & 0x3C
  ay = Wire.read() << 8 | Wire.read(); // 0x3D & 0x3E
  az = Wire.read() << 8 | Wire.read(); // 0x3F & 0x40
}

void alarm()
{
  downTime = (millis() - checkpoint) / 1000.0;
  if (downTime > ALARM_THRESHOLD)
  {
    if (!alarmOn)
      setPosition();
    alarmOn = true;
    tone(BUZZER_PIN, 1000);
    if (downTime > SIGNAL_THRESHOLD && !distressSignal)
      sendSignal();
  }
}
```

```
void reset()
{
  checkpoint = millis();
  noTone(BUZZER_PIN);
  alarmOn = false;
  distressSignal = false;
}

void sendSignal()
{
  if ((WiFi.status() == WL_CONNECTED))
  {
    http.begin(client, "http://" SERVER_IP);
    http.addHeader("Content-Type", "application/json");
    String data = "{\"user_id\":\"1\",\"timestamp\":\"\",\"
        position\":\"" + String(roll) + "-" + String(pitch) + "
        \"}";
    int httpCode = http.POST(data);
    distressSignal = httpCode == HTTP_CODE_OK;
  }
}

void oled()
{
  // set display
  display.clearDisplay();
  display.setTextColor(WHITE);
  display.setTextSize(1);
  display.setCursor(32, 0);

  // set message
  if (irValue < 50000)
  {
    reset();
    output = "no contact";
  }
  else
    output = String(roll) + " " + String(pitch);
  display.println(output);

  // display message
  display.display();
}
```