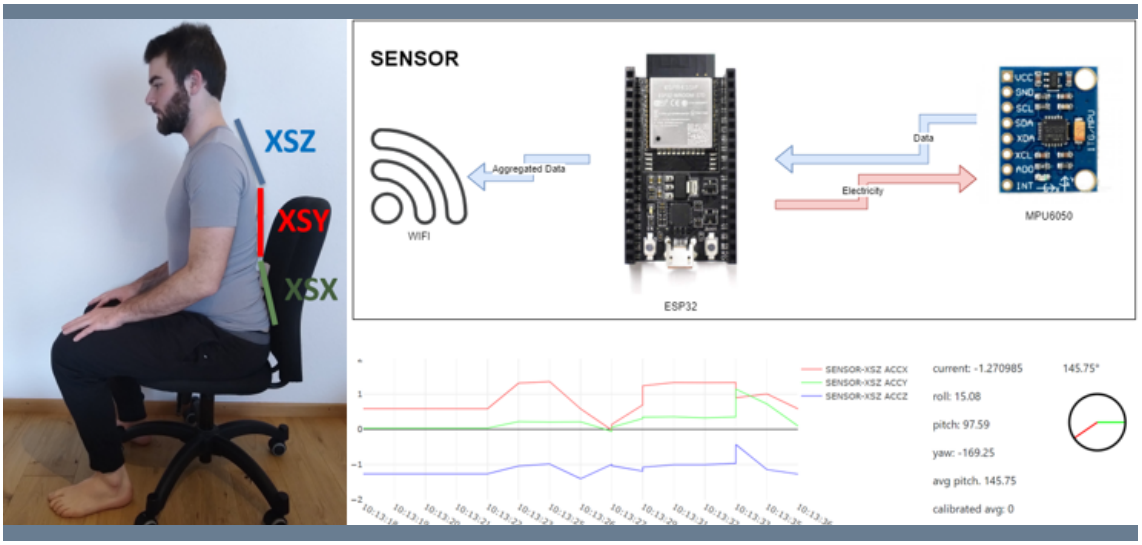

Position Yourself

A wearable IoT solution

Creating a prototype with modern IoT comoponents and topical knowhow to track and improve posture.

Authors:	Gabriel Frischknecht
Tutor:	Prof. Dr. Reto Koenig
Expert:	Prof. Dr. Torsten Braun
Date:	15.01.2020

Abstract

Back pain is a very common issue and can, in many cases, be traced back to a bad posture. Technical solutions to avoid bad posture already do exist. However, none of them seemed to be affordable, simple and open source. This is what has been inspiring my journey: trying to provide a simple and available solution, user friendly for a very common issue.

To track and improve posture, I went with simple and affordable sensors and actors which are widely available. To analyse posture it was quickly obvious, that a gyroscope or/and an accelerometer are needed to determine how a user is positioned. During testing and exchanging with physiotherapists, it became clear, that with the current implemented algorithms and my limited know-how, one sensor alone was not enough for providing clear data. But, with a minimum of two sensors the curvature of the back can be detected with sufficient certainty. To transfer the data of these sensors I chose MQTT. It simplifies data communication and also reduces the specialized hardware necessary for this project, which further improves affordability and recreatability. With a simple web app the user can set personal targets for postures or postures to be avoided, which the web app visually, and the sensors tactically, enforce.

The data gathered and the acquired experience of technical implementations are a great starting point for further development to portray individual healthy posture in the digital image. However, much more data and interdisciplinary work, with physiotherapists and other medical experts, will be needed to achieve the quality needed for such a venture.

Versions

Version	Date	Status	Remarks
1.0	14.09.2019	Draft	First version for B
2.0	24.10.2019	Draft	added technical basics
3.0	13.12.2019	Draft	Draft with second attempt
4.0	03.01.2020	Draft	Near final draft
4.1	03.01.2020	Final	Revised final version

Contents

Acknowledgements

First and foremost, I would like to thank my tutor, Prof. Dr. Reto Koenig for his guidance and support during my work. I further thank Prof. Dr. Torsten Braun for supervising my thesis in the role of the expert.

Finally, I would like to thank my partner, MSc cand. BSc Valerie Zumbrunnen, who supported me during this thesis, not only emotionally but also with her in depth know-how and experience as a physiotherapist she contributed greatly to the outcome of this thesis.

Introduction

Through my partner, who is a physiotherapist, I get some insights into the medicinal field and medicinal devices which are used. Some of these devices are equipped with simple sensors and still cost a fortune. This shocked me but also sparked my interest. Products which she encountered are applied to improve gait and posture. Since Back pain is a very common issue ("reported lifetime prevalence ranges widely from 49%-70%" [**?**]) and can, in many cases, be traced back to a bad posture while seated [**?**]. I had the idea of creating an affordable device, which improves posture while sitting. I felt like this would offer a great benefit for others and also something myself could use.

Technical solutions to avoid bad posture already do exist. However, none of them seemed to be affordable, simple and open source. These products can be found in the section "Similar Products". Some of them are quite expensive (up to 10'000 CHF) others are obfuscated and closed source and some have a great concept, however are in a different area of application. Therefore I decided to try and create a simple and cheap device which can measure and improve posture. Additionally, it should all be open source and recreatable. Besides posture, the goal is to also offer a way to improve balance, which might one day be applied by physiotherapists. With the idea of improving balance also comes a much larger goal which might not even be achievable. The solutions should be able to be applied in the medicinal field, however this seams to be nearly impossible due to the complexity and restrictions in this field.

Before beginning the technical implementation it is very important to investigate what already exists on the market.

1 Similar Devices

Usually someone already had a similar idea, or something which tries to solve the same problems has already been produced. Therefore it is important to investigate similar devices and products. A few examples of such devices are described below.

1.1 Sming

The first similar product I would like to name is the "SMING". The Sming is a sensor node created with the BFH. It is using the Sensor "TXW51" which has an acceleromter and a gyroscope. This devices uses Bluetooth smart to communicate. It was created for a master thesis and is a great example of how small such a device can be.

It was planned to use the device for sports measurement and was designed to very energy efficient. However according to my know-how the device never has been in use productively. [**?**] Such a device and the know-how gathered for this projects might be a great starting point when designing a "all in one" sensor packet for my use case. "SMING" might offer a lot even when used as is. This will need to be evaluated in the near future.

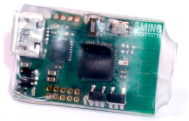

Figure 1: Sming [**?**]

1.2 Swaystar

The following existing product was the trigger for my project idea. The so called Swaystar **??** offers almost the same functionality, however it is at least 10 times bigger and costs about 10'000 CHF. I have never used such a device, however I have talked to people who worked with one. Apparently it tracks the movement / positioning of users and stores the data. An additional headband can be purchased with which vibrations are sent to the user so he can improve his body ergonomics and balance.

Figure 2: Swaystar

As said, it offers a very similar functionality, however not it covers the same use cases. Due to the size and price it can almost exclusively only be used in therapies and by medical professionals. My device intends to target the end user, and offer an easier and more affordable solution. The swaystar also is closed-source and appears to be a rather old product. [**?**]

1.3 Go Posture trainer

Further investigations into other products, were almost fruitless. There are some apps which offer almost the same thing I have tried to achieve with my app, but a bit more refined and there is one product which offers a use-case that could be considered equal to the first defined use-case (improved sitting posture), at least on first glance (**??** Go Trainer).

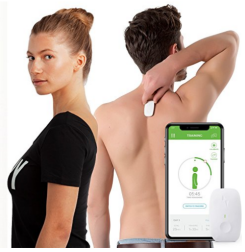

Figure 3: Go Posture trainer

The product functions as a posture trainer, it has to be attached to the back, and it is, even if small, quite visible under a shirt. Secondly the device only offers a mono-directional biofeedback. [**?**] The costs about 80 CHF which I would consider affordable.

Furthermore, the device is closed source and is only targeted to users who want to improve their posture. As mentioned the plan for my device would be to keep everything open-source and cover a wider range of use-cases.

Other contraptions to improve posture use cloth to physically "pull" the user in the right position. This is an interesting concept and usually is quite cheap. Since it is a completely different approach, I would not consider these contraptions to be directly comparable.

After these investigations and also further investigations into the medical field I considered my device to still be an innovation, offering something which is not yet available on the market.

The first attempts

I thought that the interesting part of this project, is the simplicity and low-technicality. However the implementation turned out to be much more complex than expected. This attempt unfortunately was not accompanied with the expected success.

1 The Android App

The very first approach to the posture "problem" was an android App. The standard android phone offers all necessary actors and sensors and is therefore a great test environment:

1. Accelerometer - Detects acceleration
2. Gyroscope - Detect orientation and angular velocity [**?**]
3. Vibrators
4. Storage

This means it was possible to get live data and calculate position and pitch in real time. Thanks to the built in vibration units, it was also possible to vibrate when a threshold was reached. With enough Android development know-how, this implementation is achievable swiftly and did work as expected.

The usability was not great and it did not feel very professional. The phone had to be held and placed near the chest or back to work. This would mean the first use case, improving the sitting position, was not achievable without holding or attaching the phone to the user and not using your phone while at work. Therefore the android app idea was quickly put aside. Since, for now at least, it does not offer what is desired.

2 The IoT Solution

The next plan was to create a simple IoT solution (Internet of Things).Which I thought could be achieved by combining actors, sensors and a bit of code. However it was not as easy as expected, since the sensors and positional data is rather complex.

The code is running on a microcontroller, an ESP 32, specifically an ESP32 WEMOS Lolin V1.0.0. The very specific version is quite important. I will go further in depth into why, in the **difficulties**.

The ESP32 was chosen, since it offers everything you can imagine. It has a built in Bluetooth module, Wifi module and supports nearly all modern Android libraries and components. On top of that the ESP32 is cheap. It can be bought from Swiss sellers for about 10-15 CHF. It can also be imported via aliexpress.com for as low as 2-5 CH. The delivery time however, can take up to 2 months. Therefore, I considered it to be a perfect contestant for this project.

Additionally, to the microcontroller a few more items are needed to fullfill the needed requirements. A Gyroscope, an Accelerometer, Vibrators, A Real Time Clock (RTC) and a SD-Card (Secure Digital) module. The RTC is needed since we want to log the measured values, and to analyse or visualise these logs, a time-stamp is necessary.

All modules I have used, are used in so called breakout boards. These make the testing and development easier by offering pin-out and -in to enable swift testing. Without breakout boards the parts would be a lot smaller, however, not suitable for a first prototype since it would complicate the development process.

2.1 The Gyroscope and Accelerometer

After investigating, which modules are available on the market, I have decided to use the MPU6050 Chip since it is widely used and is documented very well.

Figure 4: MPU6050

The MPU6050 communicates via the **Wire library** and has a built in thermometer, accelerometer and gyroscope. The module is also really tiny even if it is a breakout board. Without the breakout board this sensor would be reduced to the black square since in the picture (**??**).

The MPU6050 is documented very precisely which was a great help during development and made it quite easy to get the correct values. This Chip would also offer the possibility to perform some calculations directly on the Chip. Unfortunately, this part of the MPU6050 is very obfuscated. It would increase the complexity vastly without offering real benefits. This however, might be something, which will be considered in further development steps of this project.

The MPU has a very high complexity and it was quite a challenge to understand all the data, that the device provides. Especially reading the data showed to be much harder than expected. However, after some research and

a lot of testing, it turned out to be much easier than firstly expected, since I was able to read the gravity which affects the accelerometer. Currently only the values of the accelerometer are used to determine the position and sense the direction in which the user is leaning. This might be a bit to simple for the ergonomic use case, however, it is a good starting point and no technical modification are needed to get additional values.

The code for the calculation and reading of values is quite complex and really long. Therefore I will not put any code snippets in this sections since they would not provide the desired benefits to the reader. However, the whole source code will be available in my git repository. [**?**] [**?**]

2.2 The Wire library

To communicate with the MPU6050 and the RTC, SD Card module the wire library is necessary. With this library the communication and data exchange over only two cables (SDA Serial Data Line, SCL Serial Clock Lin) is made possible. Both modules communicate through these cables. These are located at port 21 and portvis 22 on the ESP32. The communication protocol is quite simple, and as soon as more than one device is communicating, it reduces the complexity of communication, and also soldering. Here a quick code example of the communication with the MPU6050:

```
void readBytes(uint8_t address, uint8_t subAddress, uint8_t count, uint8_t * dest)
{
        Wire.beginTransmission(address);
        Wire.write(subAddress);
        Wire.endTransmission(false);
        uint8_t i = 0;
        Wire.requestFrom(address, count);
        // Read bytes from slave register address
        while (Wire.available()) {
        dest[i++] = Wire.read();
    }         // Put read results in the Rx buffer
}
```

[**?**]

The communication is very well documented [**?**] and worked on the first try. However, a difficulty definitely is and will also be, the addresses. The address of these parts is given by their manufacturer and is the same for every part.

2.3 The Vibrators

Vibrators are needed to signal to the user where he should correct his position. These are, technically speaking, very basic, there is a ground and a power input, that's it. In the code, the communication therefore is also not complicated. I have created an enum with which, the port of the vibrators is defined:

```
enum VIBROPIN {
LEFT = 16,
RIGHT = 4,
FORWARD = 2,
BACKWARD = 17
};
```

Then the vibration can be triggered by simply using the keyword:

```
digitalWrite(RIGHT, HIGH);
```

Even though the technical implementation was manageable, the sensors are physically very fragile and definitely a week point of the build. They are very cheap and also replaceable, however, soldering them every time is quite time consuming. Therefore a work around will need to be found. Maybe these vibrators might be exchanged by led's during the development process, since that would be easier to debug and more stable. Additionally, it would provide a nice way to demonstrate the functionality.

2.4 SD Card

The SD Card reader and RTC module are in the same module which makes it a bit of a special case. This might be an issue in the future since it appears to be discontinued. However, it simplified the connecting of the two modules. Therefore I will still look at them individually.

To access the SD card module, the module "SD.h" was needed. It simplifies any file access drastically. The initialising is as follows:

```
if(!SD.begin()){
        Serial.println("Card Mount Failed");
        return;
}
uint8_t cardType = SD.cardType();
if(cardType == CARD_NONE){
    Serial.println("No SD card attached");
    return;
}

Serial.print("SD Card Type: ");
if(cardType == CARD_MMC){
    Serial.println("MMC");
} else if(cardType == CARD_SD){
    Serial.println("SDSC");
} else if(cardType == CARD_SDHC){
    Serial.println("SDHC");
} else {
    Serial.println("UNKNOWN");
}

uint64_t cardSize = SD.cardSize() / (1024 * 1024);
Serial.printf("SD Card Size: %lluMB\n", cardSize);
```

[?]

To append data to a file a helper function is used which interacts a File object which is defined in "FS.h". It enables a way to interact with different kinds of files and even create, delete or move folders and files.

```
void appendFile(fs::FS &fs, const char * path, const char * message){
    File file = fs.open(path, FILE_APPEND);
    if(!file){
        Serial.println("Failed to open file for appending");
        return;
    }
    if(!file.print(message)){
        Serial.println("Append failed");
    }
    file.close();
}
```

[?] this function is used in the main loop to append the recorded data by the MPU to the file:

```
result = **all accelometerdata** + **current time**
Serial.println(result);
result.toCharArray(charBuf, 150);
appendFile(SD, "/data.txt", charBuf);
appendFile(SD, "/data.txt", "\n");
```

2.5 Real Time Clock

The communication with the RTC also happens via the "Wire.h" library. Furthermore, the Timelib library is needed which defines the structure tmElements_t. This simplifies the reading of the RTC but would not be necessary. To get the time, such a tmElements_t is passed which gets filled:

```
bool read(tmElements_t &tm)
{
  uint8_t sec;
  Wire.beginTransmission(DS1307_CTRL_ID);
  Wire.write((uint8_t)0x00);
  if (Wire.endTransmission() != 0) {
    exists = false;
    return false;
  }
  exists = true;
```

```
  // request the 7 data fields   (secs, min, hr, dow, date, mth, yr)
  Wire.requestFrom(DS1307_CTRL_ID, tmNbrFields);
  if (Wire.available() < tmNbrFields) return false;
  sec = Wire.read();
  tm.Second = bcd2dec(sec & 0x7f);
  tm.Minute = bcd2dec(Wire.read() );
  tm.Hour =   bcd2dec(Wire.read() & 0x3f);  // mask assumes 24hr clock
  tm.Wday = bcd2dec(Wire.read() );
  tm.Day = bcd2dec(Wire.read() );
  tm.Month = bcd2dec(Wire.read() );
  tm.Year = y2kYearToTm((bcd2dec(Wire.read())));

  if (sec & 0x80) return false; // clock is halted
  return true;
}
```

[**?**]

This function is used to get the current time. There is also a function which sets the time, which is necessary in a first run the have the correct time. This time is needed to create a useful log of the MPU6050 data.

2.6 Difficulties

During the technical implementation a few difficulties arose. The biggest one, which nearly killed the whole project, was the wire library. Especially the fix addresses. As luck, or the manufacturers, decided, both the SD Card module and the MPU6050 have the exact same adores **0x68**. This meant, that communicating while both devices were connected could not work, or at least not reliable. At the moment of realisation, many ideas came to my head, most of them are hacky and ugly. I started researching, googling and investigating. I was sure I could not have been the first one with such an issue. After about an hour of freaking out, I have found a nice solution. The MPU6050 offers the possibility of changing the address from 0x68 to 0x69. This can be achieved by simply connecting the AD0 port of the MPU to a "High" or 5V input.

Another bigger issue was the specific hardware I used, without knowing it. My Development board was, as mentioned ESP32 WEMOS Lolin V1.0.0. I used this board more by accident, since I did not investigate into the differences of ESP32 micro-controllers. This was only realised, when I ordered additional parts to create a "final" proof of concept (POC). The special properties of the Lolin v1.0.0 is that it provides a 5v output, which is needed by the SD card module. A standard ESP32 microcontroller only provides a 3.3v output. There would be way to route the power source separately from the ESP32, however, this was not a feasible change in the last four weeks of the project. Luckily a swiss reseller, hat the right parts which were delivery within three to four days and saved the day.

3 Results

After these first attempts I created a simple proof of concept, which is able to read the MPU Data and interpret it to a degree, where I can tell in what direction the user is leaning. So to say, if the broom has fallen or not. This is achieved by simply reading out the accelerometer data and determining the exact position. furthermore, at the beginning of each power-on phase, a baseline reading is done to see the difference in orientation. This baseline reading currently contains of the first second of data. Then a threshold is defined, which is currently defined as follows:

```
if(baseZacc - (az * 1000) < -150){
      Serial.println("LEANING BACKWARD");
     digitalWrite(BACKWARD, HIGH);
   }else if(baseZacc - az * 1000 > 150){
     Serial.println("LEANING FORWARD");
     digitalWrite(FORWARD, HIGH);
   }
   if(baseYacc - ay * 1000 > 100){
      Serial.println("LEANING RIGHT");
    digitalWrite(RIGHT, HIGH);

   }else if(baseYacc - ay * 1000 < -100){

     Serial.println("LEANING LEFT");
     digitalWrite(LEFT, HIGH);
    }
```

The baseZacc, baseYacc and baseXacc are the values which was are read in the first second.

Furthermore, there are four vibrators which vibrate in the direction the "broom is falling", so front, back, left or right. Additionally, the data is also recorded into an SD Card with a time-stamp. All these individual parts are soldered together and fitted into a belt which can be worn, which is displayed in the picture on the right.

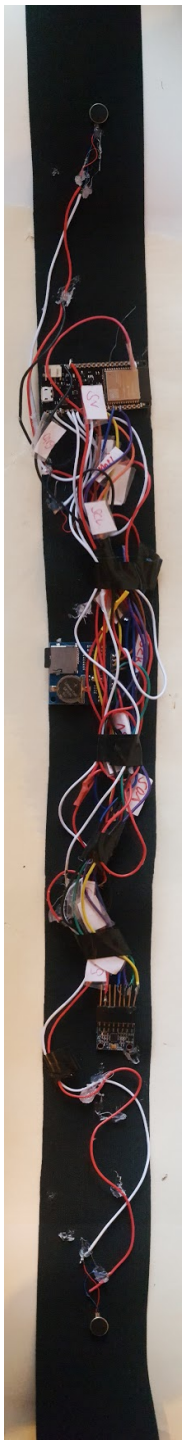

Figure 5: Belt

The power-source still is outside of the belt, in the form of a lithium polymer battery which is simply connected via a micro usb cable.

The data collected has will need to be analysed in-depth. Furthermore, it appears that one single sensor does not offer the needed results or precision to detect posture. The current posture model lacks a lot of information and does not have the quality to give useful information.

4 Learnings

After this first attempts I quickly realised, that connecting all the sensors via cable limited me drastically due to the restrictions of the Wire.h library and the I2C protocol. Due to the addressing issues with the current configuration, some sort of multiplexer, or custom created accelerometers and gyroscopes is needed. Custom parts would currently be way to complicated and multiplexers add unnecessary complexity to the project, I have been looking for another solution. The solution should enable me to scale up the number sensors without back-draws.

Furthermore, the current cable management and hardware complexity will need to be simplified. The current build is much higher than desired and will need to be simplified. Especially since it should be reproductionable.

Version 2

After my first attempts and some discussion with my supervisor I realised that the system needs to be distributed. This means that the sensors send their data wirelessly. This enables me to add additional sensors without having issues with the Wire library or physical wires. However, this increases the cost and complexity of each sensor slightly, since each sensor needs some sort of "brain", which handles the connection and communication. Which means each sensor needs to be equipped with a microcontroller, for example an ESP32. These are needed to establish and manage the wireless connection. However, it reduces the complexity of the construction and the entire solution drastically. The logic will be distributed to many smaller services rather than an entire monolith of logic.

1 Attempt A - Local Client Server Network

The communication was achieved via WIFI since I know the protocol and can get it running swiftly. Also the ESP32 I have been using has WIFI natively enabled. Connecting the devices using bluetooth or bluetooth low energy (BLE) might help reduce electricity consumption. This will need testing in the future and will not be further investigated in this thesis.

An issue with wireless communication is the synchronisation of the data flow. To simplify the process and due to lack of know how and time to achieve a correct synchronisation I will currently not take it into account, while knowing the data might be time shifted. This might be an issue when visualising and analysing the data, but will be tackled then.

Furthermore, for the communication via WIFI a simple "Client Server" protocol was used which has already been implement in ESP32 Micro-controllers. This protocol, as far as I understood, communicated on the transport layer using Transmission Control Protocol (TCP). [**?**] [**?**]

Server Client Request:

```
String clientRequest(String input)
{
  Serial.println(input);
  String response = "\0";
  for (int i = 0; i < NUM_CLIENTS; i++)
  {
    WiFiClient client = server.available();
    client.setTimeout(50);
```

```
    if (client) {
      if (client.connected()) {
        client.println(input);
        data = client.readStringUntil('\r'); // received the server's answer
        Serial.println(data);
        if (data != "\0")
        {
          int Index = data.indexOf(':');

          CLIENT = data.substring(0, Index);
          ACTION = data.substring(Index + 1);
          Serial.println(data);

          if (CLIENT == "ACK")
          {
            response = ACTION;
          }

          //client.flush();
          //data = "\0";
        }
      }else{
        Serial.println("client not connected");
      }
    }
    else{
        Serial.println("client null or false");
      }
  }
```

[?]

Client Loop:

```
void loop () {
  if (!client.connect(server, 80)) {
    while (WiFi.status() != WL_CONNECTED) {
      Serial.print(".");
      delay(500);
    }
```

```
    Serial.print("+");
    delay(100);
    return;
  }
  data = client.readStringUntil('\r');  // received the server's answer
  Serial.println(data);
  if (data != "\0")
  {
    int Index = data.indexOf(':');

    CLIENT = data.substring(0, Index);
    ACTION = data.substring(Index + 1);

    if (CLIENT == CLIENT_NAME)
    {
        client.println("ACK:" + getData());

    }else{
        client.println("\0");
    }

    client.flush();
    data = "\0";
  }
}
```

[?]

The data is sent 10 times per second for evaluation purposes. However this will be reduced and aggregated. Since the position of the user is not as time sensitive, that a millisecond response time is necessary. This enables us to simplify the data transfer from the devices, which is a benefit only achievable by the current configuration of "Smart-sensors".

The data is then saved onto a SD Card. This enables me to analyse the data after measuring it and is simple to implement. This concept works, however, it has some major flaws.

2 Flaws of Attempt A

Since I wanted to build an all in one system, I tried to save all the data locally. This was e mental barrier, set by the first attempt. In which everything had to be fit into a single belt. Where everything was collected and then

sent from a single node to another device, with Bluetooth or a local WiFi hotspot. This "closed system" and single node idea hat to thrown out before achieving a real improvement.

Furthermore, during the development and testing of this concept further flaws arose. The client server network was, in its current implementation, not very stable. Additionally, the implementation required much more hardware than I would have liked. The Server needed a few modules for it to work, as shown below:

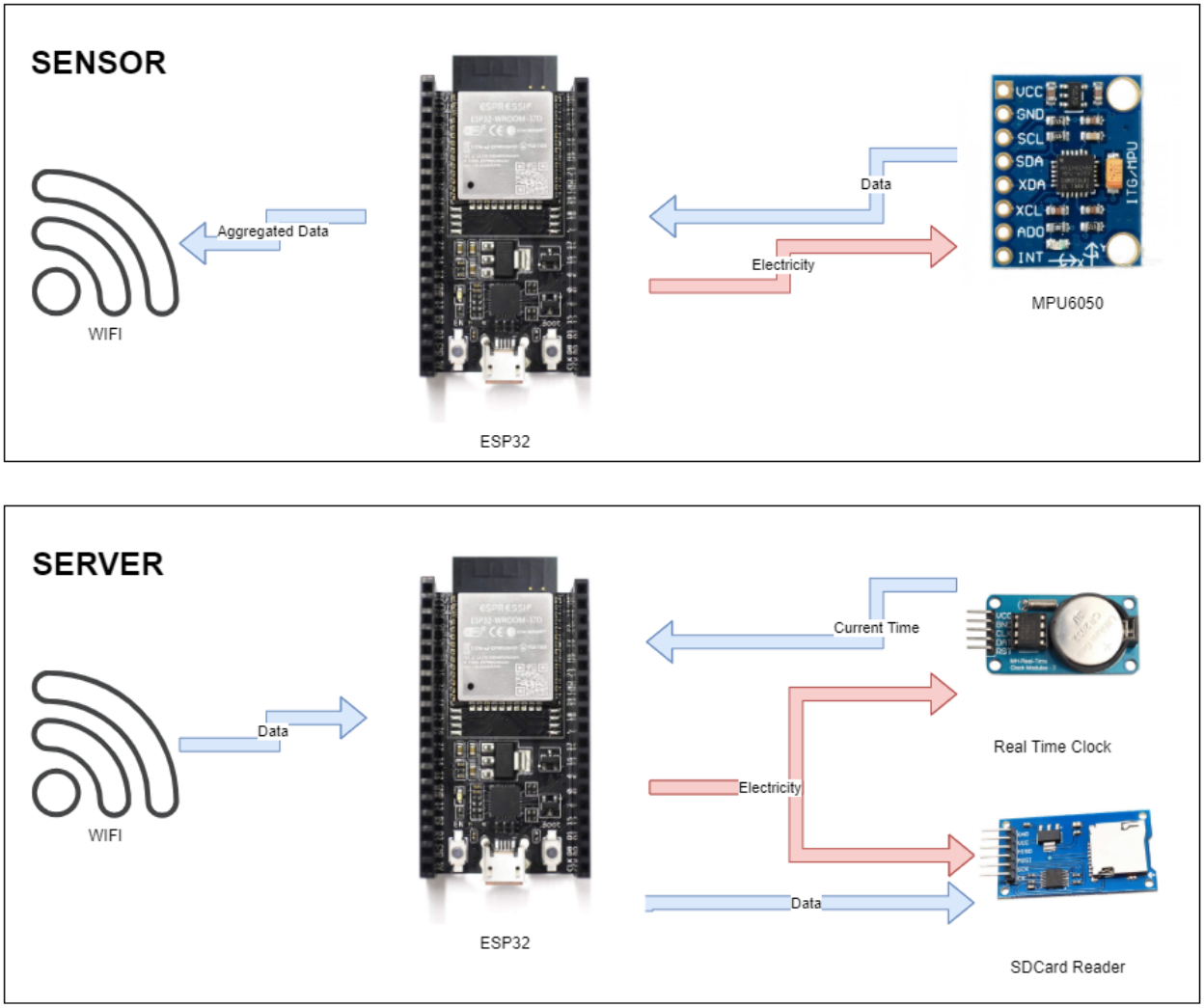

Figure 6: Client Server Network

These elements, the RTC and the SD Card reader, may look simple. However, tehy add an additional point of failure, are still quite complex to work with and need to be configured additionally. Additionally, many different kinds of RTC and SD card reader exists which do not have the exact same communication protocol or library. Which is a big flaw, when trying to create something open-source and useful. Since it restricts users and would require multiple different implementations. All this felt like quite the hassle and I soon realised that the communication needed to be changed.

Beside these technical issues the data transfer during the development phase, the transfer and visualisation of the

data was quite awkward. The device needed to be stopped and the data needed to be transferred manually from the SD-Card. This made the analysis very hard and static.

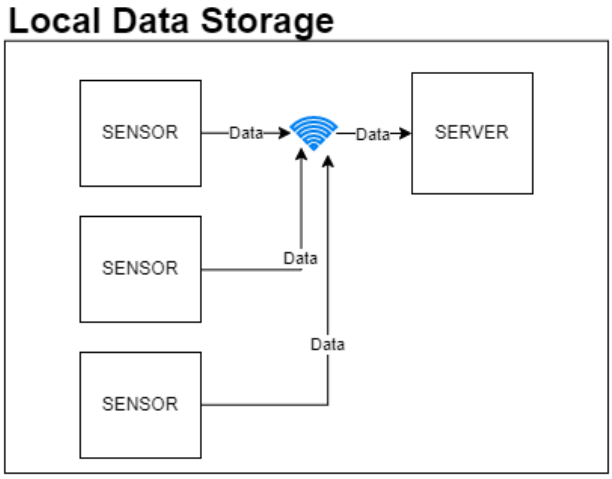

Figure 7: Communication Diagramm Local

After working with it for a while I tried to find a much simpler way. An implementation closer to the goal of an easy and affordable sensor module. To improve the client server implementation, I first thought of using a simple http request. Since The messages would have been sent on the Application layer and not the network layer (see Figure **??**) the HTTP protocol would have had much more overhead. However, in theory, this also would mean simpler and more stable communication. This however did not feel right. Running a HTTP server on a microcontroller did not offer any apparent benefits over using a "real" server. Furthermore, HTTP has way to much overhead for such a scenario. I realised that the solution, was something I already was quite familiar with. A simple MQTT broker.

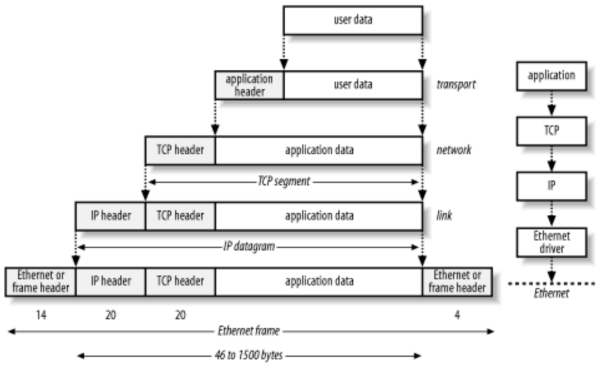

Figure 8: Network Packet [**?**]

3 Attempt B - Communication via MQTT

After this first attempt I understood the received data and how to handle the message from the sensors. The communication in my new attempt was handled with a MQTT broker (MQ Telemetry Transport), which can be

setup for free within seconds. For my endeavours I used cloudmqtt.com which is completely free and easy to apply cloud service. The MQTT broker could also run locally. However, this approach was not further investigated or tested. A local setup would be simple and but would currently not offer any real benefit.

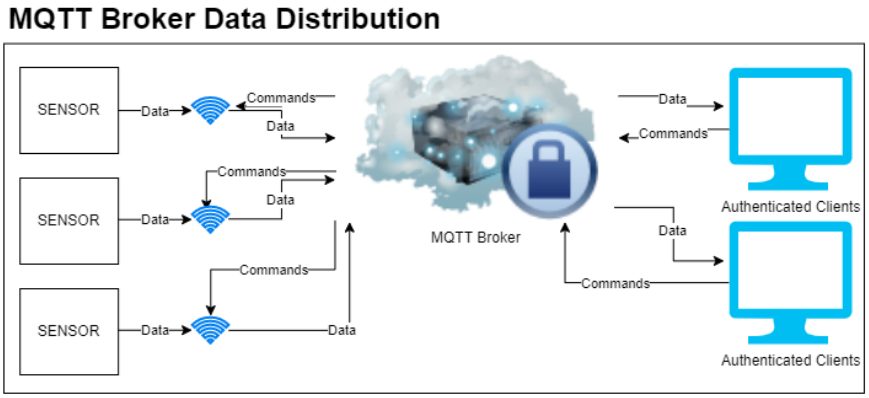

Figure 9: Communication Diagramm MQTT

MQTT uses a simple publish subscribe protocol which I have already implemented a few times. Below you see the whole setup and loop code, which is almost readable thanks to helper functions:

```
void setup() {
  Serial.begin(115200);
  Wire.begin();
  USERID = getRegisterdUserid();
  setupMPU();
  setupWifi();
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(callback);

  strcpy(fullsubtopic, TOPIC);
  strcat(fullsubtopic, USERID);
  strcat(fullsubtopic, SUB);

  strcpy(fullpubtopic, TOPIC);
  strcat(fullpubtopic, USERID);
  //strcat(fullpubtopic, PUB);

  while (!client.connected()) {
   reconnect();
  }
  startMillis = millis();
```

```
  getData();
}

void loop () {
  client.loop();
  currentMillis = millis();
  //get the number of milliseconds since the program started
  if (currentMillis - startMillis >= period)
  //test whether the period has elapsed
  {
    Serial.println("trying to send");
    char * message = getMessage();
    publish(message);
    Serial.println(message);
    startMillis = currentMillis;
    //IMPORTANT to save the start time of the current LED state.
  }
 getData();
 }
```

This loop collected the data into an average in "getData()" function

```
void getData(){
  if(readByte(MPU6050_ADDRESS, INT_STATUS) & 0x01) {
  // check if data ready interrupt
    readAccelData(accelCount);  // Read the x/y/z adc values
    getAres();
    ax = (float)accelCount[0]*aRes - accelBias[0];
    // get actual g value, this depends on scale being set
    ay = (float)accelCount[1]*aRes - accelBias[1];
    az = (float)accelCount[2]*aRes - accelBias[2];
    readGyroData(gyroCount);  // Read the x/y/z adc values
    getGres();
    gx = (float)gyroCount[0]*gRes - gyroBias[0];
    // get actual gyro value, this depends on scale being set
    gy = (float)gyroCount[1]*gRes - gyroBias[1];
    gz = (float)gyroCount[2]*gRes - gyroBias[2];
    tempCount = readTempData();  // Read the x/y/z adc values
    temperature = ((float) tempCount) / 340. + 36.53;
```

```
    // Temperature in degrees Centigrade

    accelVal[0] = ax;
    accelVal[1] = ay;
    accelVal[2] = az;
    gyroVal[0] = gx;
    gyroVal[1] = gy;
    gyroVal[2] = gz;
    setAvg();

   }
}
void setAvg(){
    for (int i = 0; i < 3; i++){
      avgAccelVal[i] = (avgAccelVal[i] + accelVal[i])/2;
      avgGyroVal[i] = (avgGyroVal[i] + gyroVal[i])/2;
  }
}
```

and created a JSON from the collected data in the "getMessage()" function which was sent every second:

```
char* getMessage(){
  char* a = "{ \"id\": \"";
  char* b = "\", \"acc\":[";
  char* c= "], \"gyro\":[";
  char* d= "]}";
  char accelbuff[64];
  char gyrobuff[64];
  Serial.println("loading data to buffers");
  char* loc = accelbuff;
   size_t tempLen;
   int i = 0;
   for(i = 0; i < DIM(avgAccelVal)-1; ++i)
    {
        snprintf(loc, 12, "%f,", avgAccelVal[i]);
        tempLen = strlen(loc);
        loc += tempLen;
    }
    snprintf(loc, 12, "%f", avgAccelVal[i]);
```

```
    //snprintf(loc, 12, "%f", avgAccelVal[i+1]);
    tempLen = strlen(loc);
    loc += tempLen;
    loc = gyrobuff;
    for(i = 0; i < DIM(avgGyroVal)-1; ++i)
     {
          snprintf(loc, 12, "%f,", avgGyroVal[i]);
          tempLen = strlen(loc);
          loc += tempLen;
     }
   snprintf(loc, 12, "%f", avgGyroVal[i]);
   //snprintf(loc, 12, "%f", avgGyroVal[i+1]);
   tempLen = strlen(loc);
   loc += tempLen;
   strcpy(messagebuffer, a );
   strcat(messagebuffer, DEVICEID);
   strcat(messagebuffer, b);
   strcat(messagebuffer, accelbuff);
   strcat(messagebuffer, c);
   strcat(messagebuffer, gyrobuff);
   strcat(messagebuffer, d);
   return messagebuffer;
}
```

The creation of the JSON message, was a very complex part of the implementation. This was due to the fact, that the "PubSubClient" Library [**?**], used to communicate with the MQTT broker, could not handle "arduino" strings. Therefore char pointers were needed, which are a quite complicated to work with. The high complexity also caused an error while concatenating the char pointers. The lines which are commented out, is where the error occurred. Data was transferred and the JSON message almost looked correct. However, since the variable "i" is incremented once more, when accessing the index of the array. The wrong address, avgAccelVal[3] instead of avgAccelVal[2] is accessed. Therefore, the last acceleremoter value, is actually the first gyroscope value. This occurred, since the two char arrays were allocated directly after each other. The last gyroscope value was pointed to a random address, therefore the value was random as well. This mistake was unfortunately only discovered when visualising the data, since C++ does not throw any errors when accessing items out of the array, and simply returns the values in the addresses accessed. Nonetheless after some tinkering I finally managed to get a correct JSON message (JavaScript Object Notation) over the MQTT broker:

```
{
    "id": "SENSOR-XSZ",
```

```
    "acc":[-0.003835,0.001486,0.056012],
    "gyro":[0.056012,0.240598,0.038814]
}
```

Protocol Buffers (Protobuf) would be a great alternative, which I will try to implement in the future. Protobuf does take some time to set up but would simplify the data transfer greatly since bytes could be directly sent and would not need to be concatenated to a JSON. However this was not attempted in this thesis.

Additionally, I enabled all devices to be calibrated remotely. This since when I put these devices on I will need to calibrate them after they are set in position. This is quite a simple task with MQTT since the clients can simply subscribe to a topic, from which they get messages:

```
void callback(char* topic, byte* message, unsigned int length) {
  Serial.print("Message arrived on topic: ");
  Serial.print(topic);
  Serial.print(". Message: ");
  String command;

  for (int i = 0; i < length; i++) {
    Serial.print((char)message[i]);
    command += (char)message[i];
  }
  if(command == "calibrate"){
    calibrateMPU6050(gyroBias, accelBias);
    // Calibrate gyro and accelerometers, load biases in bias registers
    initMPU6050();
    Serial.println("MPU6050 initialized for active data mode....");
  }
}
```

This callback gets registered when connecting to the mqtt broker.

3.1 Benefits Version 2

The benefits of managing the messages via a MQTT broker was, that I could get the messages and work with them from any device with an Internet connection. This means that any calculation heavy tasks can be done from a "real" computer which handles these much better.

This also enabled me to visualise and analyse the data in real-time which made it much easier.

Furthermore, it also achieves the goal of being cheap and affordable. Since we now only need MPU6050 and microntrollers to send the data and this can be setup much easier than with an ESP32 "server".

This also means that there does not need to be any more logic on the devices itself since everything can be done from a Computer. The vibration device which is attached to a sensor pin, can also be activated via the MQTT broker.

The Fact that these sensors now can simply send data without knowing where they are sending their data to opens quite a lot of doors. I will get further into that in the chapter "Prospects".

3.2 Backdraws Version 2

The Obvious backdraw is, that the sensors need to have a constant Internet connection. This is obviously not great, especially when we try to develop something for an every usage since we are somewhat bound to a single place.

Furthermore, if used as a product, these sensors would need to be registered first before usage, this does increase complexity for a first usage however simplifies setup greatly.

4 Identifying Posture

The biggest hurdle, was and still is identifying and correctly analysing the data. I have had several different approaches on how to interpret and visualise the data. During my first attempt (Attempt A) I tried to visualised the data in two ways, with Python and Power BI.

Firstly I transformed the data into a Comma Separated Values (CSV). CSV is a commonly used format viable for many different applications

According to the datasheet of the MPU6050 the collected data is "g-force". The accelerometer measure in mg and the gyroscope in °/s.

ZERO-G OUTPUT						
Initial Calibration Tolerance	X and Y axes		±50		mg	1
	Z axis		±80		mg	
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C		±35			
	Z axis, 0°C to +70°C		±60		mg	

Figure 10: MPU Datasheet

The entire documentation can be found online [**?**].

4.1 Visualising with Python

I tried to visualise the data using Python since it was recommended on a few forums and seamed to be feasible for this task. It actually did work quite well and I have achieved within 2 Days with Python. I have never worked with Python before and was happy that I managed to use and apply it within days.

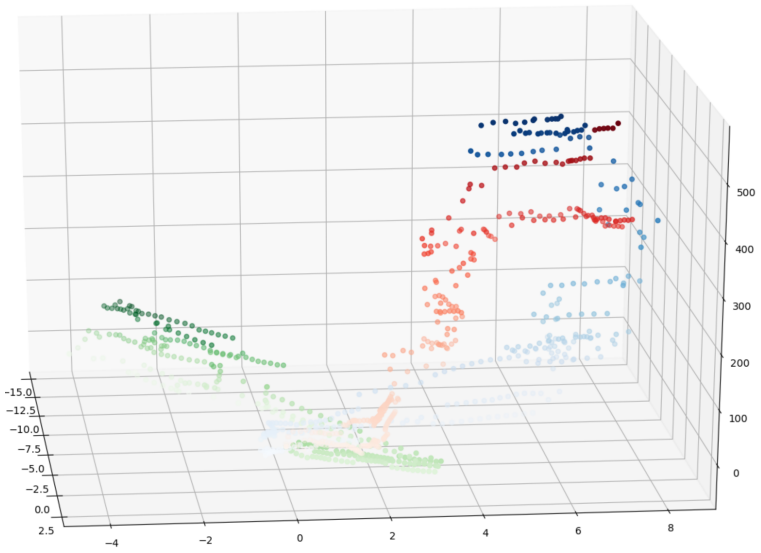

Figure 11: Python Visualisation

Visualised you see my first attempt in figure **??**, showing movement of three sensors with the data of the accelerometers. The sensors were attached to my shirt with scotch. This is not a perfect solution, which is visible in the different orientation of the green dots.

Here I tried to add up the movement from the accelerometers to see how this stacks up. The Data was not very helpful and I tried to anime the accelerometer data directly to see how it changes, this was a bit clearer but still very hard to interpret since it was static data.

To visualise I used numpy and matplotlib which are quite handy but still took some time the get used to. The data was simple display on a grapth (ax) from np arrays:

```
ax.scatter3D(XSXAcc[:,0],XSXAcc[:,1],XSXAcc[:,2], c=XSXAcc[:,2], cmap='Reds')
ax.scatter3D(XSYAcc[:,0],XSYAcc[:,1],XSYAcc[:,2], c=XSYAcc[:,2], cmap='Blues')
ax.scatter3D(XSZAcc[:,0],XSZAcc[:,1],XSZAcc[:,2], c=XSZAcc[:,2], cmap='Greens')
```

4.2 Visualising with Power BI

Power BI is a very powerful tool which I was already used to, since we used it at work to visualise project data and quality gates. Therefore, it was much easier for me to visualise the data.

During different attempts of visualising the data I realised, it made sense to visualise it with a two dimensional graph, since the three dimensional visualisation did not offer any real benefits. Furthermore, during this visualisation I also quickly saw, that the data collected had to be incorrect, since the accelerometer should never fluctuate as much as it did. The error was in my code and was quickly fixed. However, I still had issues making conclusions from this static data.

Due to the static data, I was not able to make concise conclusions of the collected data. I had some understanding, what effects large movement have on each axis, but no clear picture has been made. Therefore I decided to to create a way to visualise the data dynamically.

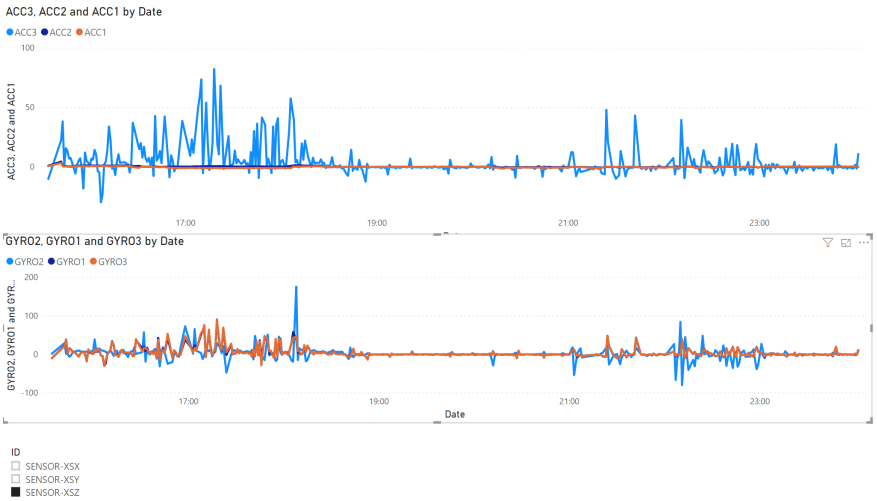

Figure 12: Power BI Visualisation

5 The Web App

The static visualisation of data has a lot of issues, especially when the data is dynamic. To get a real understanding of the collected data, it would needed to be compared to a video. It was very hard comprehend what the effect, especiall of smaller movement had, on sensor. Therefore, I decided to implement a small web app which visualises the data in real time.

This was achieved with a simple C# Model-View-Controller website which subscribes to the MQTT broker. Here the switch to MQTT and the extra effort for this switch, was highly beneficial.

The website started small and was simply used to visualise the collected data:

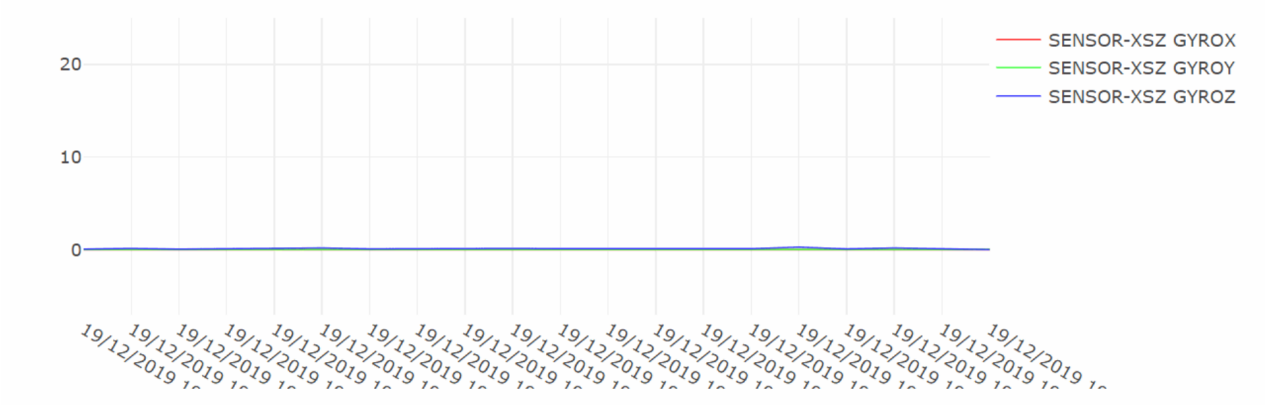

Figure 13: Simple Web Visualisation

Thanks to this visualisation I was quickly able to understand the data and add additional visualisations and make first conclusions which data was necessary for an in depth analysis. It became clear, that the data needed to be "transferred" into a much more readable format, like pitch, roll and yaw. The conversion was quite simple and needs only the acceleremoter data:

```
function getRoll(x, y, z) {
    pitch = 180 * Math.atan(x / Math.sqrt(y * y + z * z)) / Math.PI * 4;
    roll = 180 * Math.atan(y / Math.sqrt(x * x + z * z)) / Math.PI * 4;
    yaw = 180 * Math.atan(z / Math.sqrt(x * x + z * z)) / Math.PI * 4;
    return {
        "roll": roll,
        "pitch": pitch,
        "yaw": yaw
    }
}
```

[?]

With this formula the position of the sensors was readable and elementary to visualise. This function is great for the implemented sensors and the current use case. It uses only the current measured values for calculations. Other formulas for positional analysis, like Quaternions [**?**], are much more complex and use the sum of the measured data. This can cause drift, by small errors adding up. Which, especially when working with "amateur" sensors and long term measurements, can be a real problem.

When this formula is applied with the literal and live data which is sent from the sensor, still a lot of jitter is present. Due to this the data was not very readable or meaningful. To reduce this jitter, I decided decided to visualise the median of a 10 second measurement. Since the refresh rate is not very important for the current use case, this did not have any disadvantages. It became clear after some testing and discussion, that one sensor would not be enough, since its position does not offer sufficient information about the posture. (see Figure **??**)

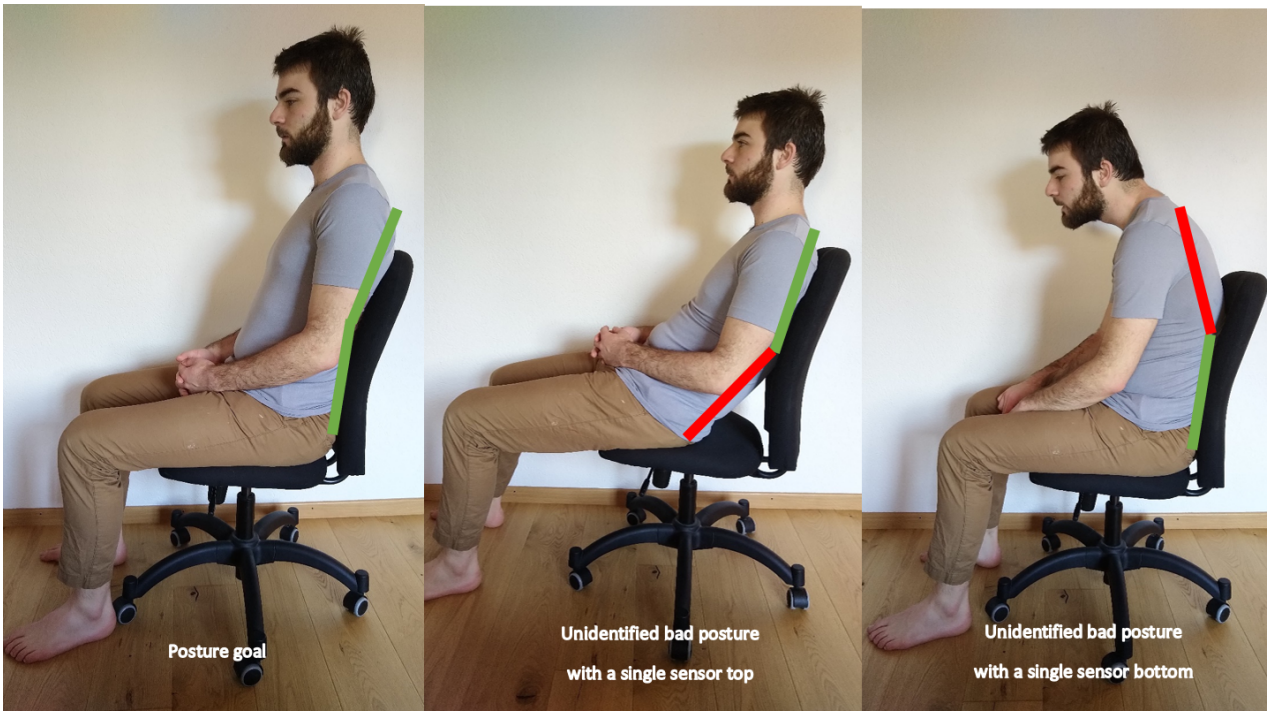

Figure 14: Back position

Therefore, a minimum of two sensor would definitely be needed to make general conclusions from the data. This due to the fact, that posture can change on the lower or the upper back, as visualised in Figure **??**). Which means it is possible that the upper back stays straight and the lower back is buckled or the other way around. Additional sensors would be optimal to get more detailed information about posture and the curvature of the spine. Also the shoulders are not tracked with the current sensor position. Therefore, it must be pointed out, that currently it is only possible to determine the positions of the lower and upper back. Not much further insight can be given with only two sensor. However, I consider this to satisfy the goals of this thesis. It offers enough insight to a casual wearer and has the potential to help improve posture. Therefore, also due to time restrictions and missing know-how, an in-depth analysis of the spine and curvature was not done in this thesis. To improve the overall understanding and visualisation of the data, three sensors were used during testing. The Idea was to calculate

the degrees between each sensor to check whether the user has a bent back. This was later overturned since the curvature of the back does not give any insight in healthy posture [**?**]. However, I might get back to that idea since it would enable the user to have more flexibility when moving. It might allow to increase the threshold, when the entire back is moved correctly.

Through the MQTT-Broker the data of three sensors is sent as JSON. In the JSON is the UID of the sensor with which it is identified. Then the data is parsed and passed through a javascript library (plot.ly) [**?**] to create the graphs. To simplify the analysis of the data I further also knew the position of each sensor:

Figure 15: Sensor position

After some research, testing and discussion with a physiotherapist, we realised, that the approach of trying to set a general goal for all users was not correct, in many ways. Different studies found that posture is a very individual thing and cannot be set globally [**?**] therefore, we decided to implement functionality so that each user can set his/her own individual goals. I will get further into these decision in the chapter "Results".

5.1 Functionality

Thanks to distributed setup and the data transfer using a MQTT broker, the web app could not only be used to visualise the data. Additional functionality was added which uses the live data to help the user improve his/her posture. This was achieved with two, lets call them, modes.

The first mode is the "Goal" mode, where the user can set a goal posture which he would like to achieve and keep. The user clicks to button "calibrate" and then the desired posture needs to be held for at least 10 seconds. Now every sensor has a calibrated optimal value. Each deviation of this optimal value is visualised, and when a threshold is reached a warning is triggered to the user. In the beginning of this project the warning was only visual. However a tactical feedback from the sensor itself was implemented later on.

These modes are visualised as below. In the circle in figure **??** you see a red line and a green line. Here the avoid mode is visualised where the red line is the position the user would like to avoid and the green line is his current position.

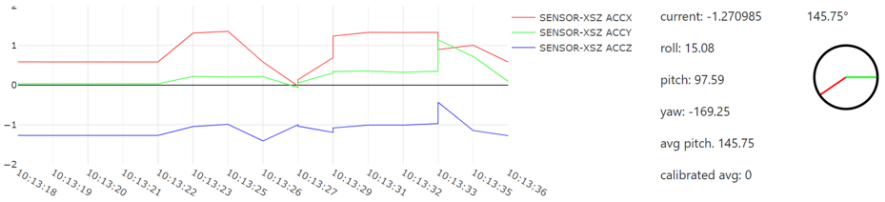

Figure 16: Target position

The second mode is very similar. However, a position the user would like to avoid is programmed and the sensor react when the user gets to close to this position rather than to far away.

For the calculation of these modes each sensor was only analysed individually. However, this could be easily improved by calculating the overall changes of position, so to accumulate each sensors deviation in degrees. This would enable the web app to trigger a warning when the user is bent to much or too little.

6 Hardware

The Implementation with MQTT not only simplifies the communication but also reduces the hardware complexity greatly. The only parts needed are an MPU6050 a vibration unit and some sort the micro-controller. Since MPU and vibration unit are suitable with almost any micro-controller there are almost no restrictions left. Additionally, even the accelerometer and gyroscope hardware could be changed. The only restriction to analyse the data, is that the data is sent in the same JSON format.

The parts were not all soldered together. The first implementation with the belt, tought me, that soldering things together is not optimal for prototyping. So I decided to create modules which can be attached to each other using cables. (Figure **??**) This meant the different parts can be reconfigured multiple times.

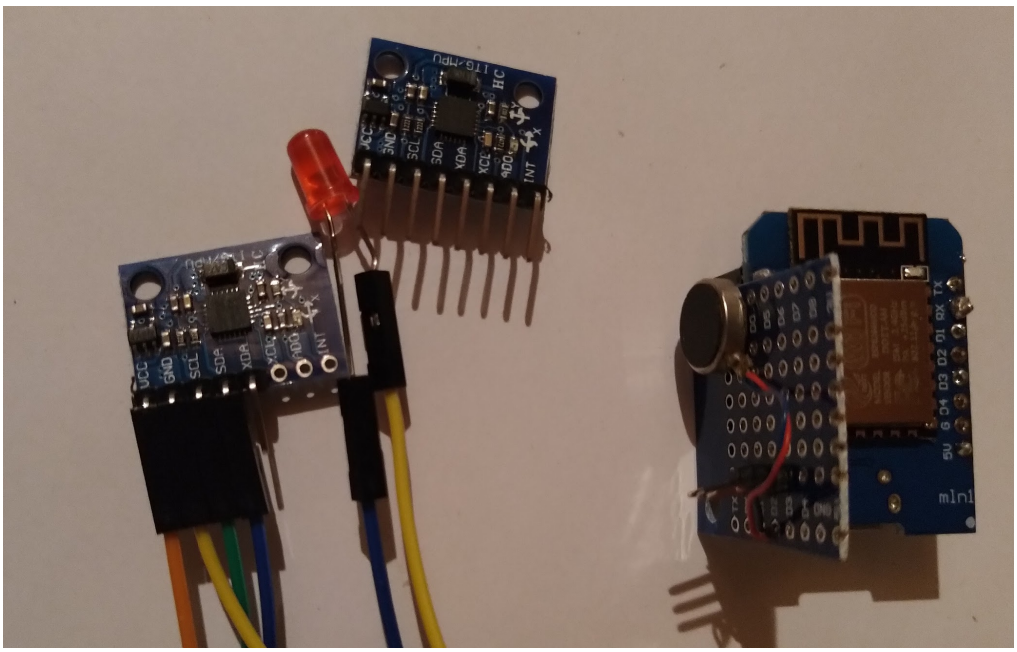

Figure 17: Sensors Attached via cable

Currently, the hardware all together costs, even when bought in Switzerland and as a single unit, about 20 CHF. When bought from china, still as single units, the price can be reduced to about 12 CHF. I would estimate that the price could be lowered further, when buying in bulk, to about 5 CHF.

Optimally a single circuit board would be created where all the necessary processors are combined. According to an electrical engineers I have asked, designing such a board would not take much time. Such boards could then be ordered and would probably reduce the cost. Such a board would also reduce the complexity of the build drastically. These could bought in bulk and would probably reduce the cost even further. Since the needed sensors and configuration only became clearer during this thesis. Therefore, no such circuit board was designed or ordered yet.

7 Technical Difficulties

Independent of the communication protocol, there are some issues, or difficulties when trying to visualise and interpret accelerometer and gyroscope data.

One big issue with movement data is, it is only usable when the measured target is not moving while sitting. For example a bus driver would not be able to get correct data, since the sensors would not only measure the movement of the driver but also the movement of the bus. To fix this another sensor would need to be attached to the bus itself to subtract the movement of the bus.

Another difficulty especially with roll, pitch and yaw, is the wrap around. The Sensor might be in a state where his pitch is 0 degrees and one degree of movement would lead to a new pitch value of 359 degrees of pitch. This is not a very difficult problem, however when not addressed leads to almost unusable data. For the visualisation and it does not matter however when trying to tell the user how many degrees he is off his target, a special calculation needed to be implemented [**?**]:

```
function GetAngleDifference(from, to)
{
        var phi = Math.abs(from - to) % 360;
        // This is either the distance or 360 - distance
        var distance = phi > 180 ? 360 - phi : phi;
        return distance;
}
```

[**?**]

The synchronisation of the data was expected to be an issue, however since the data collected is not very time sensitive, and I only react to an average collected every 10 seconds the synchronisation should not be an issue. Furthermore MQTT has a latency of "little above 120 milliseconds" [**?**] which should be fast enough for my current scenario. This might be an issue in further steps of this project. I go into this issue in the chapter "Prospects".

Results

During my thesis I learnt a lot about posture and sensors and achieved a great base for further projects and continuing this journey. A finished product was not yet created, however, this was never a goal of this thesis. Nonetheless, I achieved a view goals which I have listed below.

1 Quick Summary

As a quick summary of all goals and learnings, it can be said, that a minimum of two Sensors are needed. One must be positioned on the lower and one on the upper back (As visualised in Figure **??**). Thanks to this positioning any change of posture should be detected from at least on sensor. Additional sensors will help improving precision however, are not necessary. The desired posture is defined by the user itself and not predefined by the software. A "global" correct position could be set using this setup, however, was not found useful or recommendable. [**?**] Additionally, the user can also set posture he would like to avoid.

When the set posture goal is not met, the user gets a vibration feedback from the sensor and a visual feedback from the software. The analysis of these value is done with the median of a 10 second measurement, since a quicker refresh rate does not offer any real benefit and it reduces useless vibrations due to hectic movement.

2 Achieved Goals

In the evolution of this project, I simplified the data transfer and created a first definition how the sensors can be "dumb". Thanks to this the "sensor packets" got much simpler and need much less soldering. I was able to move all logic from the devices to any language or system desired without losing any significant response time.

With the flow displayed above, in figure **??**, the sensors can all have the same logic and only need an unique id to be identified and no further setup. It needs to be noted, that the register service was not fully implemented for this thesis, however, it was mocked. This since it does not add any value to the thesis or the goals of this thesis.

Thanks to these adjustments and improvements during this project, live data visualisation was achieved. This is on one very convenient feature, since it simplifies the analysis and understanding of the collected data vastly. It helped me really understand the data and transfer it to a simple posture model. Which then enabled me to visualise the positioning of the sensors and the posture. Additionally, thanks to my better understanding of the sensor data and collected data, how it can be used to model posture and my experience gained through the first versions, I know

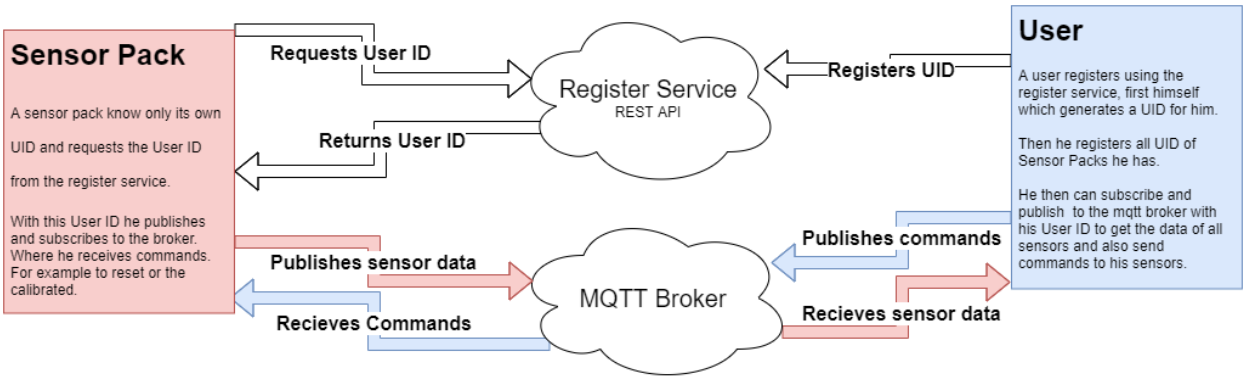

Figure 18: Sensor workflow

how a single self contained sensor pack could be implemented. This would be easier to user however, much more restricted.

Also due to the abstraction of logic, it became much easier working with the data and I think it also improved the re-usability, since logic can be implemented in any language desired and is not restricted to c++.

2.1 Improving Posture

Posture is quite a difficult topic, and I would be overselling, when writing, that I can improve posture with this concept. Since I have not many long term tests or studies I cannot confirm or deny that. however, what I can do is read about good posture and how it can be improved and try to follow these principals. This is exactly what I did. According to the paper ""Sit Up Straight": Time to Re-evaluate" [**?**] the right posture is not globally given or can be set for every person on this planet.

In the following graphics the writer of the paper summarised their findings:

Most interesting for me are the first two points, which indicate that my posture module must be agile and adjustable to each user individually. This is also why I did not set a default goal, which a user must try to achieve. Each user can set their own goals. According to my understanding, it is the only way to safely and individually improve posture.This approach still has some risks and might lead to uncertainties with users. The user might not know what a good posture is and train a wrong posture or be afraid of training a wrong posture and not using this concept. This could be tackled by contacting a professional for a first input and an initial setup of the device.

2.2 Being Open source

All the code created for this project will be released on my git repository soon. An introduction and explanation will be available as well, including a list of parts needed to build the same devices. The code and all the documentation will be available as soon, as the code has reached a standard and cleanliness I can get behind.

1. **There is no single "correct" posture.** Despite common posture beliefs, there is no strong evidence that one optimal posture exists or that avoiding "incorrect" postures will prevent back pain.
2. **Differences in postures are a fact of life.** There are natural variations in spinal curvatures, and there is no single spinal curvature strongly associated with pain. Pain should not be attributed to relatively "normal" variations.
3. **Posture reflects beliefs and mood.** Posture can offer insights into a person's emotions, thoughts, and body image. Some postures are adopted as a protective strategy and may reflect concerns regarding body vulnerability. Understanding reasons behind preferred postures can be useful.
4. **It is safe to adopt more comfortable postures.** Comfortable postures vary between individuals. Exploring different postures, including those frequently avoided, and changing habitual postures may provide symptom relief.
5. **The spine is robust and can be trusted.** The spine is a robust, adaptable structure capable of safely moving and loading in a variety of postures. Common warnings to protect the spine are not necessary and can lead to fear.
6. **Sitting is not dangerous.** Sitting down for more than 30 minutes in one position is not dangerous, nor should it always be avoided. However, moving and changing position can be helpful, and being physically active is important for your health.
7. **One size does not fit all.** Postural and movement screening does not prevent pain in the workplace. Preferred lifting styles are influenced by the naturally varying spinal curvatures, and advice to adopt a specific posture or to brace the core is not evidence based.

Acknowledgment: The authors would like to thank Kevin Wernli @KWernliPhysio for his assistance in developing the illustrations for the figure.

Figure 19: Posture Study [**?**]

The source code will be licensed under the "GNU General Public License v3.0" [**?**] [**?**]. This enables the source code to stay open source and still be used and distributed. As described here: "Permissions of this strong copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights." [**?**] Other licenses, like the MIT license, would enable users to use the code however, keep their changes hidden. Since this projects goal is to improve and stay open source, the availability of the source code is crucial. [**?**]

3 Open Goals

From my goal of creating a simple affordable solution to improve posture, a lot has been achieved. Nonetheless, I did not create the type of solution I first had in mind. I would not consider this a problem, since the current solution is much more flexible. However, my first goal, which I tried to achieve in the first version, was a single contained unit which achieves posture improvements. This would be possible with the know-how I have now and

will also be kept as a long term goal. however, I feel that much more can be achieved with the distributed network I have implemented currently.

Currently this sensor network is a simple proof of concept and I will definitely need to implemented some things I have mocked to create a usable and user friendly product. The Register service would need to be implemented and also the web app needs a lot more work. It is currently not very user friendly and only configured to work with my 3 sensors.

Furthermore, when the register service is available the sensors need more work, to enable a complete self provisioning, since currently the network connection is hard coded. This is not very interesting or hard since it has been implemented multiple times (https://www.arduino.cc/en/Reference/WiFiNINABeginProvision), therefore I left it out.

Security, of the data transfer and storage, also is an issue that was only thought of but not implemented. The MQTT broker is secured by credentials however, these are hard-coded and would currently be the same every user.

Lastly the size of the sensor packets is far from optimal and could be drastically reduced by creating a single plate with all logic withing. This will not be a goal that I will tackles soon, since this would be the last step of creating such a "product".

4 Learnings

Firstly, I understood how the MPU6050 sensor works with the wire library, learned much more about Arduino coding and the communication with other sensors and actors. In detail I learned applying and using the I2C standard and what restrictions it has. Furthermore, I have learned what to be aware of when working with micro-controllers, how to setup a simple client server network with minimal resources and what restrictions micro-controllers have. These are all necessary to create the implemented sensor module. To position the sensor pack I had to test multiple positioning and discuss with a physiotherapist what a useful position would be. According to my understanding the current optimal position would be directly on the spine, and at least two sensors, one on the upper and one on the lower back. Every additional sensor improves the precision of the posture model.

Additionally, the learning will be applicable on the first version of the project. As mentioned two sensors are necessary for a useful analysis of posture. Nonetheless, with the current know-how, I know that some measurements are still possible with a single sensor unit. Long-term testing and further analysis will be needed, to determine how much information is lost when using a single sensor. Furthermore, it must be ensured that the lost information is not critical for a daily wearer.

To understand the positioning I also had to understand what posture is. This was only possible through research and unfortunately this know how is still very limited.

5 Project Management

In the beginning of the Project a lot of goals were defined. These goals were tagged as necessary or optional. however, the goals changed greatly during the thesis, since new ideas emerged. Nonetheless the primary necessary goals, stayed the same and are defined as follows:

Goal	**Planned**	**Completed By**
Communication between ESP32 and Sensors	July 2019	July 2019
Collect the data	August 2019	August 2019
Enable multi Sensor communication (Client, Server)	September 2019	September 2019
Enable multi Sensor communication (MQTT)	-	October 2019
Visualise the data	October 2019	November 2019
Real time visualisation (WebApp)	-	December 2019
Sensor position and amount	November 2019	December 2019
Identify posture	November 2019	December 2019
Visualise posture	December 2019	December 2019
Visualise and communicate wrong posture	December 2019	January 2020

Table 1: Planned and unplanned goals

5.1 Technical implementation later than expected

Originally it was planned to have everything technical, software and hardware, done in the beginning of December. however, due to the MQTT data transfer implementation and the visualisation of the data with a web app a lot of the work got done later than expected. however, this was manageable since a time buffer was planned in the beginning. Three weeks of holidays in December were planned in the beginning of the project to ensure a certain and calm finish of the implementation. As visible in the table, this three weeks were used and were definitely necessary.

Prospects

As mentioned much more has to be achieved, even so, a great base was made. However, a lot of interdisciplinary work between computer scientists, medicinal experts and electric engineers will be needed to create a "product". By the term "product" I do not mean something that could be sold, rather a module which can be used by a non-technical user.

1 The Next Steps

A lot more work and studies are necessary to prove, that this concept offers a real benefit to posture or improves back pain. Much more testing and data is needed to see whether these sensor can be used long-term and that the hardware is durable enough. These modules were merely tested for few hours and are simple prototypes.

The learning from this thesis will enable me to try and finalise the first version with a single sensor. Surely the precision would not be ideal. It might be enough for daily usage and has the potential to help with posture on a very elementary level. As discussed in the chapter "The Web App" a lot of information is missed with a single sensor. Therefore, a lot of "field" testing must be done, to ensure a high enough certainty.

These potential of these sensors a much bigger use case than expected. With my current know-how and more time I might be able to measure more than posture. My partner is a physiotherapist and already had some great ideas where these sensors might be used. For example to measure bending of knees and other joints.

Another ambitious goal is to visualise movement with the sensors. I am not sure if this is a possibility since the drift of the measurement could be drastic, however this is something I will definitely try to implement, since if possible, would open a lot of opportunities.

Adding an additional sensor as a reference point would offer a great improvement. Since it would enable users to wear and track posture even when working in a moving environment. The calculation is probably quite complicated. Since the movement of the reference point has to be calculated in real time. This increases the complexity, since synchronisation and latency will need to be considered, which I was able to mostly ignore during this project.

Furthermore, I will also try out other accelerometers and gyroscopes to determine precision and which sensors are optimal for my use case. This was not considered, since the MPU6050 does offer enough precision for the current implementation. Furthermore comparing sensors requires a lot of know-how and time.

Declaration of primary authorship

I hereby confirm that I have written this thesis independently and without using other sources and resources than those specified in the bibliography. All text passages which were not written by me are marked as quotations and provided with the exact indication of its origin.

Place, Date: Ostermundigen, 15.01.2020

Last Name/s, First Name/s: Gabriel Frischknecht

Signature/s: .....

List of Figures

List of Tables