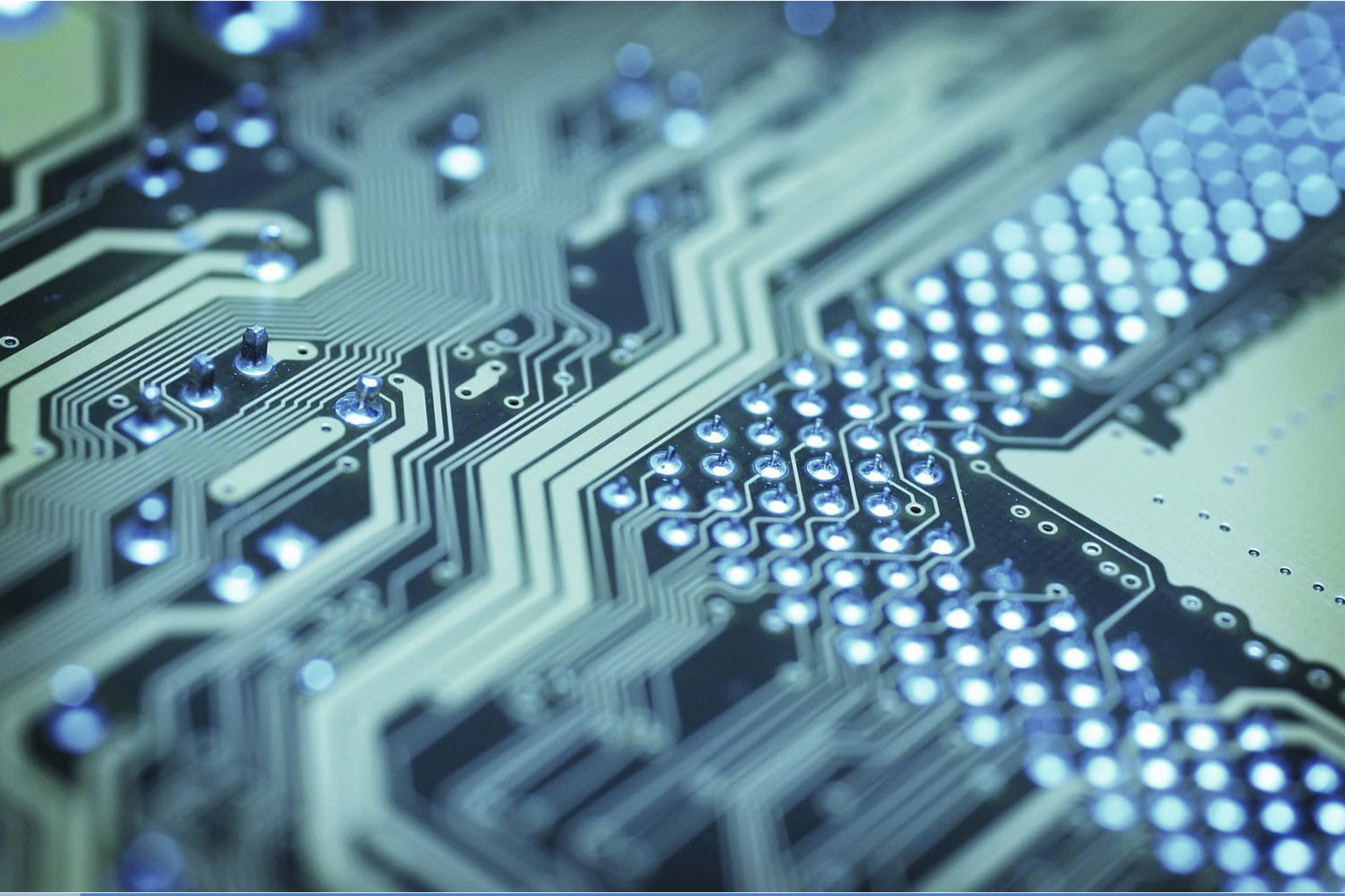


Smart Leg Physical Therapy

Using SensorTile

Project Report

DGMD E-14

Physical therapy may be underestimated due to several reasons with catastrophic consequences to patients. This project brings Intelligent Help available 24/7 for faster and friendly recovery.

By Alvaro Ramírez, Harvard Extension School, December 2021

Motivation

Erwin Flores[1], Alvaro's best friend, is a brilliant 76-year old ex-NASA physicist, phylosophy lover, amazing short-story writer, successful entrepreneur and, in his youth, a renowned Peruvian rock star that, due to a second brain surgery to extract a cancerous tumor, ended on a wheelchair with the right side of his body paralized. Even though he has an incredible intelligence and many talents, the lack of movility pushed him to a deep depression. His condition requires a lot of physical therapy but having a therapist all those hours at home seven days a week is logistically complex. This is where a better solution is needed to complete the work of the physical therapist, something like a friendly system that tells him if the exercise is done correctly or not and that do not complain if the same exercise must be repeated dozens of times, at any time of day or night.

Project Classification

This project fits into the following categories:

- Wearable Inertial Sensor Application
- Physical Activity Monitoring
- Monitoring of Physical and Physiological Changes in Daily Life
- Data Visualization and Data Analysis
- Real Time Data Visualization (future implementation)
- Gesture Recognition
- Human Activity Recognition
- Pose Estimation and Recognition
- Physical Rehabilitation
- Artificial Intelligence Application
- General Event Detection Application

The main project goal is to "improve life quality by giving back physical mobility"

[1] Erwin Flores - Wikipedia, la enciclopedia libre

Description

The project explores how wearable technology can transform the rehabilitation of patients with physical disabilities by providing feedback that could help improve the effectiveness of therapy and improve recovery time.

Physical therapy is a common treatment for individuals with various physical disabilities and injuries. It can improve muscle strength and mobility, decrease pain levels, relieve stress on joints and tissues in the body, restore independence and improve quality of life. A typical physical therapy session includes a range of activities that target the damaged or impaired muscles and joints. The physical therapist specifies a set of activities to engage in during each therapy session and coaches on the proper form or posture required.

Technology

The first approach for this project was to use three SensorTiles as shown in Figure 1:

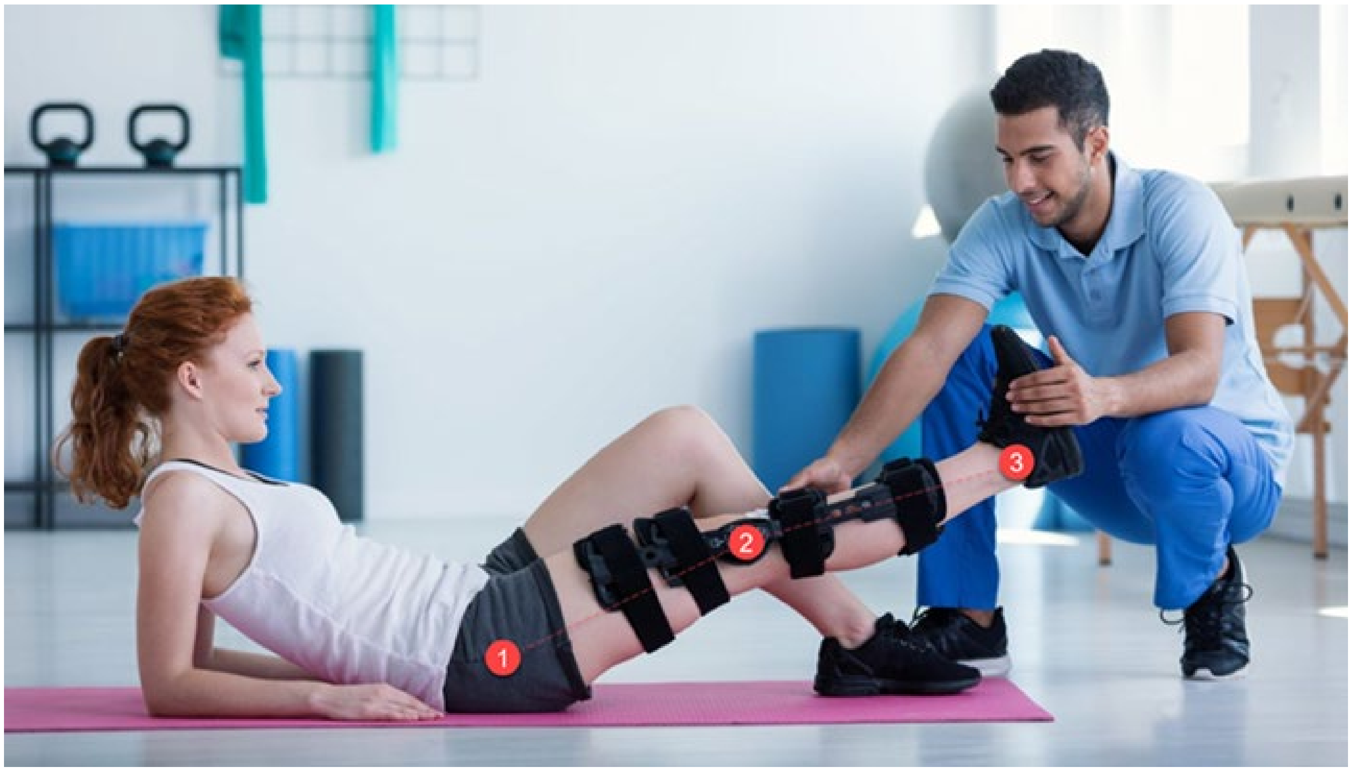


Figure 1

The purpose of having three sensors was to create a triangle with the spatial position that combined, would tell us if the leg was flexed and how many degrees. In this approach, the distances that matter are the distances among the three sensors, which play perfectly for any calculation. But this approach has several challenges, where the most critical is reading the three sensors to process the data as one unit. After a lot of research and trials with three SensorTiles as well as with three SensorTile.boxes and a few algorithmic tricks to match the samples, I got the readings in the computer ready to be processed by the neural network.

However, analyzing in more detail the purpose of the project, I found that only one sensor at the extreme of the limb, in this case the ankle, would provide enough information to determine if the leg went up or down, or if the knee was flexed or extended. It can even tell us if the leg was twisted. This finding simplified the code and the entire solution, bringing to a patient not only ease of use but also a better price. For these reasons, the final design was defined to have only one sensor in the ankle, as shown in Figure 2:

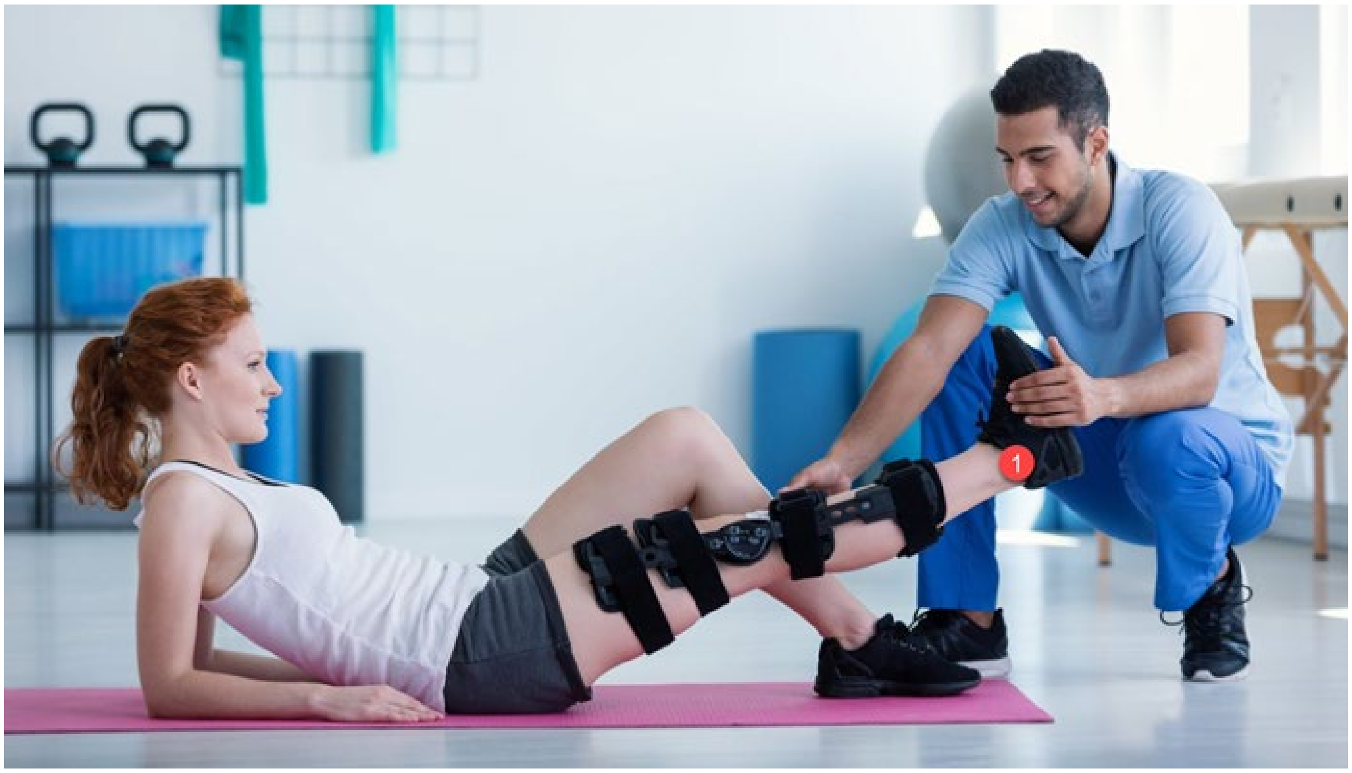


Figure 2

Use Case

The use case for this project is based on a patient that fully or partially lost the mobility of a leg. SensorTile wearable device is used to take advantage of its accelerometer, magnetometer, and gyroscope, although other features such as the microphone could be used in future versions. With these sensors, the system measures movements and changes in posture while storing data captured and providing feedback to let the patient know if the exercise is done correctly or not.

For leg exercises, the SensorTile must be located at the end of the limb to obtain the maximum change in acceleration and position.

To obtain the position of the limb, the system uses a SensorTile attached to the patient's ankle as shown in Figure 3 below:

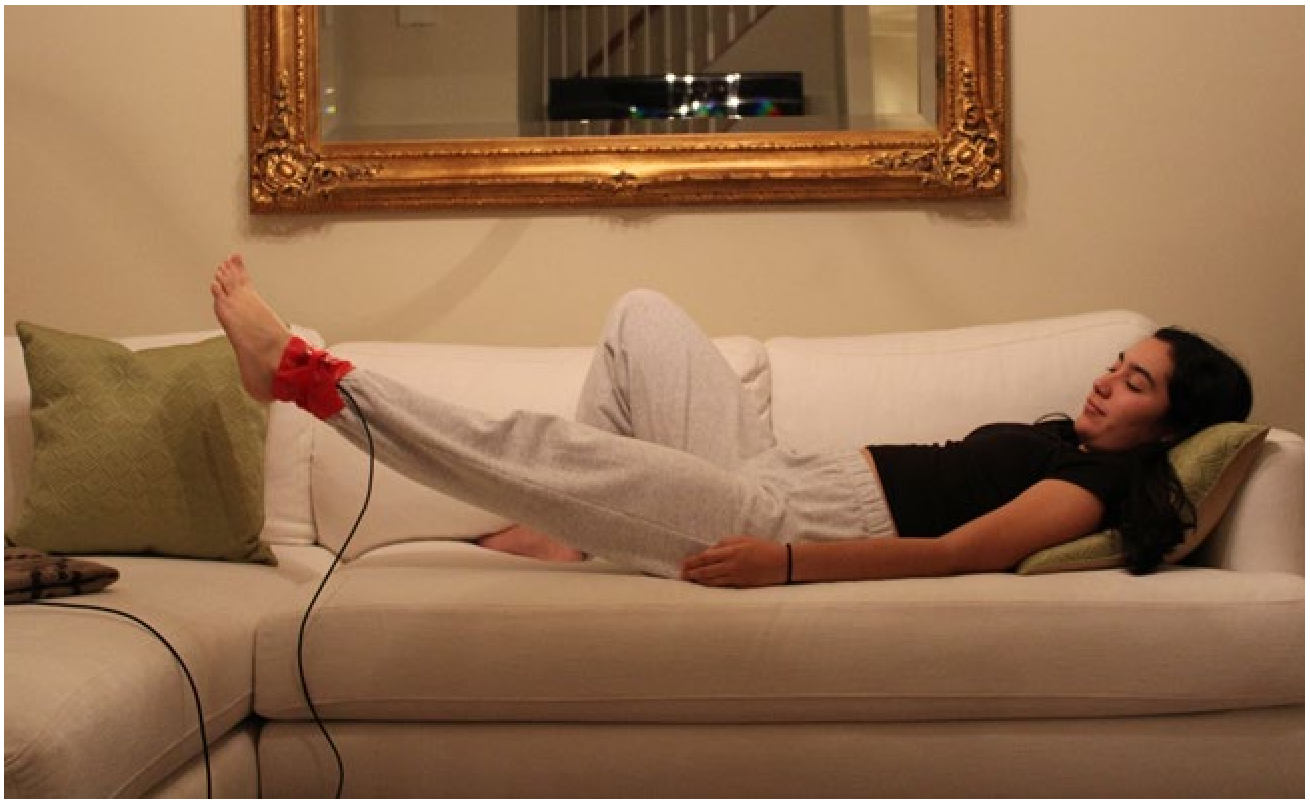


Figure 3

Objectives

The objectives of the project are mainly oriented to the physical and mental health of the patient. These are the main objectives:

1. Improve life quality by bringing back physical mobility
2. Enhance the effectiveness of physical therapy process
3. Enable physical therapy to be properly and safely performed without physical therapist presence
4. Cost effective solution

Tools Required

The equipment used for development is:

- SensorTile Development Kit STEVAL-STLKT01V1
- STM32 Nucleo-64 Development Board NUCLEO-F303RE
- USB-C 3.1 10-Port Hub with Power Adapter - 36W Powered (12V/3A)
- USB 2.0 A-Male to Micro-B Cable
- USB 2.0 A-Male to Mini-B Cable
- Laptop computer

- iPhone

The software used in this project includes:

- STM32CubeIDE
- VS Code
- Jupyter Notebook
- Bleak
- Edge Impulse
- Tableau
- Excel with VBA

Proof of Concept or Minimum Value Product

In the initial proposal, the proof of concept was defined to:

Recognize a set of physical therapy leg movement activities and provide real time feedback on proper form by placing three SensorTiles on the patient leg to track movement, as shown in Figure 1. These sensors are intended to be placed one on the ankle, one on the knee, and one on the hip, sending accelerometer and gyroscope data to a Raspberry Pi over BLE. Under this architecture, the Raspberry Pi fuses or combines the samples of the three SensorTiles (one sample of each SensorTile at a time), to classify the activity and provide visual feedback. The system processes the data using a trained classifier to identify if proper physical therapy motion was performed.

But the evolution of the project and the research performed suggested that there is a more efficient way as mentioned in section DESCRIPTION/TECHNOLOGY while pushed some other changes. Going through a huge code adjustment after reaccommodating my mindset as a result of my findings, the number of SensorTiles was reduced from three to one, magnetometer data was added and, as listed in TOOLS REQUIRED, a Windows computer replaced the Raspberry Pi to add enough computation power to the solution. Due to the lack of resources in the team, it was necessary to reduce the scope by removing the real time response feature, but I went deeper on the prediction component by coming back to my original idea of using an unsupervised neural network for anomaly detection instead of a classification model. The new proof of concept utterance is as follows:

Recognize a set of physical therapy leg movement activities and provide feedback indicating if the movement was done correctly or incorrectly. This is to be done by placing a SensorTile on the patient ankle to track movement, as shown in Figure 3. This sensor will send accelerometer, magnetometer, and gyroscope data to a Windows computer over BLE. Under this architecture, the system processes the data using an unsupervised neural network for anomaly detection to identify if proper physical therapy motion was performed.

For the proof of concept, I decided to use three knee exercises as shown below. For each exercise, I generated data for training, testing and live testing.

Exercise 1

Exercise 1 is the same shown in Figure 3, also shown here. It consists of raising the straight leg up and down.

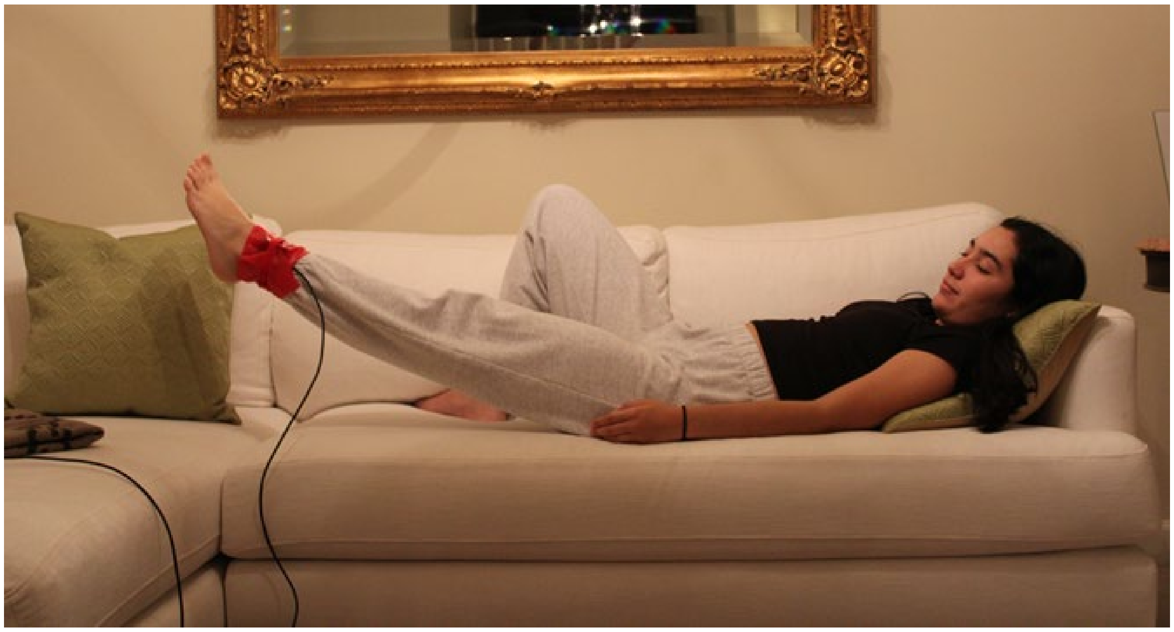


Figure 4

Exercise 2

Exercise 2 requires bending the leg making the knee work as well as other muscles of the leg, as shown in Figure 5:

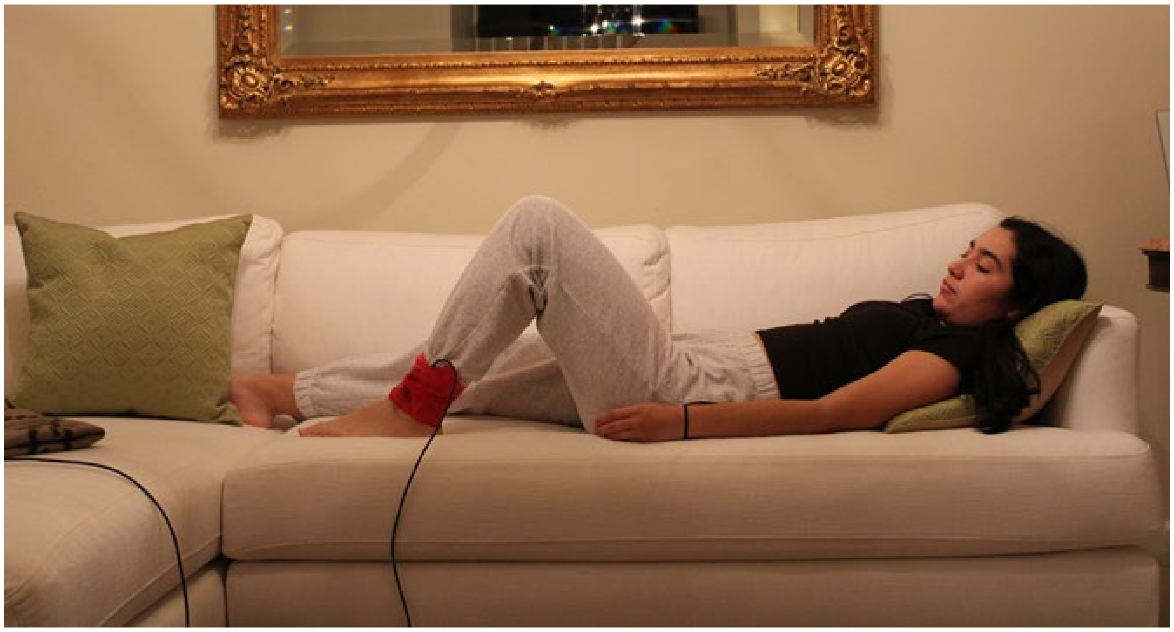


Figure 5

Exercise 3

Exercise 3 is also an exercise bending the knee, but this time the patient is facing down. In this case the leg is bended to its maximum angle where it can go freely, without pushing it:

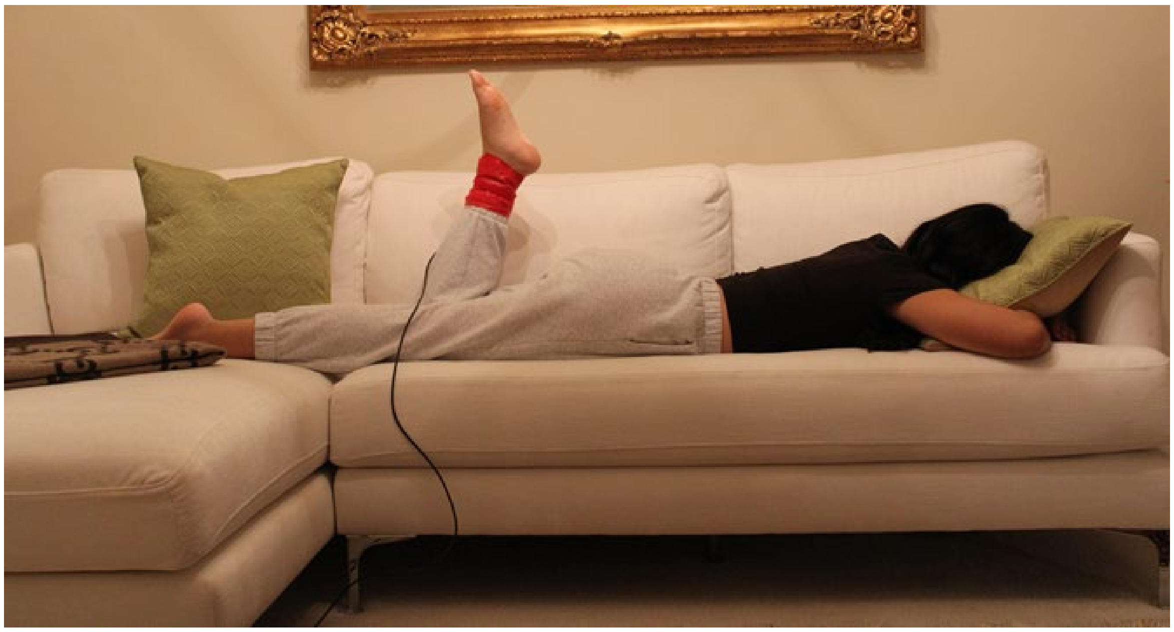


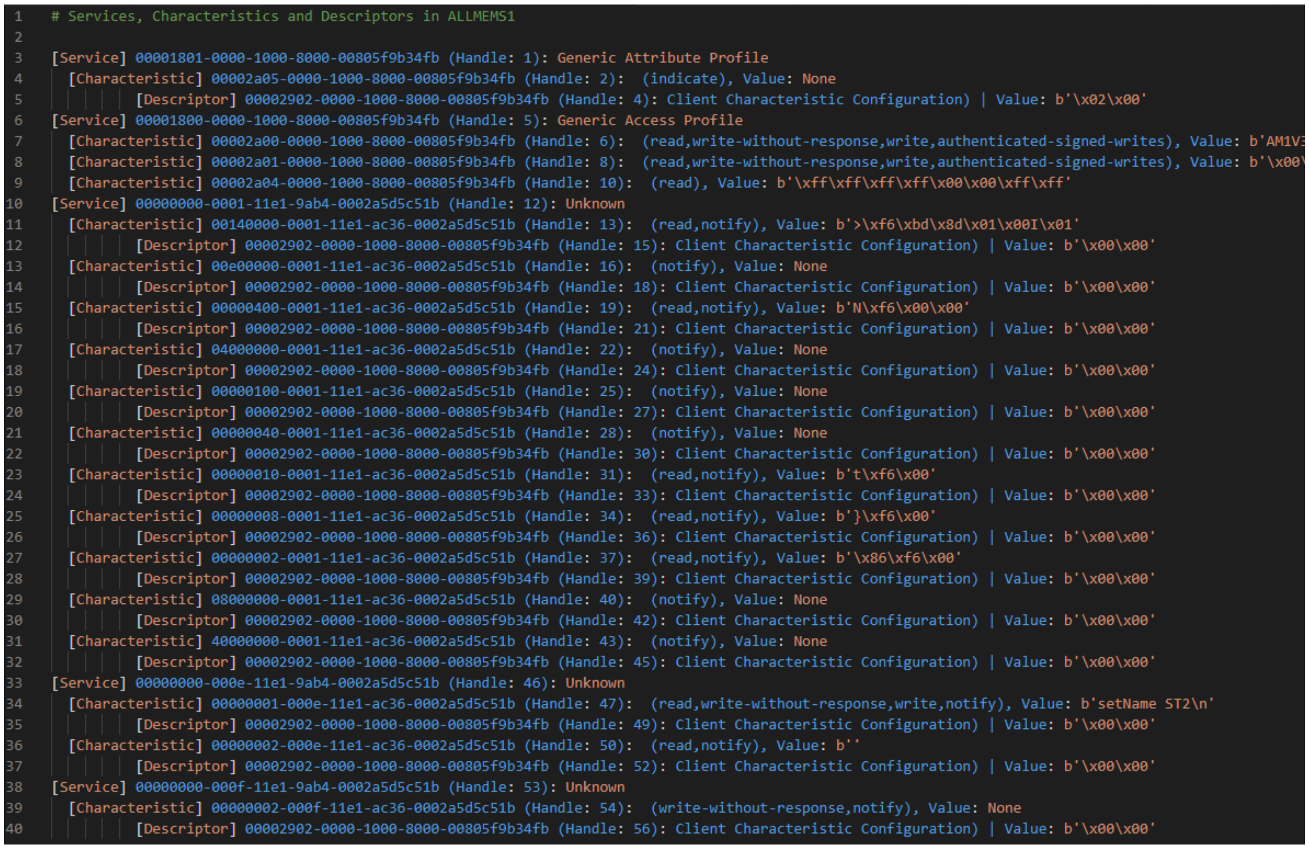
Figure 6

Technical Description

As mentioned above, the system's architecture evolved from its original concept to a simplified one, making it more affordable, easy to use and precise with the addition of the magnetometer. The following pages describe in detail this new approach and how it is designed.

Firmware

The first component of the solution is the Firmware. I am using ALLMEMS1 V.3.1.0 developed in C++ by ST Microelectronics. I used ALLMEMS1 V.3.1.0 because it uses all sensors in the SensorTile and allows me to experiment in different ways to understand how services, characteristics, and descriptors work. This is what we have in the SensorTile with this firmware:



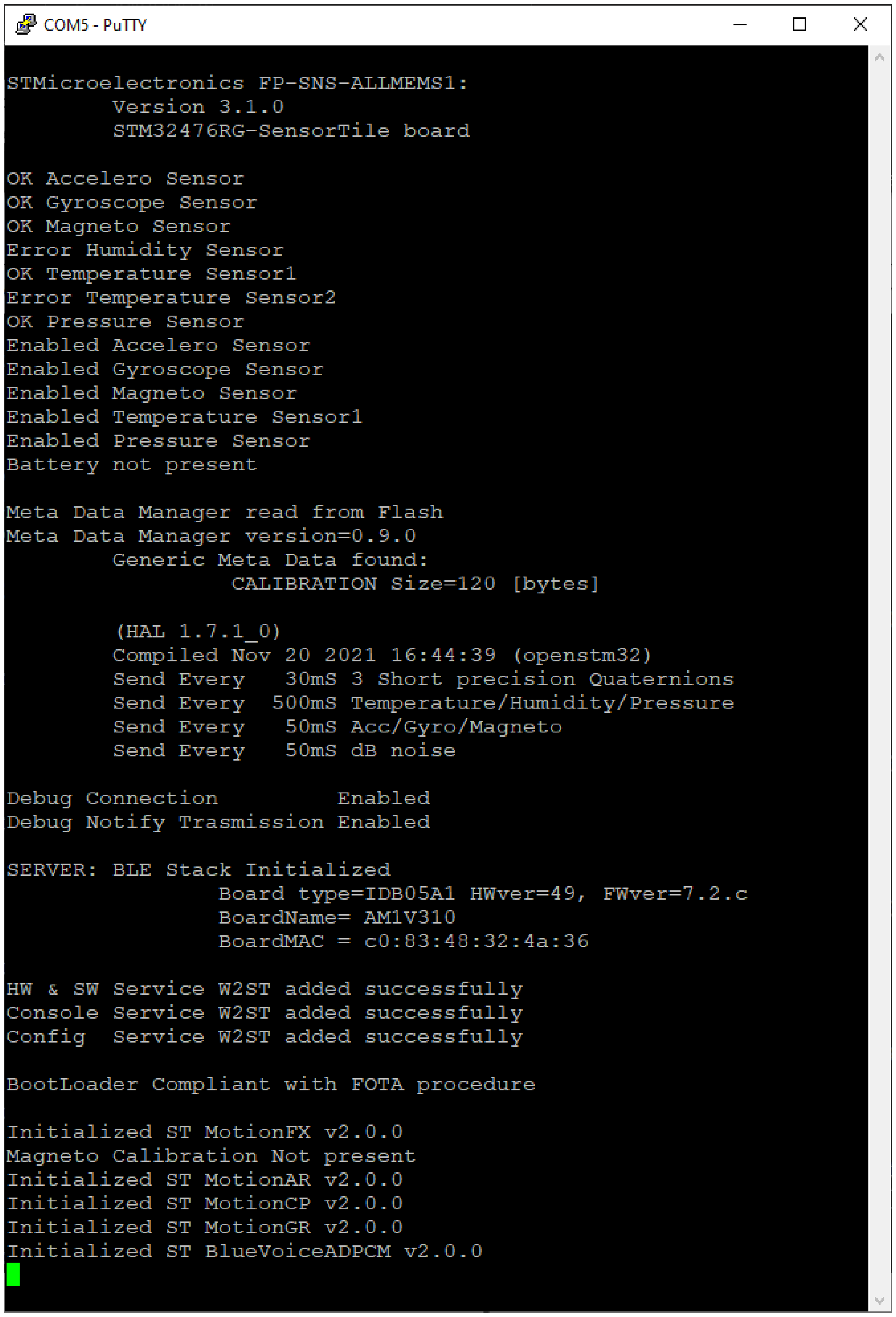
```
# Services, Characteristics and Descriptors in ALLMEMS1

[Service] 00001801-0000-1000-8000-00805f9b34fb (Handle: 1): Generic Attribute Profile
  [Characteristic] 00002a05-0000-1000-8000-00805f9b34fb (Handle: 2):  (indicate), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 4): Client Characteristic Configuration) | Value: b'\x02\x00'
[Service] 00001800-0000-1000-8000-00805f9b34fb (Handle: 5): Generic Access Profile
  [Characteristic] 00002a00-0000-1000-8000-00805f9b34fb (Handle: 6):  (read,write-without-response,write,authenticated-signed-writes), Value: b'AM1V
  [Characteristic] 00002a01-0000-1000-8000-00805f9b34fb (Handle: 8):  (read,write-without-response,write,authenticated-signed-writes), Value: b'\x00
  [Characteristic] 00002a04-0000-1000-8000-00805f9b34fb (Handle: 10):  (read), Value: b'\xff\xff\xff\xff\x00\x00\xff\xff'
[Service] 00000000-0001-11e1-9ab4-0002a5d5c51b (Handle: 12): Unknown
  [Characteristic] 00140000-0001-11e1-ac36-0002a5d5c51b (Handle: 13):  (read,notify), Value: b'>\xf6\xbd\x8d\x01\x00I\x01'
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 15): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00e00000-0001-11e1-ac36-0002a5d5c51b (Handle: 16):  (notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 18): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000400-0001-11e1-ac36-0002a5d5c51b (Handle: 19):  (read,notify), Value: b'N\xf6\x00\x00'
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 21): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 04000000-0001-11e1-ac36-0002a5d5c51b (Handle: 22):  (notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 24): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000100-0001-11e1-ac36-0002a5d5c51b (Handle: 25):  (notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 27): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000040-0001-11e1-ac36-0002a5d5c51b (Handle: 28):  (notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 30): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000010-0001-11e1-ac36-0002a5d5c51b (Handle: 31):  (read,notify), Value: b't\xf6\x00'
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 33): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000008-0001-11e1-ac36-0002a5d5c51b (Handle: 34):  (read,notify), Value: b'}\xf6\x00'
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 36): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000002-0001-11e1-ac36-0002a5d5c51b (Handle: 37):  (read,notify), Value: b'\x86\xf6\x00'
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 39): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 08000000-0001-11e1-ac36-0002a5d5c51b (Handle: 40):  (notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 42): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 40000000-0001-11e1-ac36-0002a5d5c51b (Handle: 43):  (notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 45): Client Characteristic Configuration) | Value: b'\x00\x00'
[Service] 00000000-000e-11e1-9ab4-0002a5d5c51b (Handle: 46): Unknown
  [Characteristic] 00000001-000e-11e1-ac36-0002a5d5c51b (Handle: 47):  (read,write-without-response,write,notify), Value: b'setName ST2\n'
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 49): Client Characteristic Configuration) | Value: b'\x00\x00'
  [Characteristic] 00000002-000e-11e1-ac36-0002a5d5c51b (Handle: 50):  (read,notify), Value: b''
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 52): Client Characteristic Configuration) | Value: b'\x00\x00'
[Service] 00000000-000f-11e1-9ab4-0002a5d5c51b (Handle: 53): Unknown
  [Characteristic] 00000002-000f-11e1-ac36-0002a5d5c51b (Handle: 54):  (write-without-response,notify), Value: None
        [Descriptor] 00002902-0000-1000-8000-00805f9b34fb (Handle: 56): Client Characteristic Configuration) | Value: b'\x00\x00'
```

Figure 7

For this project I used `00e00000-0001-11e1-ac36-0002a5d5c51b (Handle: 16)` to read all values of the three sensors I am interested in: Accelerometer, magnetometer, and gyroscope.

The code to obtain the information shown in Figure 7 is also part of this project. It can be found as SERVICE_EXPLORER.PY, included in the source code.

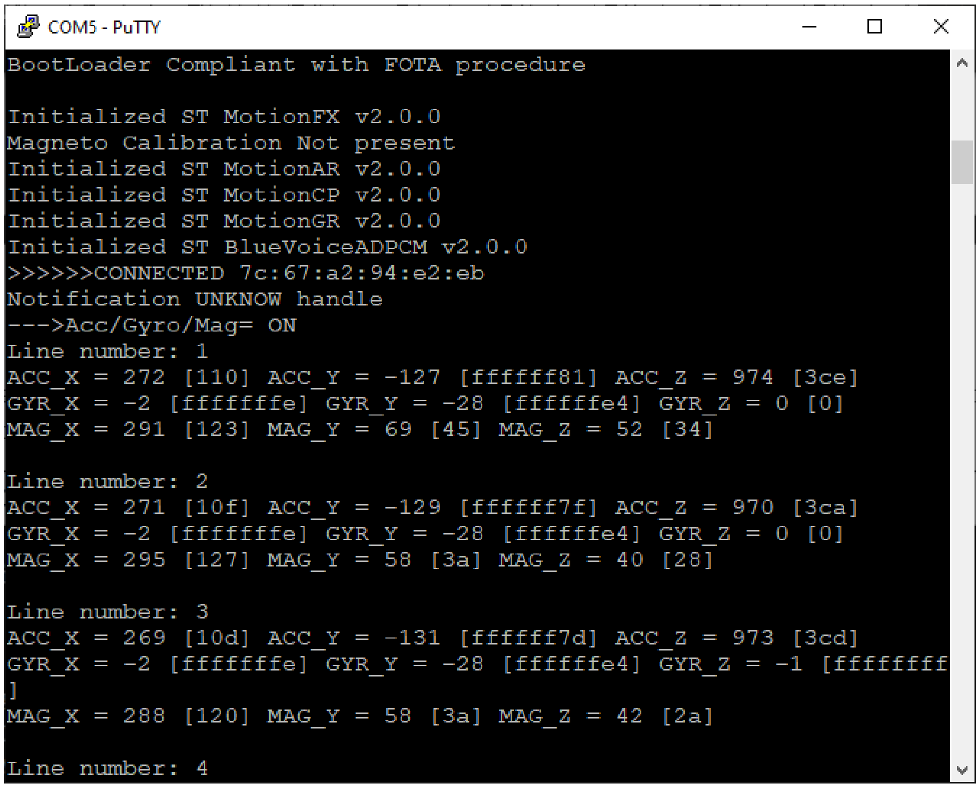
The firmware is configured as shown in Figure 8:



```
COM5 - PuTTY

STMicroelectronics FP-SNS-ALLMEMS1:
        Version 3.1.0
        STM32476RG-SensorTile board

OK Accelero Sensor
OK Gyroscope Sensor
OK Magneto Sensor
Error Humidity Sensor
OK Temperature Sensor1
Error Temperature Sensor2
OK Pressure Sensor
Enabled Accelero Sensor
Enabled Gyroscope Sensor
Enabled Magneto Sensor
Enabled Temperature Sensor1
Enabled Pressure Sensor
Battery not present

Meta Data Manager read from Flash
Meta Data Manager version=0.9.0
        Generic Meta Data found:
                 CALIBRATION Size=120 [bytes]

        (HAL 1.7.1_0)
        Compiled Nov 20 2021 16:44:39 (openstm32)
        Send Every   30mS 3 Short precision Quaternions
        Send Every  500mS Temperature/Humidity/Pressure
        Send Every   50mS Acc/Gyro/Magneto
        Send Every   50mS dB noise

Debug Connection          Enabled
Debug Notify Trasmission Enabled

SERVER: BLE Stack Initialized
                Board type=IDB05A1 HWver=49, FWver=7.2.c
                BoardName= AM1V310
                BoardMAC = c0:83:48:32:4a:36

HW & SW Service W2ST added successfully
Console Service W2ST added successfully
Config  Service W2ST added successfully

BootLoader Compliant with FOTA procedure

Initialized ST MotionFX v2.0.0
Magneto Calibration Not present
Initialized ST MotionAR v2.0.0
Initialized ST MotionCP v2.0.0
Initialized ST MotionGR v2.0.0
Initialized ST BlueVoiceADPCM v2.0.0
```

Figure 8

Figure 8 tells us that SensorTile will send accelerometer, gyroscope, and magnetometer data every 50ms. Figure 9 shows the data in PuTTY:



```
BootLoader Compliant with FOTA procedure

Initialized ST MotionFX v2.0.0
Magneto Calibration Not present
Initialized ST MotionAR v2.0.0
Initialized ST MotionCP v2.0.0
Initialized ST MotionGR v2.0.0
Initialized ST BlueVoiceADPCM v2.0.0
>>>>>>CONNECTED 7c:67:a2:94:e2:eb
Notification UNKNOW handle
--->Acc/Gyro/Mag= ON
Line number: 1
ACC_X = 272 [110] ACC_Y = -127 [ffffff81] ACC_Z = 974 [3ce]
GYR_X = -2 [fffffffe] GYR_Y = -28 [ffffffe4] GYR_Z = 0 [0]
MAG_X = 291 [123] MAG_Y = 69 [45] MAG_Z = 52 [34]

Line number: 2
ACC_X = 271 [10f] ACC_Y = -129 [ffffff7f] ACC_Z = 970 [3ca]
GYR_X = -2 [fffffffe] GYR_Y = -28 [ffffffe4] GYR_Z = 0 [0]
MAG_X = 295 [127] MAG_Y = 58 [3a] MAG_Z = 40 [28]

Line number: 3
ACC_X = 269 [10d] ACC_Y = -131 [ffffff7d] ACC_Z = 973 [3cd]
GYR_X = -2 [fffffffe] GYR_Y = -28 [ffffffe4] GYR_Z = -1 [ffffffff
]
MAG_X = 288 [120] MAG_Y = 58 [3a] MAG_Z = 42 [2a]

Line number: 4
```

Figure 9

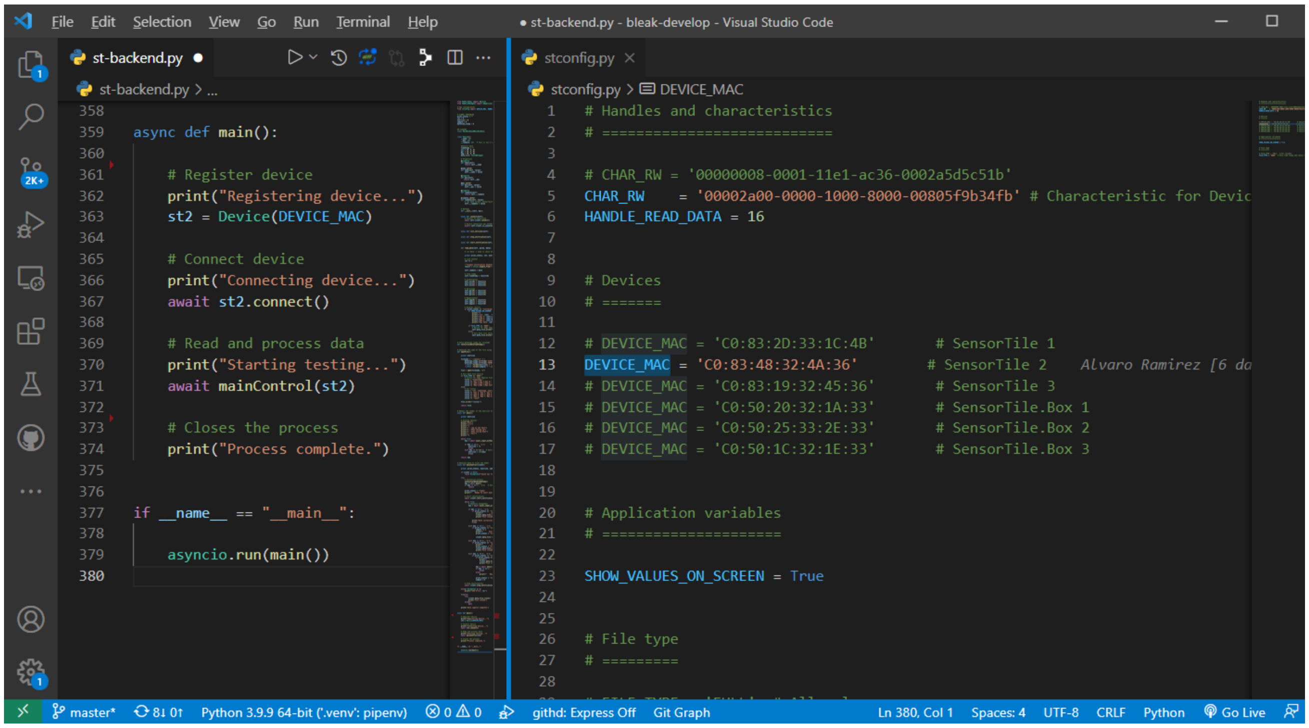
This data is sent as a 20-byte string using little endian format.

Data Capture Application

The second component of the solution is the Data Capture Application. This is a Python program that runs in Windows to collect the data from the SensorTile via BLE. Although the source code has dozens of files, the main program is ST-BACKEND.PY. This program searches for the SensorTile by MAC address (defined in STCONFIG.PY as `DEVICE_MAC`). If found, the program registers the device, connects to it, reads, and returns data and finally disconnects the device.

The development of ST-BACKEND.PY was one of the most challenging tasks in the project, since it required to understand in detail how BLE works, how services, characteristics, and descriptors work, how asyncio works, how can keyboard can be captured when there are coroutines working in an infinite loop and a few other things. Fortunately, I found Bleak (by Henrik Blidh), which is a great API that works very well on Windows. ST Microelectronics also has its own library for Python (BlueST-SDK) which seems to be very powerful, but it does not work for Windows. However, BlueST-SDK works in Linux, Android, and iOS, meaning that it could be considered for future implementations, especially if the project goes mobile.

The following figure shows the `main()` method of the application:


File Edit Selection View Go Run Terminal Help
st-backend.py - bleak-develop - Visual Studio Code
st-backend.py
stconfig.py > DEVICE_MAC
master* 81 0↑ Python 3.9.9 64-bit ('.venv': pipenv) 0 0 githd: Express Off Git Graph Ln 380, Col 1 Spaces: 4 UTF-8 CRLF Python Go Live

```
async def main():

    # Register device
    print("Registering device...")
    st2 = Device(DEVICE_MAC)

    # Connect device
    print("Connecting device...")
    await st2.connect()

    # Read and process data
    print("Starting testing...")
    await mainControl(st2)

    # Closes the process
    print("Process complete.")


if __name__ == "__main__":

    asyncio.run(main())

```

```
# Handles and characteristics
# ==========================

# CHAR_RW = '00000008-0001-11e1-ac36-0002a5d5c51b'
CHAR_RW     = '00002a00-0000-1000-8000-00805f9b34fb' # Characteristic for Devic
HANDLE_READ_DATA = 16


# Devices
# =======

# DEVICE_MAC = 'C0:83:2D:33:1C:4B'       # SensorTile 1
DEVICE_MAC = 'C0:83:48:32:4A:36'        # SensorTile 2    Alvaro Ramirez [6 da
# DEVICE_MAC = 'C0:83:19:32:45:36'       # SensorTile 3
# DEVICE_MAC = 'C0:50:20:32:1A:33'       # SensorTile.Box 1
# DEVICE_MAC = 'C0:50:25:33:2E:33'       # SensorTile.Box 2
# DEVICE_MAC = 'C0:50:1C:32:1E:33'       # SensorTile.Box 3


# Application variables
# =====================

SHOW_VALUES_ON_SCREEN = True


# File type
# =========

```

Figure 10

Note that `main()` is defined as `async`. The `asyncio` library was used to allow asynchronous processing.

ST-BACKEND.PY has a main class called `Device()`, which defines the properties and methods offered by the `device` object in code. In ST-BACKGROUND.PY, properties are defined as follows:

```
class Device():
    __name: str
    __mac: str
    __rawdata: str   # This is not a string. Update.

    # Sensor's data
    timeStamp = ""
    acc = [0, 0, 0]
    gyr = [0, 0, 0]
    mag = [0, 0, 0]
    data_file: TextIOWrapper

    # Properties
    @property
    def name(self):
        return self.__name

    @name.setter
    def name(self, value):
        self.__name = value

    @property
    def mac(self):
       return self.__mac

    @mac.setter
    def mac(self, value):
        self.__mac = value

    @property
    def rawdata(self):
        return self.__rawdata

    @rawdata.setter
    def rawdata(self, value):
        self.__rawdata = value
```

Looking at Figure 9, you can see the hexadecimal representation of each value. In ST-BACKEND.PY we read that data as a 20-byte bytearray(). It looks like this:

```
# raw data format:
bytearray(b'\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00')
```

Where the byte array (read from left to right) is organized as described in the table below. Remember that each pair of bytes is in little endian format:

Bytes	Value
1-2	Datetime stamp
3-4	Acceleration X
5-6	Acceleration Y
7-8	Acceleration Z
9-10	Gyroscope X
11-12	Gyroscope Y
13-14	Gyroscope Z
15-16	Magnetometer X
17-18	Magnetometer Y
19-20	Magnetometer Z

ST-BACKEND.PY generates files in csv format to be consumed by the neural network.

Data Cleaning

Data Cleaning had two main phases in this project. The first phase occurred when I was developing my own neural network. In this phase, data is stored in one big file to feed the neural network as train and test data. For this phase, I developed a few macros in Excel using VBA to clean the data. The purpose of these macros is:

- Count how many exercises there are for the process.
- Count how many samples there are per exercise.
- Count how many records there are per sample.
- Obtain the minimum number of samples per exercise.

- Reduce the number of samples of all exercise to the minimum number of records per exercise.
- Obtain the minimum number of records per sample.
- Reduce all samples for each exercise to the minimum number of records per sample for that exercise.

After running these macros, all exercises end with the same number of samples and all samples within an exercise end with the same number of records. To reduce the number of records, the macro does the following:

- Compare the number of records in the sample (nrs) with the minimum number of records per sample (min) for that exercise.
- If nrs > min:
    - Remove the first record
- If nrs > min:
    - Remove the last record
- If nrs >min:
    - rtr (records to remove) = nrs – min
    - jump = nrs \ (rtr + 1)
    - Remove one record every jump records

In this algorithm, if there is at least one record difference between the sample and the minimum, the first record is removed because usually there are multiple similar records at the beginning of the exercise meaning that it should not affect significatively the result. Similarly, if the initial difference between nrs and min is at least two, not only the first record is removed but also the last one. Th last record is removed because like the first one, it is normal that at the end of the exercise there are several records that are similar.

If the initial difference between nrs and min is greater than two, the first and the last records are removed but we still need to remove more to make nrs = min. Those additional records to be removed are rtr. The question here is: which records must be removed? What I did was to define an algorithm to remove records keeping the same distance between them. For instance, if sample n has 28 records as shown in Figure 11,

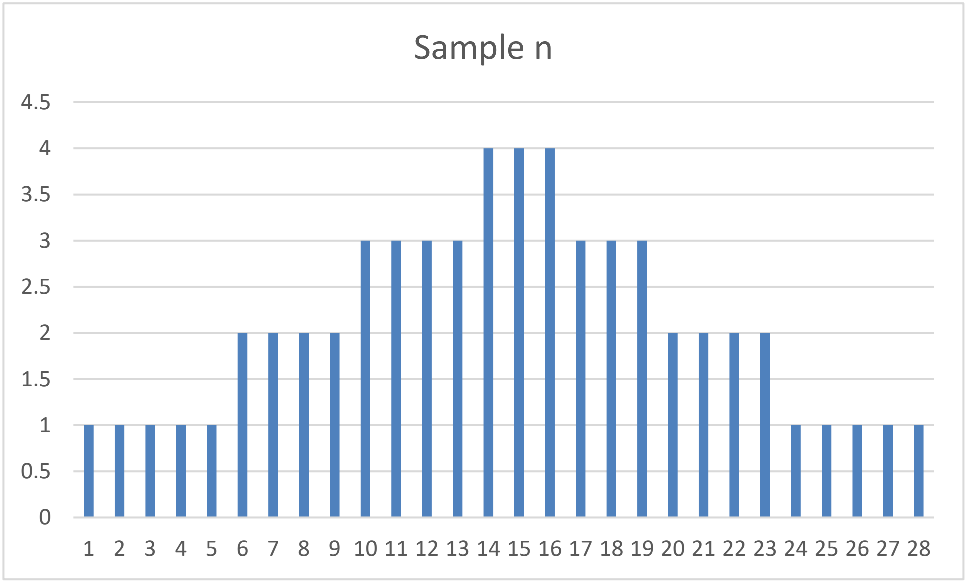


Figure 11

I would do the following depending on how many records I need to remove (red indicates records to be removed):

Remove 1 record:

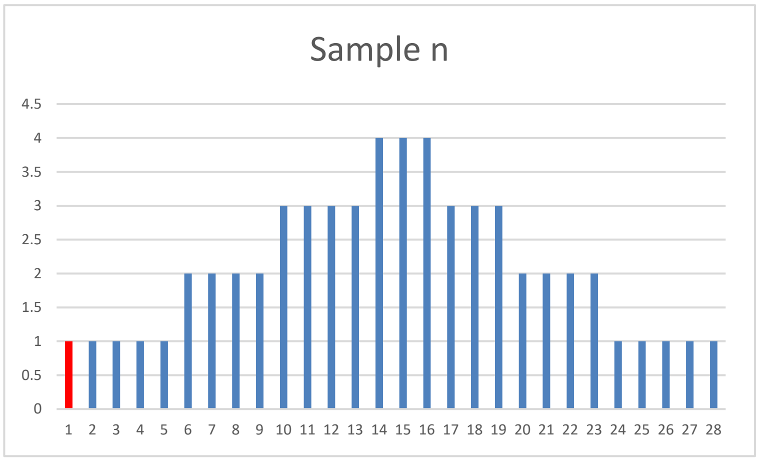


Figure 12

Remove 2 records:

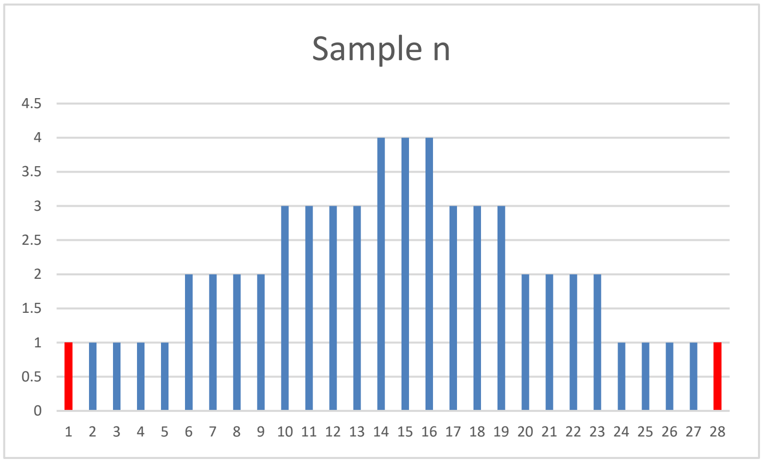


Figure 13

Remove 3 records:

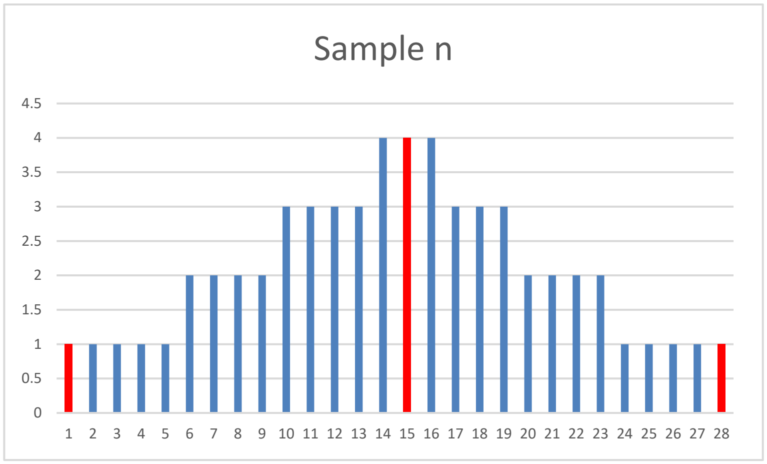


Figure 14

Remove 4 records:

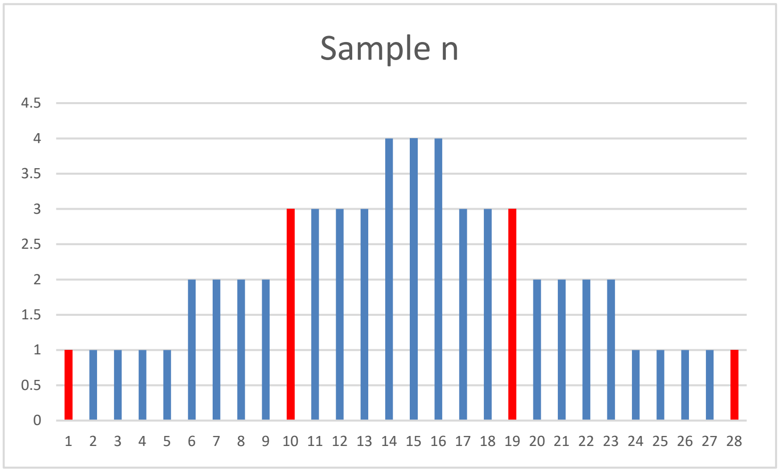
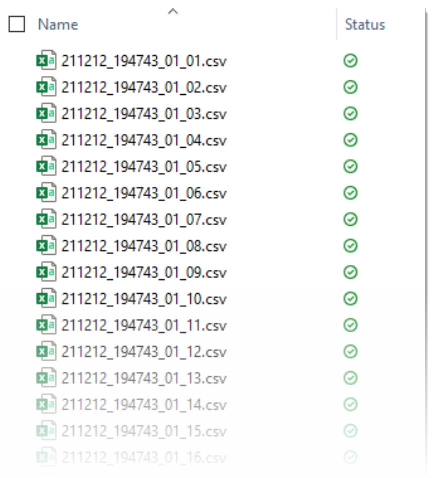


Figure 15

And so on.

But I got some news: Edge Impulse could make life easier. With that in mind, I started the implementation in that tool. That required a change in ST-BACKEND.PY to split the files into samples. The final implementation uses the datetime stamp, the number of the exercise and the number of the sample to name each file, as shown in Figure 16:

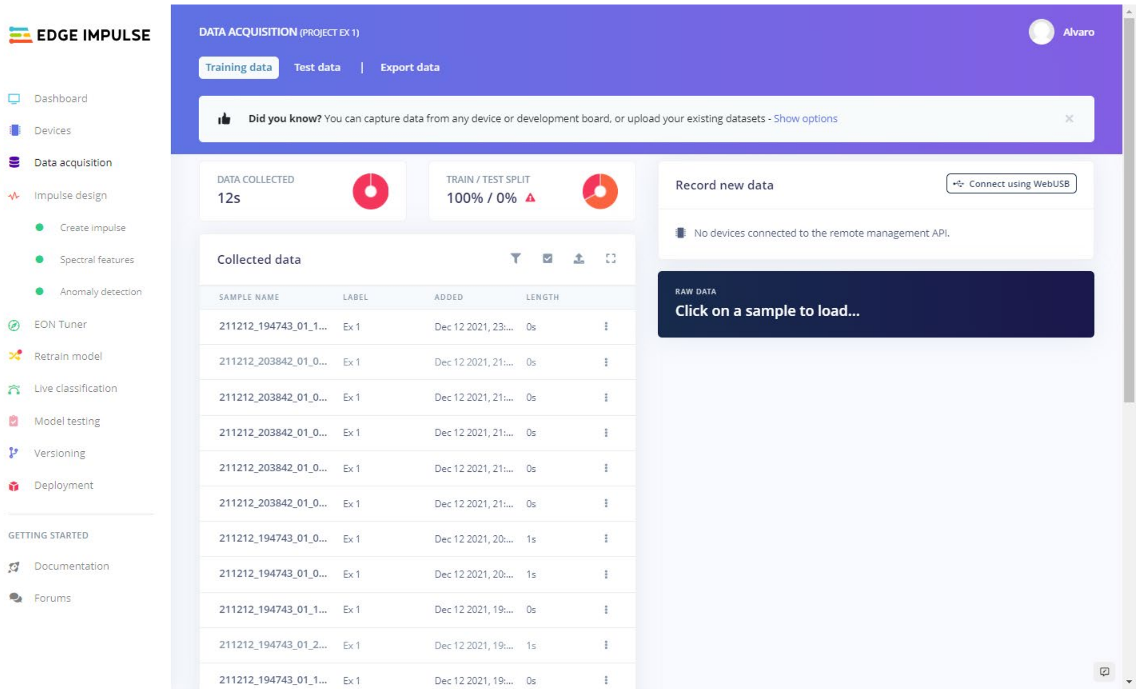


Name	Status
211212_194743_01_01.csv	✓
211212_194743_01_02.csv	✓
211212_194743_01_03.csv	✓
211212_194743_01_04.csv	✓
211212_194743_01_05.csv	✓
211212_194743_01_06.csv	✓
211212_194743_01_07.csv	✓
211212_194743_01_08.csv	✓
211212_194743_01_09.csv	✓
211212_194743_01_10.csv	✓
211212_194743_01_11.csv	✓
211212_194743_01_12.csv	✓
211212_194743_01_13.csv	✓
211212_194743_01_14.csv	✓
211212_194743_01_15.csv	✓
211212_194743_01_16.csv	✓

Figure 16

I am not explaining in this document what was done in Phase 1 after this step since it is not included in the results of the project.

Once in Edge Impulse, the number of records per sample is irrelevant but the system has a bug that rejects some of the samples. This is how the Data acquisition page looks like:



DATA ACQUISITION (PROJECT EX 1)

Did you know? You can capture data from any device or development board, or upload your existing datasets - Show options

DATA COLLECTED 12s

TRAIN / TEST SPLIT 100% / 0%

Record new data

No devices connected to the remote management API.

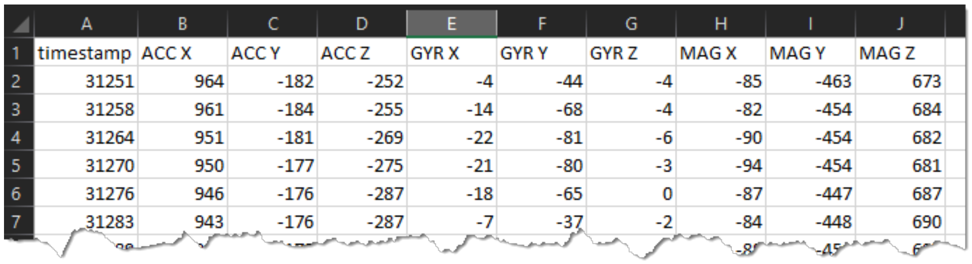
RAW DATA
Click on a sample to load...

Collected data

SAMPLE NAME	LABEL	ADDED	LENGTH
211212_194743_01_1...	Ex 1	Dec 12 2021, 23:...	0s
211212_203842_01_0...	Ex 1	Dec 12 2021, 21:...	0s
211212_203842_01_0...	Ex 1	Dec 12 2021, 21:...	0s
211212_203842_01_0...	Ex 1	Dec 12 2021, 21:...	0s
211212_203842_01_0...	Ex 1	Dec 12 2021, 21:...	0s
211212_203842_01_0...	Ex 1	Dec 12 2021, 21:...	0s
211212_194743_01_0...	Ex 1	Dec 12 2021, 20:...	1s
211212_194743_01_0...	Ex 1	Dec 12 2021, 20:...	1s
211212_194743_01_1...	Ex 1	Dec 12 2021, 19:...	0s
211212_194743_01_2...	Ex 1	Dec 12 2021, 19:...	1s
211212_194743_01_1...	Ex 1	Dec 12 2021, 19:...	0s

Figure 17

The files uploaded to Edge Impulse are generated with ST-BACKEND.PY. These files are in .csv format and have the following structure:



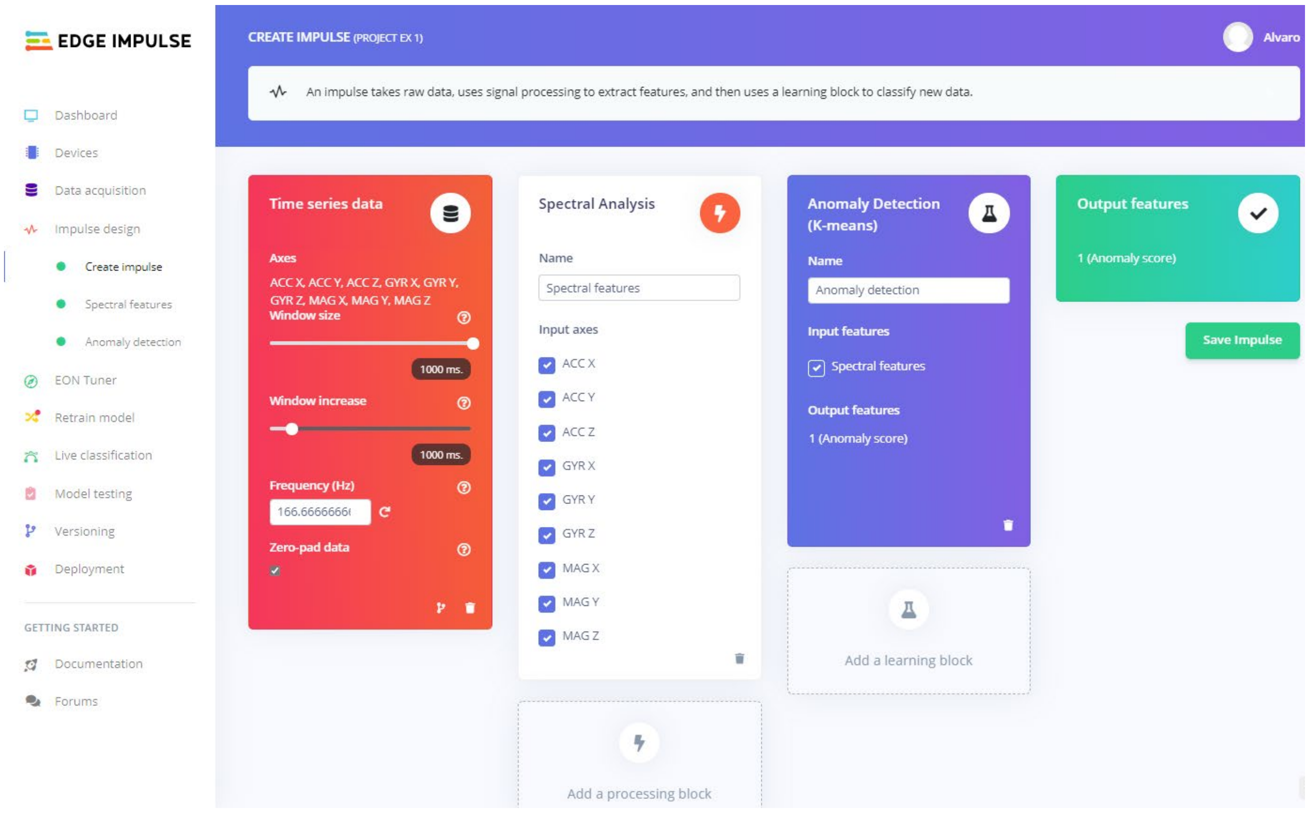
	A	B	C	D	E	F	G	H	I	J
1	timestamp	ACC X	ACC Y	ACC Z	GYR X	GYR Y	GYR Z	MAG X	MAG Y	MAG Z
2	31251	964	-182	-252	-4	-44	-4	-85	-463	673
3	31258	961	-184	-255	-14	-68	-4	-82	-454	684
4	31264	951	-181	-269	-22	-81	-6	-90	-454	682
5	31270	950	-177	-275	-21	-80	-3	-94	-454	681
6	31276	946	-176	-287	-18	-65	0	-87	-447	687
7	31283	943	-176	-287	-7	-37	-2	-84	-448	690

Figure 18

Data Processing

Model Definition

Once data is uploaded in Edge Impulse, the next step is to define what to do with it. As mentioned earlier in this document, I decided to go with an anomaly detection neural network. The model I used was K-means. Figure 18 shows how this page looks after the model was configured. Note that this screen is for Exercise 1. The other two exercises were defined the same. The References section at the end of the document have links to the public projects in Edge Impulse.



An impulse takes raw data, uses signal processing to extract features, and then uses a learning block to classify new data.

Time series data

Axes: ACC X, ACC Y, ACC Z, GYR X, GYR Y, GYR Z, MAG X, MAG Y, MAG Z

Window size: 1000 ms.

Window increase: 1000 ms.

Frequency (Hz): 166.66666666

Zero-pad data: ✓

Spectral Analysis

Name: Spectral features

Input axes: ✓ ACC X, ✓ ACC Y, ✓ ACC Z, ✓ GYR X, ✓ GYR Y, ✓ GYR Z, ✓ MAG X, ✓ MAG Y, ✓ MAG Z

Anomaly Detection (K-means)

Name: Anomaly detection

Input features: ✓ Spectral features

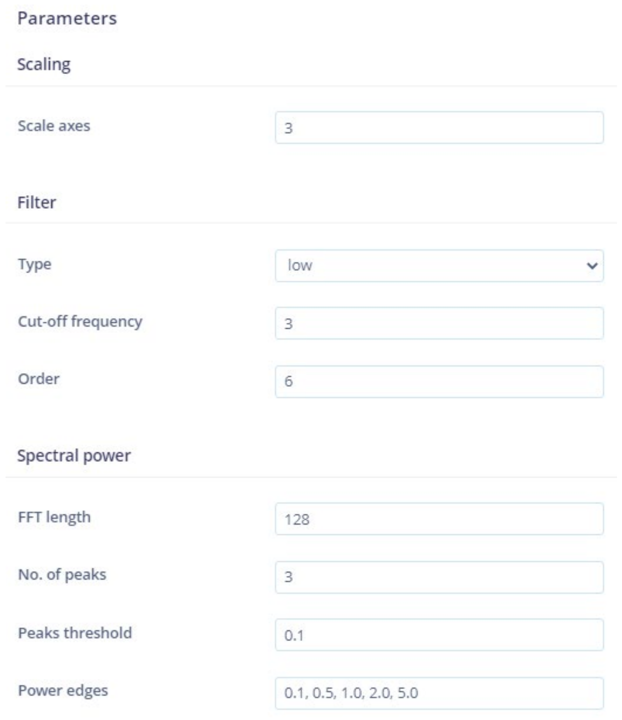
Output features: 1 (Anomaly score)

Output features

1 (Anomaly score)

Figure 19

The next step was to define the parameters. All three exercises use the same values as shown below:



Parameters

Scaling

Scale axes	3

Filter

Type	low
Cut-off frequency	3
Order	6

Spectral power

FFT length	128
No. of peaks	3
Peaks threshold	0.1
Power edges	0.1, 0.5, 1.0, 2.0, 5.0

Figure 20

Using 3 scale axes, a low filter with a cut-off frequency of 3 and an order of 6, FFT length of 128, 3 peaks, a peak threshold of 0.1 and power edges of 0.1, 0.5, 1.0, 2.0 and 5.0, raw data that usually looks like shown in Figure 21:

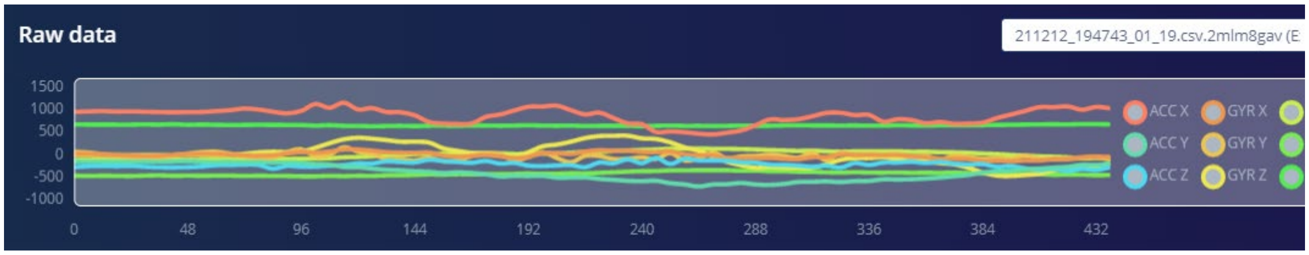


Figure 21

Produces this DSP results:

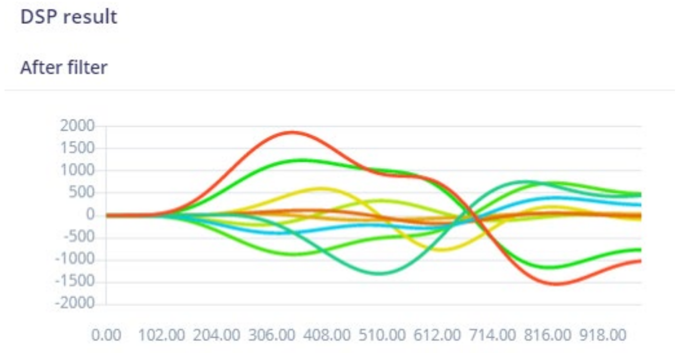


Figure 22

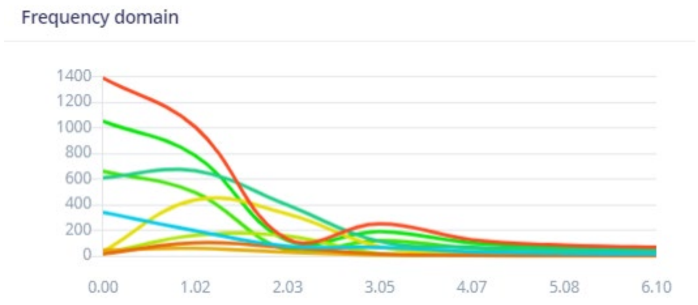


Figure 23

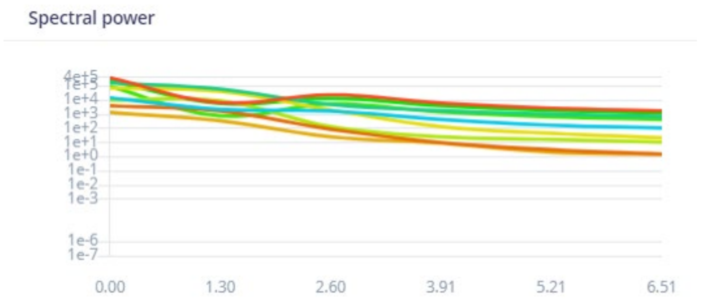


Figure 24

Spectral Features

These parameters allow us to generate the features of the model. This is an example of how they look like using spectral features:

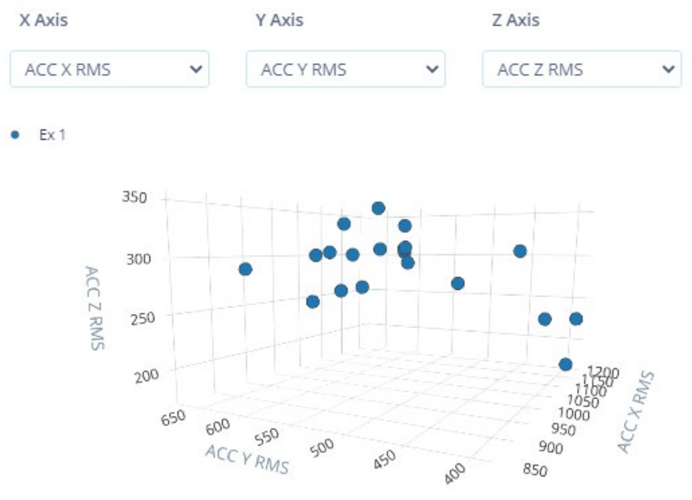


Figure 25

Edge Impulse is a great tool with wonderful graphs. However, the real magic comes when you interact directly with them specially with the 3D graphs like the one in Figure 25.

Training the Anomaly Detection Model

To train the anomaly detection neural network I used 3 clusters with the following axes:

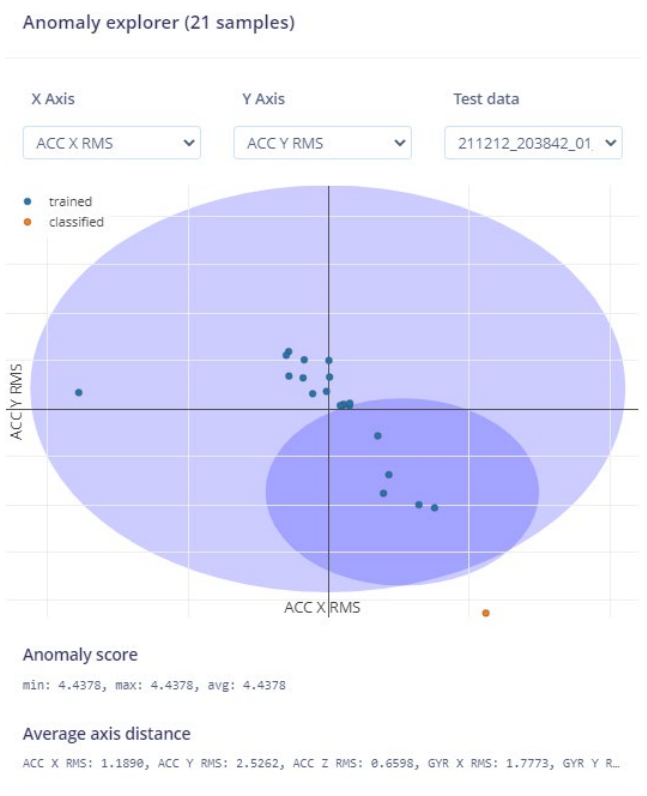
- ACC X RMS
- ACC Y RMS
- ACC Z RMS
- GYR X RMS
- GYR Y RMS
- GYR Z RMS
- MAG X RMS
- MAG Y RMS
- MAG Z RMS

The system has 90 more axes to select with, allowing all combinations. However, it failed when selecting all at the same time. I ran experiments with multiple combinations but the one listed above worked very well.

Running the Test Data with Anomaly Explorer

I ran the test data using the Anomaly Explorer and found that, in general, the model is acceptably accurate considering that the training data set is small. The following figure graphically shows the output for a test sample that should be classified as no anomaly. As it can be observed, the blue dots are the values used for training and the orange dot is our test sample once classified. In this case the system places it outside the cluster indicating it is an anomaly, which is the right answer.

In this model, train data is not labeled. That makes it easier to classify a sample as anomaly and more difficult to classify it as no anomaly.



Anomaly score

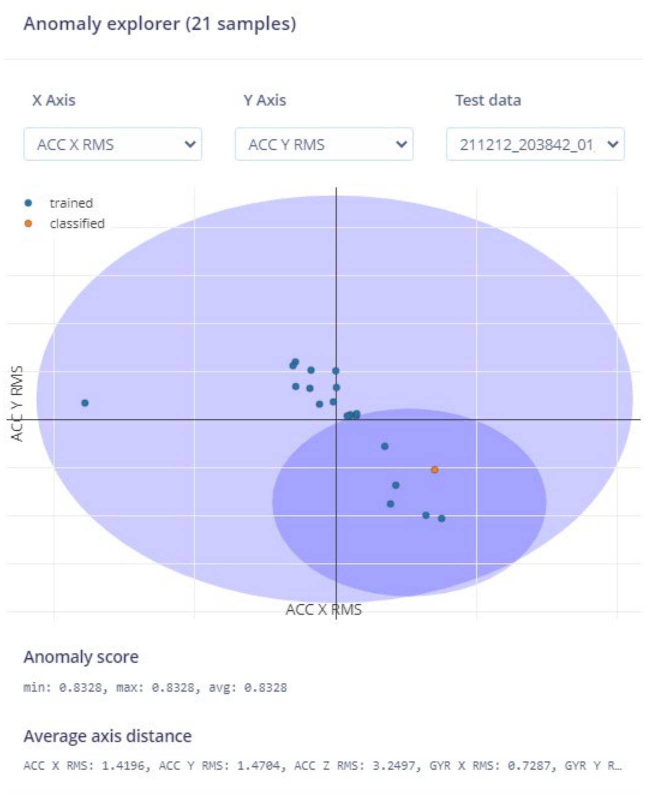
```
min: 4.4378, max: 4.4378, avg: 4.4378
```

Average axis distance

```
ACC X RMS: 1.1890, ACC Y RMS: 2.5262, ACC Z RMS: 0.6598, GYR X RMS: 1.7773, GYR Y R…
```

Figure 26

Let's try now with a test sample for a non-anomaly case. This is what we get:


Anomaly explorer (21 samples)

X Axis: ACC X RMS

Y Axis: ACC Y RMS

Test data: 211212_203842_01

Anomaly score

```
min: 0.8328, max: 0.8328, avg: 0.8328
```

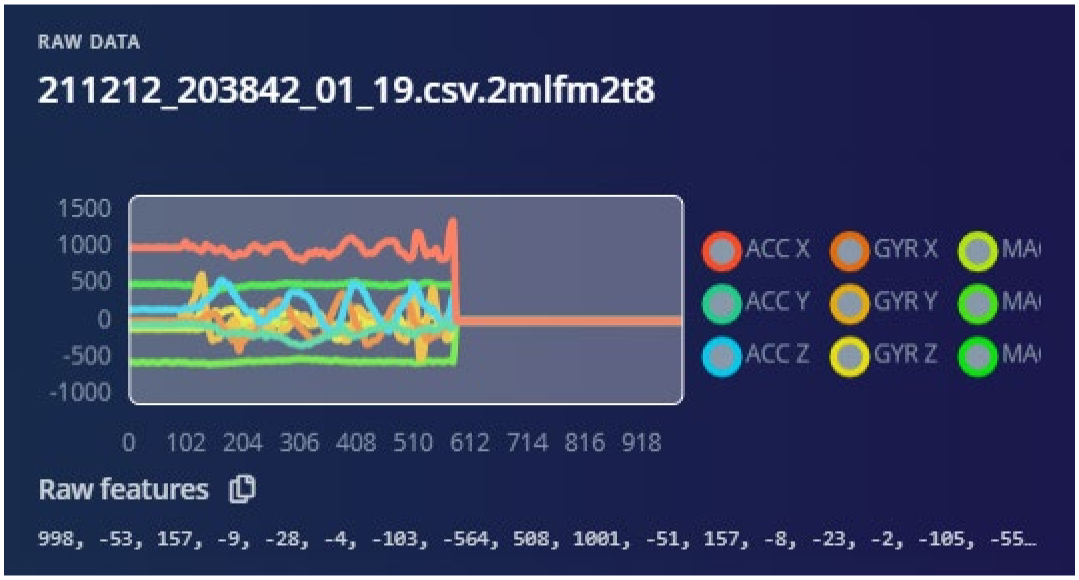
Average axis distance

```
ACC X RMS: 1.4196, ACC Y RMS: 1.4704, ACC Z RMS: 3.2497, GYR X RMS: 0.7287, GYR Y R…
```

Figure 27

Live Classification

I also prepared some samples to be used in live classification. These samples are available for the three exercises. This is what I get for the first sample that belongs to exercise 1. This sample should be classified as anomaly:

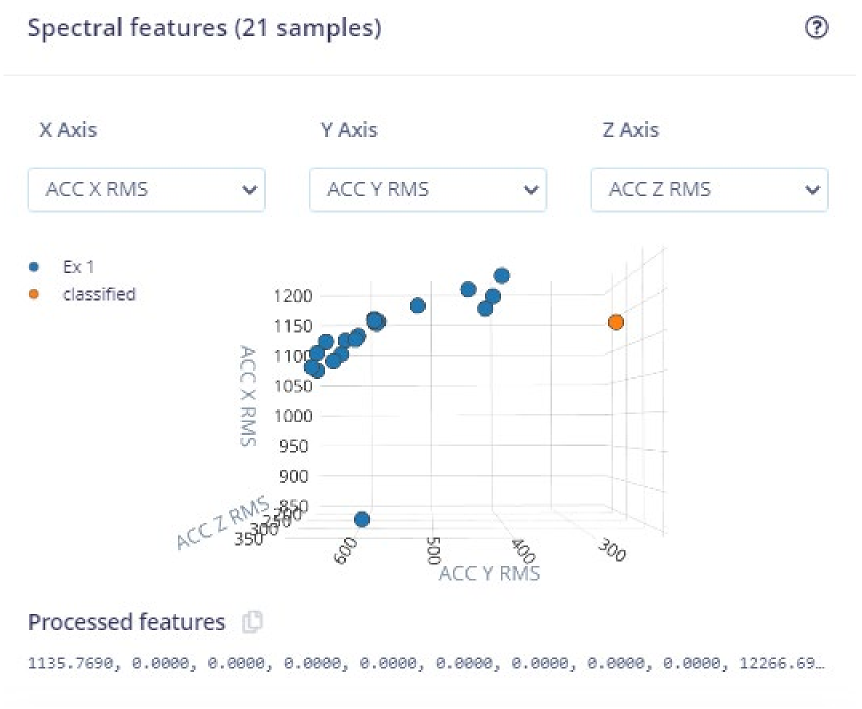

RAW DATA

211212_203842_01_19.csv.2mlfm2t8

Raw features

```
998, -53, 157, -9, -28, -4, -103, -564, 508, 1001, -51, 157, -8, -23, -2, -105, -55…
```

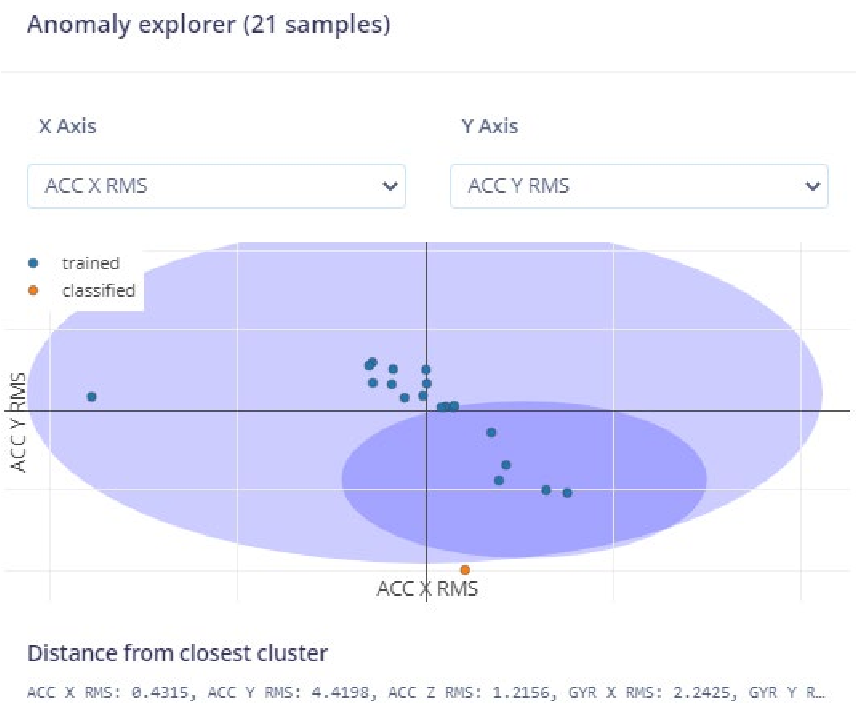
Figure 28



Processed features

```
1135.7690, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 0.0000, 12266.69…
```

Figure 29

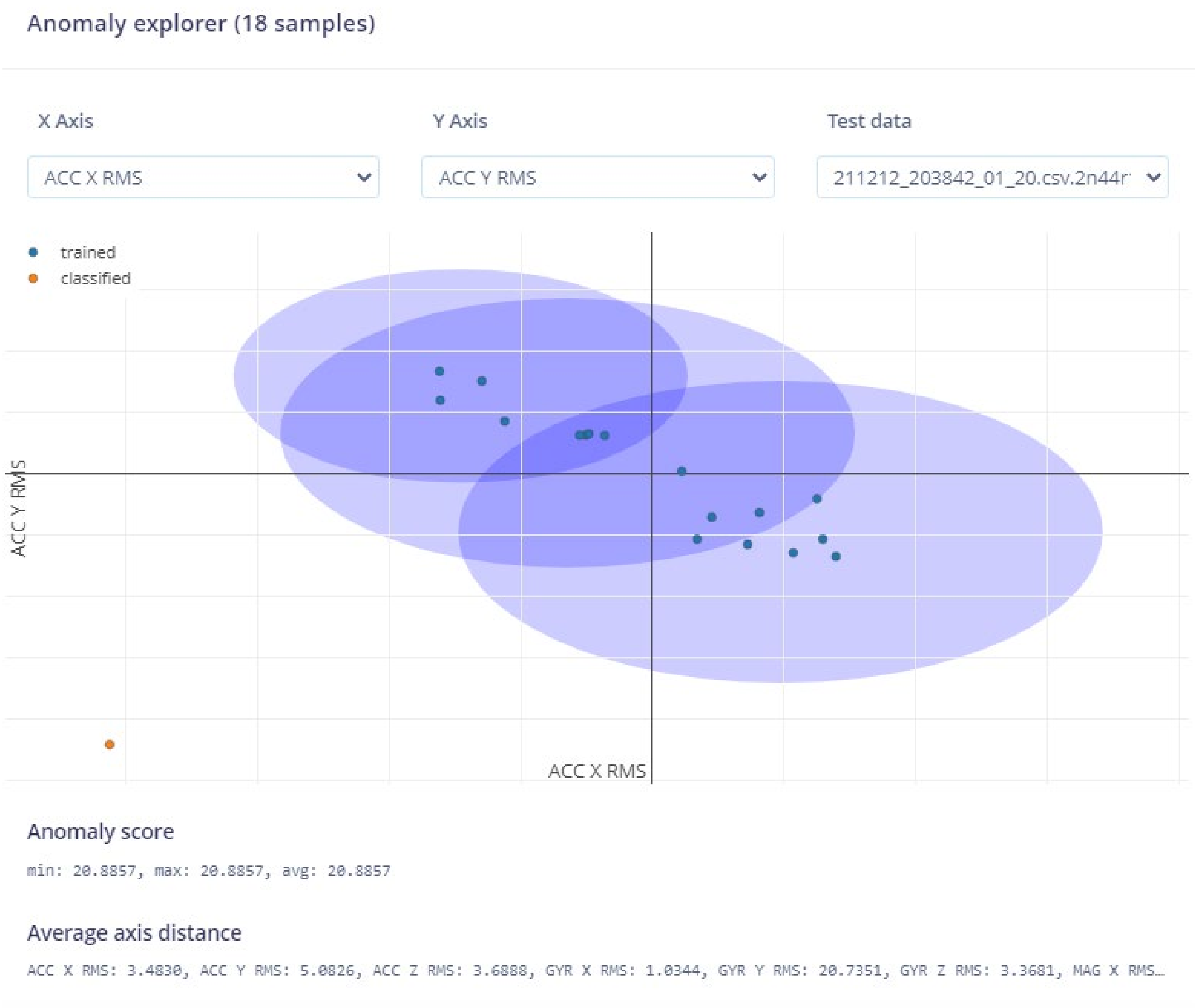


Distance from closest cluster

```
ACC X RMS: 0.4315, ACC Y RMS: 4.4198, ACC Z RMS: 1.2156, GYR X RMS: 2.2425, GYR Y R…
```

Figure 30

Although this sample was classified correctly, we can also see that there is an outlier. After removing the outliers and retraining the model with the new data set, I got this:



Anomaly score

```
min: 20.8857, max: 20.8857, avg: 20.8857
```

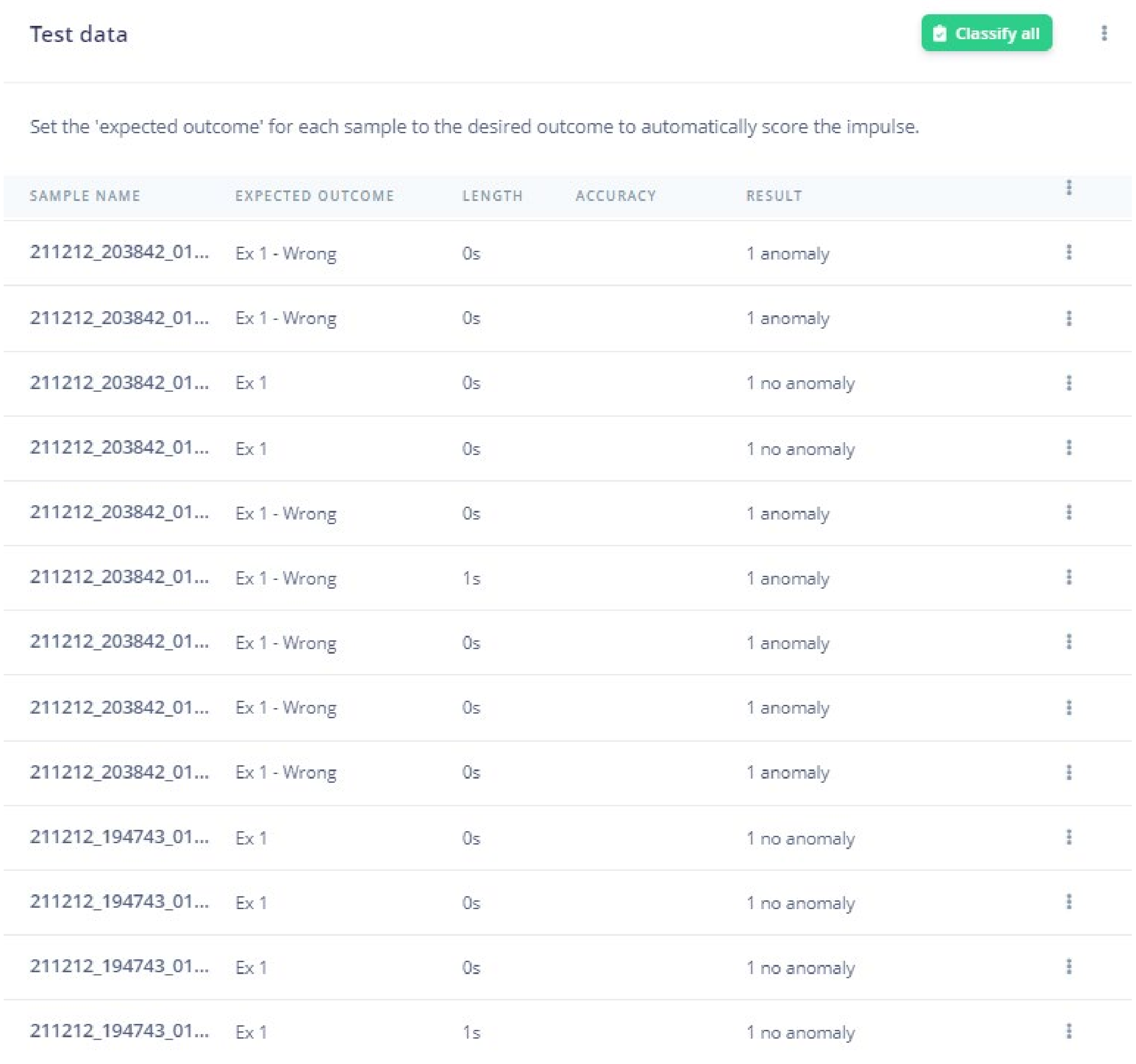
Average axis distance

```
ACC X RMS: 3.4830, ACC Y RMS: 5.0826, ACC Z RMS: 3.6888, GYR X RMS: 1.0344, GYR Y RMS: 20.7351, GYR Z RMS: 3.3681, MAG X RMS…
```

Figure 31

Model Testing

Edge Impulse has one more option called MODEL TESTING. This option allows to test the model with all samples in the test group at once. To make the test more interesting, I added new samples and then run the classification. As you can see below, after removing the outliers I get a 100% accuracy. In DATA GENERATION I explain in detail how data was generated and when it was used.



Test data

Classify all

Set the 'expected outcome' for each sample to the desired outcome to automatically score the impulse.

SAMPLE NAME	EXPECTED OUTCOME	LENGTH	ACCURACY	RESULT
211212_203842_01...	Ex 1 - Wrong	0s		1 anomaly
211212_203842_01...	Ex 1 - Wrong	0s		1 anomaly
211212_203842_01...	Ex 1	0s		1 no anomaly
211212_203842_01...	Ex 1	0s		1 no anomaly
211212_203842_01...	Ex 1 - Wrong	0s		1 anomaly
211212_203842_01...	Ex 1 - Wrong	1s		1 anomaly
211212_203842_01...	Ex 1 - Wrong	0s		1 anomaly
211212_203842_01...	Ex 1 - Wrong	0s		1 anomaly
211212_203842_01...	Ex 1 - Wrong	0s		1 anomaly
211212_194743_01...	Ex 1	0s		1 no anomaly
211212_194743_01...	Ex 1	0s		1 no anomaly
211212_194743_01...	Ex 1	0s		1 no anomaly
211212_194743_01...	Ex 1	1s		1 no anomaly

Figure 32

Data Generation

I generated data in two different sessions. The main difference is that I removed the sensors between sessions and asked the patient to have a long break. Then, there was no way to make sure that sensors would be placed exactly the same as in the first session. In each session I captured the same number of samples (20), but in the first session all samples were done correctly, while in the second session the first 10 samples were correct and the last 10 were incorrect. All these samples were uploaded into Edge Impulse and checked, because the system has a glitch that prevents it to use some of the files. After testing one by one, this is what I finally used for Exercise 1:

Session Captured	File	Accepted by System?	Dataset	Expected Result	Introduced in Step	Result Obtained
1	1	N				
1	2	Y	Train		Spectral Features	N/A
1	3	N				
1	4	Y	Train		Spectral Features	N/A
1	5	N				
1	6	Y	Test	No anomaly	Live Classification	No anomaly
1	7	Y	Test	No anomaly	Live Classification	No anomaly
1	8	Y	Test	No anomaly	Live Classification	No anomaly
1	9	Y	Train		Spectral Features	N/A
1	10	Y	Train		Spectral Features	N/A
1	11	Y	Train		Spectral Features	N/A
1	12	N				
1	13	N				
1	14	Y	Train		Spectral Features	N/A
1	15	Y	Test	No anomaly	Live Classification	No anomaly
1	16	Y	Train		Spectral Features	N/A
1	17	Y	Train		Spectral Features	N/A
1	18	Y	Train		Spectral Features	N/A
1	19	Y	Test	No anomaly	Live Classification	No anomaly
1	20	N				
2	1	Y	Train		Spectral Features	N/A
2	2	Y	Train		Spectral Features	N/A
2	3	Y	Train		Spectral Features	N/A
2	4	Y	Train		Spectral Features	N/A
2	5	N				
2	6	Y	Test	No anomaly	Model Testing	No anomaly
2	7	Y	Train		Spectral Features	N/A
2	8	Y	Train		Spectral Features	N/A
2	9	Y	Train		Spectral Features	N/A
2	10	Y	Train		Spectral Features	N/A
2	11	N				
2	12	Y	Test	No anomaly	Model Testing	No anomaly
2	13	Y	Test	Anomaly	Live Classification	Anomaly
2	14	Y	Test	Anomaly	Live Classification	Anomaly
2	15	N				
2	16	Y	Test	Anomaly	Live Classification	Anomaly
2	17	N				
2	18	Y	Test	Anomaly	Live Classification	Anomaly
2	19	Y	Test	Anomaly	Live Classification	Anomaly
2	20	N				
2	21	Y	Test	Anomaly	Model Testing	Anomaly
2	22	N				
2	23	Y	Test	Anomaly	Model Testing	Anomaly
2	24	Y	Test	Anomaly	Model Testing	Anomaly

Items to Improve

These are some topics that I would like to do better in this project:

- Use larger sample datasets for training and testing
- Better user interface
- Move neural network from Edge Impulse to code
- Sold SensorTile to use it wireless

Deliverables

The solution includes a package with the following components:

- Git repo with ALLMEMS1_V3.1.0 code
  https://github.com/alvaroiramirez/ALLMEMS1

- Git repo with Python code
  alvaroiramirez/SensorTile (github.com)

- Edge Impulse project for exercise 1
  Dashboard - Project Ex 1 - Edge Impulse

- Edge Impulse project for exercise 2
  Dashboard - Project Ex 2 - Edge Impulse

- Edge Impulse project for exercise 3
  Dashboard - Project Ex 3 - Edge Impulse

- Slides used in presentation
  https://github.com/alvaroiramirez/DGMD-E-14-Report

- Project Report (this document)
  https://github.com/alvaroiramirez/DGMD-E-14-Report

- Video: Exercises
  https://youtu.be/lkKPSssySqA

- Video: Data capture
  https://youtu.be/bEqoFrfLb0c

- Video: Model testing
  https://youtu.be/LKWQuNvw3bU

- Video: Live classification
  https://youtu.be/qT03eEJWwTY

Conclusions

This project was a great experience in my Data Science journey. It left a lot of learning and a great friend! These are my main conclusions of the project:

- Well trained unsupervised neural networks using K-means are highly accurate even with a small number of samples.
- Creating applications for wearable devices is not trivial, but with some work they are powerful tools that can be used in many fields that can make life easier.
- It is possible to create low-cost prototypes with wearable devices.

Future Work

I would love to improve this project in the future including these features:

- Mobile GUI instead of using a computer.
- Real time feedback to the patient showing a percentage value that indicates how close the patient is to correctly complete the exercise.
- Allow physical therapists to train their devices for each patient’s needs based on a pre-trained neural network that simplifies the training process.
- Improve the form factor to make it more ergonomic and attractive.
- Improve the frequency management to ensure all samples are captured at the same rate.

References

I used dozens of references. Most of them were websites to solve specific questions that ranged from translations to better understand the meaning of some topics, to programming tips to solve specific issues, to tutorials to learn complex concepts. In addition to that, I used around a dozen books, where the most useful were:

- Chollet, F. (2021). Deep Learning with Python (Second Edition). Shelter Island, NY. Manning Publications Co.
- Fowler, M. (MEAP – To be published in 2022). Python Concurrency with Asyncio. Shelter Island, NY. Manning Publications Co.
- Lott, S. & Phillips, D. (2021). Python Object-Oriented Programming (Fourth Edition). Birmingham, UK. Packt Publishing.
- Chacon, S. & Straub. B. (2014). Pro Git (Second Edition). New York, NY. Apress.
- Torres, J. (2020). Python Deep Learning – Introducción Práctica con Keras y TensorFlow 2. España. Marcombo.
- Ibrahim, D. (2020). How 2 Get Started with the SensorTile.Box. Susteren, The Netherlands. Lektor.
- Woolley, M. (2021). Blutooth Core Specifications Version 5.3 – Feature Enhancements. Bluetooth.

Python Concurrency with asyncio
Matthew Fowler
MEAP
PRINTED IN FULL COLOR
DEEP LEARNING with Python
SECOND EDITION
François Chollet
EXPERT INSIGHT
Python Object-Oriented Programming
Build robust and maintainable object-oriented Python applications and libraries
Fourth Edition
Steven F. Lott
Dusty Phillips
Packt
SECOND EDITION
Pro Git
Scott Chacon and Ben Straub
Apress open
HOW2
Get Started with the SensorTile.box
STmicroelectronics' wireless IoT & wearable sensor development kit
Dogan Ibrahim
elektor
Python Deep Learning
Introducción práctica con Keras y TensorFlow 2
Jordi Torres
Marcombo
Bluetooth® Core Specification Version 5.3 Feature Enhancements
Bluetooth

Workplan

Week	End Date	Goal
1	October 12	• Additional hardware required ordered • Software required identified • Project proposal ready
2	October 19	• Data collecting software for one sensor developed • Data collecting software for one sensor tested • User interface software designed • Physical implementation designed
3	October 26	• User interface software development started • Raspberry setup to control three sensors completed • Statistics to be applied defined
4	November 2	• Tests with three sensors started • Initial data collection started • Statistics preliminary analyses performed
5	November 9	• Training and testing data for machine learning model collected • Machine learning model developed • Machine learning model tested • Statistics adjusted
6	November 16	• Machine learning model trained • Machine learning model deployed • User interface software development completed
7	November 23	• Integration testing • Final adjustments to statistics implemented • Final adjustments to machine learning model implemented • Final adjustments to user interface software implemented • Final adjustments to physical implementation implemented
8	December 7	• Presentation development • Presentation ready