UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Barcelona Est

FINAL DEGREE PROJECT

BSc in Electronic, Industrial, and Automatic Control Engineering

DATA ACQUISITION, PROCESSING, AND INTERPRETATION OF A SINGLE-CHANNEL SURFACE ELECTROMYOGRAPHIC SENSOR WITH MACHINE LEARNING FOR THE SMART WEARABLE THERAPEUTIC GLOVE

Project Report and Attachments

Author: Alexander Ehrich Zapardiel
Advisor: Herminio Martínez García
Co-Advisor: Maziar Ahmadi Zeidabadi
Department: EEL

Call: June 2022

Abstract

Electromyography is the study of electrical signals produced by muscles during contraction representing neuromuscular activities. This project will be based on designing a data acquisition process from a single-channel surface electromyographic sensor to create a dataset, analysing this data, and, subsequently, implement through Machine Learning a model to facilitate the accurate classification of hand and wrist gestures, with a future goal of prevention of injuries.

This project forms part of the Smart Wearable Therapeutic Glove, which aims to help doctors and patients control and monitor the rehabilitation progress of everyone. The glove captures the biomechanical and neurophysiological parameters of the hand by integrating a system of sensors and captures the relative position and movements performed.

Resumen

La electromiografía es el estudio de las señales eléctricas producidas por los músculos durante la contracción representando las actividades neuromusculares. Este proyecto se basará en el diseño de un proceso de adquisición de datos a partir de un sensor electromiográfico de superficie con un único canal de medida para crear un conjunto de datos, analizar estos datos e, posteriormente, implementar mediante Machine Learning un modelo para facilitar la clasificación precisa de los gestos de la mano y la muñeca, con un objetivo futuro de prevención de posibles lesiones.

Este proyecto forma parte del Smart Wearable Therapeutic Glove, cuyo objetivo es ayudar a los médicos y a los pacientes a controlar y monitorizar el progreso de la rehabilitación de cada individuo. El guante capta los parámetros biomecánicos y neurofisiológicos de la mano mediante la integración de un sistema de sensores, y capta la posición relativa y los movimientos realizados.

Resum

L'electromiografia és l'estudi dels senyals elèctrics produïts pels músculs durant la contracció representant les activitats neuromusculars. Aquest projecte es basarà en el disseny d'un procés d'adquisició de dades a partir d'un sensor electromiogràfic de superfície amb un únic canal de mesura per crear un conjunt de dades, analitzar aquestes dades i, posteriorment, implementar mitjançant Machine Learning un model per facilitar la classificació precisa dels gestos de la mà i el canell, amb un objectiu futur de prevenció de possibles lesions.

Aquest projecte forma part del Smart Wearable Therapeutic Glove, l'objectiu del qual és ajudar els metges i els pacients a controlar i monitoritzar el progrés de la rehabilitació de cada individu. El guant capta els paràmetres biomecànics i neurofisiològics de la mà mitjançant la integració d'un sistema de sensors, i capta la posició relativa i els moviments realitzats.

Acknowledgements

First, I would like to thank Herminio Martínez García, director of this thesis, and Maziar Ahmadi Zeidabadi, co-director of this thesis, for their help and involvement in the project throughout the project, from the first day to the last. As well as for the external connections offered in case of problems, and for the constant monitoring of the evolution of the project.

I would especially like to thank Gerard Rodríguez Martínez for the help offered in the hardware section of the project and for his patience in answering all my queries and doubts during these six months.

Finally, I would like to thank my family, my friends, and my partner for helping me not to give up, to keep going, and to give me that necessary push in the hardest moments.

Thanks to all of you.

Glossary

- AI: Artificial Intelligence
- ML: Machine Learning
- EMG: Electromyogram
- sEMG: surface electromyogram
- ADC: Analog to digital converter
- ANN: Artificial neural network
- NN: Neural network
- SVM: Support Vector Machine
- SVC: Support Vector Classifier
- KNN: K-Nearest neighbours

Index

Figures Index

Tables Index

1. Preface

1.2. Motivation

The main motivation and reasons for choosing this project were to be able to tackle a problem from a subject of which one had no previous knowledge and to learn from it, as it applies concepts and knowledge that has been acquired during the whole years of university. Another motivation has been the personal willingness to learn about Artificial Intelligence and to understand how it works and develops.

1.3. Previous Requirements

For the development of this project, previous knowledge was required in electronic systems for the assembling of the circuit, programming languages: C (for the firmware code) and Python (for the data processing and gesture classification code), and online databases (to create an online real-time dataset), statistics to calculate features, data processing to order, clean and organise the incoming signal values and, Artificial Intelligence and Machine Learning for creating the classification models.

During the evolution of the project, the necessary information on the above-mentioned fields has been learned as needed. On the other hand, two courses in Artificial Intelligence (Elements of AI [1] and Building AI [2]) from the University of Helsinki have been done before starting this project to acquire basic knowledge about AI and ML.

2. Introduction

2.1. Project targets

Before starting working on this project, some main objectives were defined as goals to give this thesis a guideline. These objectives are:

- Record muscle activity from hand and wrist gestures with a surface electromyogram sensor and create an own dataset.
- Create an online real-time database to store the measures of hand and wrist gestures while they are being recorded, for a forward analysis and interpretation of it.
- Digitally pre-process and analyse the stored electromyographic data for a better result and comprehension.
- Design different Machine Learning hand-wrist gesture classification models to compare their efficiency and accuracy with the own dataset.
- To make a real-time evaluation of the most efficient Machine Learning model.

2.2. Project Scope

The final goal of the *Wearable Smart Therapeutic Glove* is to help patients and make it easier for the health professionals with rehabilitation procedures. This goal is being pursued through improved patient comfort when being tested, faster and more efficient collection of vital signs, and more effective and accurate data analysis to simplify the work of health professionals.

This report has two main parts: on one side, the electromyographic branch that includes all the hardware, firmware, and gestures measuring evolution and, on the other side, the Artificial Intelligence branch that explains all the information concerning the Machine Learning models.

2.3. Organisation

A Gantt diagram has been designed to keep a track of the evolution of this project, the time invested in each part of the project, and a general organisation as the two parts of the project needed to be carried out in parallel. This Gantt diagram is found in Attachment A1.

3. State of the Art

As a result of the fast evolution and development of Artificial Intelligence and its branches in the past decades, it is already considered a key factor in the development of companies, technology, cities, countries, and even society [3]. Therefore, it is understood that state-of-the-art utilities and applications of AI are what is seen reflected in society. **Computational Intelligence**, **Data Mining**, **Automatic Process Control,** or **Virtual Environments** are a few examples of areas where AI is developing fast due to its impact in the industry and its proven efficiency into optimising processes, tasks, environments, and workflow of companies [4].

Computational Intelligence uses techniques like **fuzzy logic**, which is very useful when it comes to solve imprecise or uncertain problems and we can find it working in aerospace altitude controllers or automobile speed control systems [5]; **artificial neural networks**, one of the most popular AI techniques and it is usually used for patterns recognition or solving other AI problems by learning from examples trying to replicate the human brain [6]; and **evolutionary computation**, a technique based on biological evolution that solves problems using, for example, genetic algorithms or swarm intelligence [7].

Data Mining combines Artificial Intelligence and statistics techniques with the goal of discovering patterns that help making predictions in a variety of fields. Some of those fields might be: finances, where it helps to predict loan payments or determine credit ratings [8]; the telecommunication or credit card industries, which use this tool to detect fraudulent use of their services; or the medical field, where it can be used to predict how efficient tests, surgical procedures or some medications might be [9].

Automatic Process Control is the most used area when it comes to factories and processing plants with a variety of duties involving energy-transfer operations, chemical conversion, storage (*See Figure 3.1.*), filling, packaging, etc. [10]. It must be said that this is not a pure AI technique but it can use, and every time uses more, these techniques in some schemes. Tools like system identification, modelling or optimization are a few examples of AI helping Automatic Processes.

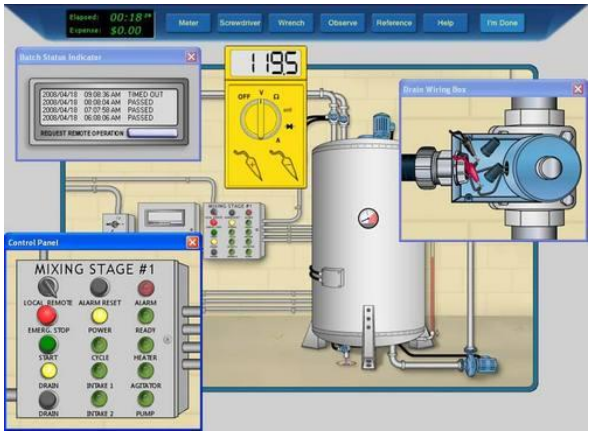


Figure 3. 1.- Example of an Automatic Process Control panel.
Source: indiamart.com

Virtual Environments are being used by companies to replicate remote operations while providing technical support and interpretation. It is very used in the gas and oil industries offering training simulators for operators*, See Figure 3.2.*, support for remote drilling operations or geological interpretation, among many other utilities.

Figure 3. 2.- Example of Virtual Environment applied to the industry. *Source: blog.isa.org*

The branch of Machine Learning is in constant development and has new advances every year which reflect in modern technologies and methodologies in scientific and non-scientific sectors. The areas with more constant improve in Machine Learning (ML) are **computer vision**, **natural language processing** and **medicine**.

In **computer science**, the last approaches have been in **semantic or image segmentation**, its goal is to group parts of an image which correspond to a same object or class with a pixel-level prediction accuracy*, See Figure 3.3.* [11]; **image classification**, has the goal of assigning images to specific labels, divided into categories. This are only pretending to classify images with one object on them. Due to it will later be used as a database for object detection [12]; and **object detection**, combines image segmentation and image classification. Its goal is to first segment different classes of an image and later detecting which, previously registered, object it belongs to*, See Figure 3.3.* [13].

Figure 3. 3.- Examples of image segmentation (left) and object detection (right). *Source: paperswithcode.com*

In **natural language processing**, Machine Learning methods have been developing in fields like **language modelling**, is used for predicting the next word or character that will be typed in a document (predictive writing). It can be later applied into text generation or classification [14]; **machine translations**, these systems translate a written sentence in a source language into a goal language, using statistical or neural-based methods [15]; or **question answering**, as its name says, the task is to answer questions from texts. It can be segmented into different domains depending on what result is wished [16].

When it comes to **medicine**, the last approaches haven been related with **drug discovery**, created models trying to discover new drugs by analysing all the molecular components of it and running tests while searching for affinities and, even, generating 3D molecular designs [17]; and **medical diagnosis**, its goal is to identify or detect illnesses or diseases from patients. These methods are based on risk factors, symptoms, test results, etc. [18].

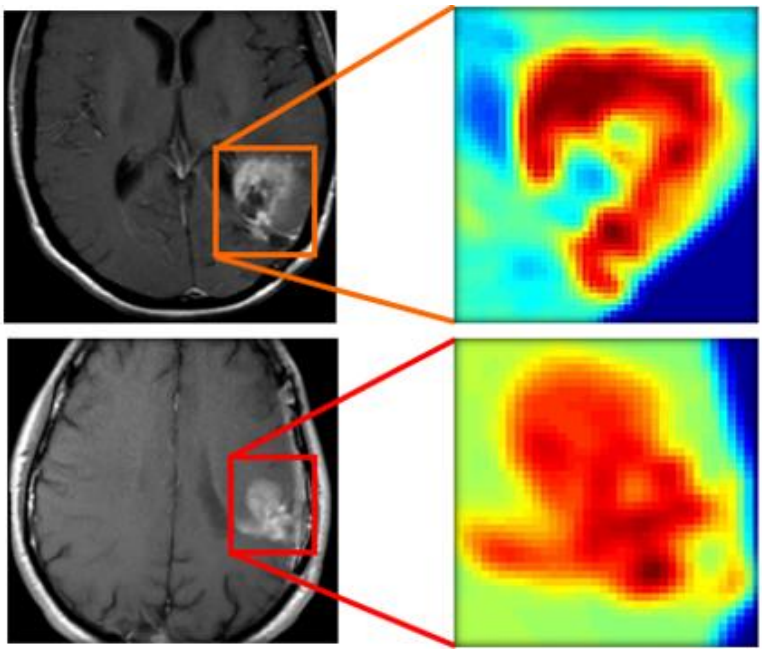


Figure 3. 4.- Example of Machine Learning applied to medical diagnosis. *Source: medium.com*

Artificial Intelligence and Machine Learning can be combined with all the labour sectors, not even industry, and the results of it are reflected in major advances, not only in technology but also in efficiency and comfort. This report combines Machine Learning (branch of AI) with Electromyography to interpret and classify gestures made with the hand and wrist.

The obtained signals after measuring with an electromyogram (EMG) have applications in many different fields, such as clinical or biomedical applications (i.e., detect isometric muscular activity), the gaming industry*, See Figure 3.5.,* (i.e., interactive computer games, which may allow the game to

have access to the player's emotional/physical state) or communicative applications (i.e., mime speech recognition via observing the activity of muscles related to the speech) [19].

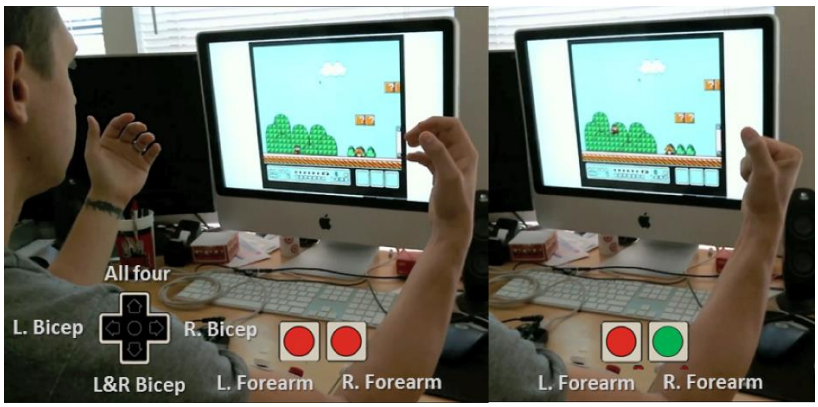


Figure 3. 5.- Example of EMG applied to the gaming industry. *Source: instructables.com*

Some examples of the use of EMG signals combined with Artificial Intelligence related with the field of this project are a "*Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion*"*, See Figure 3.6.,* [20] or "*Electromyogram-Based Classification of Hand and Finger Gestures Using Artificial Neural Networks*" [21].

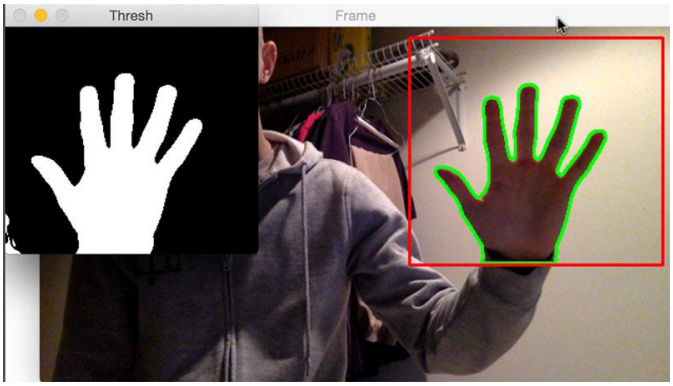


Figure 3. 6.- Hand gesture recognition via computer vision. *Source: wordpress.com*

With all these examples, among many others, it has been shown the relation and level of involvement that EMG signals and AI, especially Machine Learning, can have and all the research that has been taking place over the last few years involving health and technology with the goal of preventing, analysing and classifying diseases or illnesses in a faster and more efficient way.

4. Electromyography (EMG)

The following section contains all the required information to understand what and electromyogram is and how it works, measures the muscle activity and converts that data into a digital value. Besides that, this section shows the components and the assembly of this sensor for its optimal performance in terms of measuring these EMG signals as well as the coding part concerning the measure of data. At the end, it is explained the process of measuring and recording the data, as well as the gestures that will be recorded.

To have a clear understanding of the section, it has been divided into three blocks, hardware, firmware and measuring. Although all parts are connected, each of them has different research and development resources and procedures making it more clarifying explaining them separately.

4.1. Hardware

The hardware branch involves all the physical parts of this project and that is exactly what will be explained. First, a little bit of general research to have a better understanding of the EMG, how it works, diverse ways of carrying out the test, the signal that is obtained and the conversion of the results into analogical values. After that, all the components that have been used for the assembly will be mentioned and some relevant information will be given and last of all, a section which explains how the final assembly has been reached step by step.

4.1.1. Background Research

Electromyography (EMG) is the study of the electrical signals of muscles, which takes to the electromyogram, a medical diagnostic test that evaluates the health and state of muscles and the nerves that control them, by recording the movement of the muscles. These measures can be recorded due to a little electrical burst that takes place once the muscle is contracted [22]. This procedure takes place to find muscle and/or nerve dysfunctions, disorders or injuries.

The two most used ways of performing this test are, on one side the medical procedure, *See Figure 4.1,* which takes place in hospitals and is carried out by doctors or medical staff. To perform this test, firstly, electrodes must be placed on the skin surface to record the nerves functioning. After that, some low electrical pulses are applied to check if the nerves can carry those impulses to the electrodes. Once this "checking" is completed, a tiny needle with an electrode on its surface will be inserted into the muscle to evaluate the activity of the muscle by contracting it, as mentioned before. The following step is to visualise the activity of the muscle in a monitor, which is connected by a cable

to the needle. The last step is to remove the needle from the body of the patient [23]. This procedure is considered safe and harmless, the patient may only feel some tickling or small pain while the needle is introduced in the skin [24].

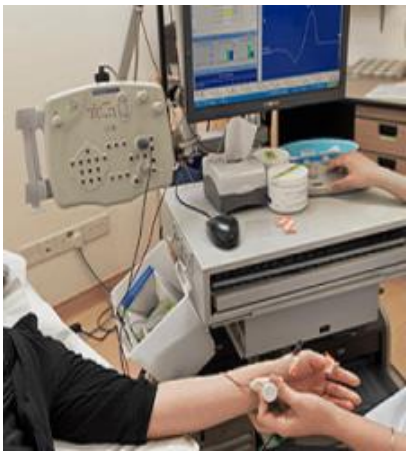


Figure 4.1.- Medial EMG test. *Source: zadpathlabs.com*

As it can be seen in *Figure 4.1*, the procedure carried out in hospitals requires a big infrastructure with a big monitor and controller, which is not a very efficient technique if the space and mobility while performing the test want to be optimised.

On the other side, there is the surface EMG (sEMG) method, *See Figure 4.2*. This other procedure requires much less space and has a lot of opportunities for performing the test anywhere. It is especially useful when it comes to long term monitoring of the muscular activity of a patient or for research. This technique uses an EMG sensor, *See Figure 4.3*, which only requires three electrodes and the connection to a microcontroller to measure the muscle activity with no need of using needles and big controllers. The measuring procedure is even more simple than the one explained before. On one side, the electrodes will be placed on the skin surface and connected to the sensor. On the other side, the sensor will be connected to the microcontroller which will run a code to measure the muscle activity. This test records the activity of muscles from the surface of the skin, without having to penetrate the muscle.

Figure 4.2.- Surface EMG method. *Source: www.medicalexpo.com*

Figure 4.3.- EMG sensor. *Source: ultra-lab.net*

Once the test is performed, the results must be analysed and for it, some previous filtering steps need to be executed with the goal of having a cleaner signal, which permits an easier classification and differentiation from other ones.

The first signal that is obtained after the measuring is a plot with the raw data of the taken measures, *See Figure 4.4*. This signal tends to contain noise, which may have various sources such as, the inherent noise originating from the electrodes, the movement of the cables connected to the sensor, electromagnetic noise or even internal noise [25].

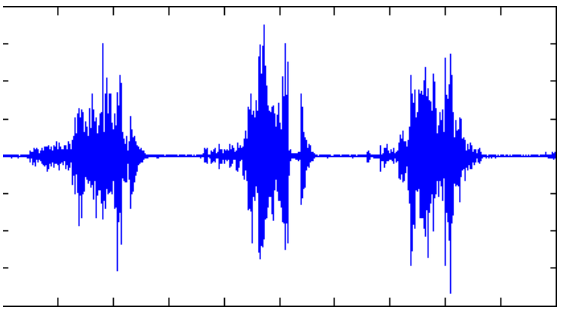


Figure 4.4.- Raw data of an EMG measure (mV/seconds). *Source: www.researchgate.net*

After having recorded and obtained the raw signal, the processing of the signals needs to be done. For it, nowadays, there are different techniques to execute this process. Some of the most used and effective methods for signal processing are explained below.

The Wavelet Analysis, which uses the Wavelet Transform (WT) is a mathematical tool that analyses the non-stationary and the fast transient signals. This method is especially useful for signal processing and for EMG classification as it can decompose patterns "hidden" in data. There are two types of Wavelet Transform, on one side the Continuous WT (*equation 4.1*) and on the other side the Discrete WT (*equation 4.2*). The first one uses all the wavelets over a range of scale and location meanwhile the second one uses a specific number of wavelets [26].

$$T(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \psi^* \frac{(t-b)}{a} dt \qquad \textbf{(Eq. 4.1.)}$$

$$T_{m,n} = \int_{-\infty}^{\infty} x(t) \psi_{m,n}(t) dt \qquad \textbf{(Eq. 4.2.)}$$

Higher Order Statistics (HOS) is a statistical method that analyses beyond lower order statistics as, which use constant or quadratic terms. It can model nonlinear phenomena or identify suitable models for nonlinear systems. Furthermore, this technique helps to obtain phase information, which could not be obtained with second order statistics [27].

Empirical Mode Decomposition (EMD) This procedure is comparable to the Fourier Transforms or the Wavelet Transform, due to it breaking down a signal without leaving the time domain. It can be extremely helpful when it comes to analyse non-linear natural signals [28].

Independent Component Analysis (ICA) this method reveals hidden factors that underlie sets of random measures or signals. This statistical and computer science model is used to analyse data given as a big database of measures or samples. The techniques assume the data variables as linear mixtures of an unknown set of latent variables and mixing system. It is a technique with a lot of potential to find those underlying variables or factors when the classical methods fail [29].

Any of these signal processing techniques are in daily use in the world of research and development. So, although some techniques may have a higher percentage of accuracy than others in some areas, any of the previously mentioned techniques could be chosen and would perform well in this project.

4.1.2. Components

The sEMG sensor that will be used in this project is the PLUX BiTalino EMG UC-E6 version, *See Figure 4.5.* This version includes 2 UC-E6 sockets on both sides, one for the measuring cable and the other for connecting to a BiTalino Core. This sensor has an output range of ±1.64 mV (with VCC = 3V) and a Bandwidth between 25-480 Hz. More specifications can be found in its datasheet [30].

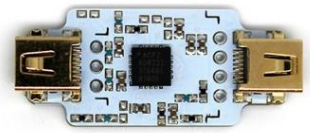


Figure 4.5.- BiTalino EMG UC-E6. *Source: bitalino.com*

This sensor records data in analogical values, to convert this value into a digital one, its transfer function will be used (equation 4.3) to obtain the signal in volts (V).

$$EMG(V) = \frac{\left(\frac{ADC}{2^n} - \frac{1}{2}\right) * VCC}{G_{EMG}} \qquad \textbf{(Eq. 4.3)}$$

All the values of the *equation 4.3* are given in the sensor's datasheet except on two of them: the ADC and 2^n. The ADC value is the one that is being read by the sensor (the analogical value), meanwhile *n* is for the number of bits of the channel, which depends on the resolution of the Analog-Digital Converter (ADC). In this case *n* is 8, due to that is the resolution of the ADC included in the ESP32 microcontroller.

In *Figure 4.6*, the utility of each of the pins of the sensor can be appreciated. On the left side, there is the socket connected to the measuring cable, *See Figure 4.7*, this cable has three electrodes with

three different sleeve colours: red, black and white. Each colour has its function and meaning, *See Table 4.1*. And on the right side goes the socket that connects to any of the products of the same brand. Although, using any of those products would have been extremely easy to use, this project needed to be done with the ESP32 microcontroller. As the sensor and the controller are not compatible via cable, the right side had to be modified and, instead of using the socket, use cables and solder them to the microcontroller (this process is explained in the next subsection).

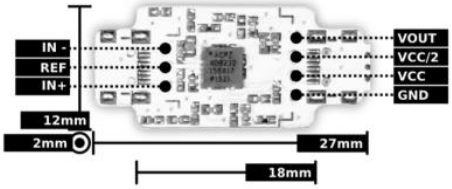


Figure 4.6.- Physical characteristics of the EMG. *Source: bitalino.com*

Figure 4.7.- 3-lead UC-E6 cable (for measuring). Source: bitalino.com

Figure 4.8.- ESP32-WROOM-32 microcontroller. *Source: aaloktech.in*

Table 4.1.- Meaning of the sleeves for the UC-E6 electrode cable.

Sleeve Colour	**Red**	**Black**	**White**
3-lead electrode cable	**IN+**	**IN-**	**REFERENCE**

As mentioned before, the microcontroller that has been used for this project is the ESP32-WROOM-32, *See Figure 4.8*. There is no special microcontroller requirement for this project, the ESP32 is the one that was available for in the laboratory and that is why it was used, but any other microcontroller with a Wi-Fi module and compatibility with Firebase could be used.

This module incorporates alongside the Wi-Fi a Bluetooth module, making the controller capable of executing many different applications such as, connecting to a Wi-Fi router through the Wi-Fi module or connecting to any device with Bluetooth (smart phones, peripherals, smart TVs, etc.) [31].

This controller has 38 pins and its distribution appears in *Figure 4.9*. As it can be seen in the figure, the microcontroller has many functions and its detailed information is found in its datasheet [31]. For this project only a few pins such as the GND or the 3.3V have been needed.

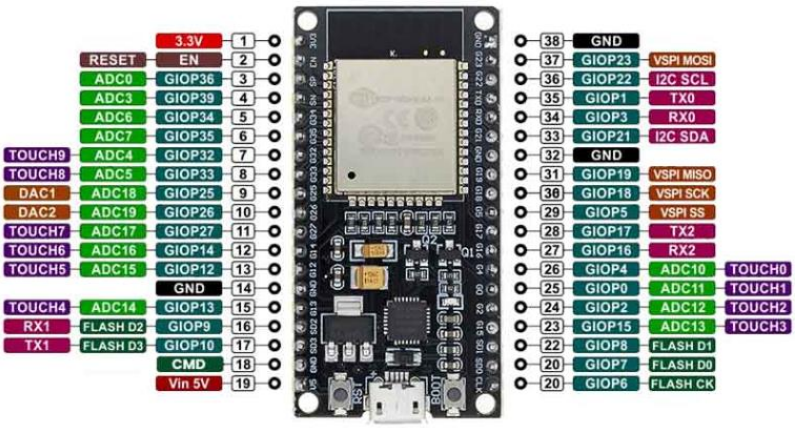


Figure 4.9.- Pin distribution of the ESP32-WROOM-32. *Source: uelectronics.com*

The rest of components are simple complements to the assembly of the microcontroller and the sensor. These components are: cables, resistors, protoboard, soldering tin and iron, electrodes for the skin and a laptop to control the process, feed the microcontroller and upload the firmware.

4.1.3. Evolution of the assembly

The final goal of this project is to design an algorithm capable of preventing hand, wrist and arm illnesses. To achieve that, the first step is to record, analyse and classify the signals from the sEMG, which is the goal of this thesis.

The assembly has evolved over time, as obstacles have been encountered or changes and readjustments needed to be made. During this section all the evolution of the assembly will be seen, from the very beginning, when it only had the sensor to the very end, when the assembly is ready to classify gestures.

The first thing that had to be done once the sensor arrived was learning and checking how the sensor does the reading and recording of the signal without having the BiTalino tools. The very first attempt was connecting an UC-E6 to USB cable and trying to record and measure directly from there. After getting no response, a second idea came to mind: soldering cables from the sEMG to the microcontroller.

So, a cable was soldered to each of the sensor connectors, *See Figure 4.10*, and then connected those cables to a protoboard where a simple tensor divider was built to get a correct REF, due to$V_{REF} =$ $= {V_{cc}}/{2}$ and then connected to the microcontroller, *See Figure 4.11*.

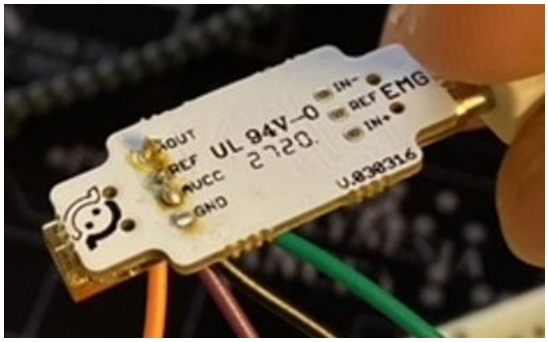


Figure 4.10.- Soldered cables to the sEMG. *Source: own source.*

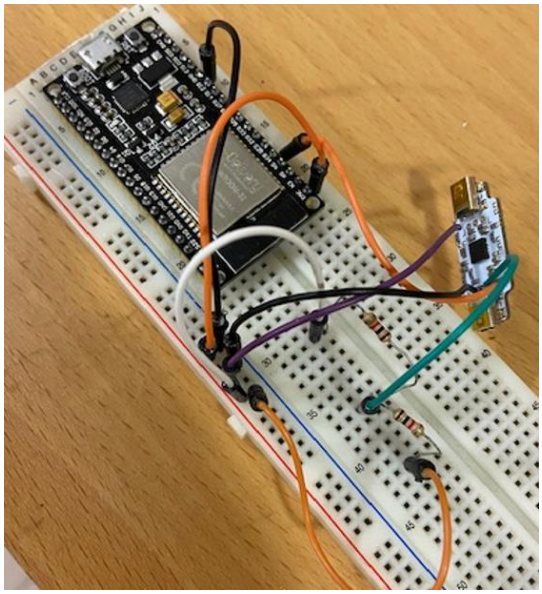


Figure 4.11.- First assembly on a protoboard. *Source: own source.*

In *Figure 4.12* the scheme of the circuit assembled before is shown. It is worth noting that the resistance size of both resistors is not relevant as it is a tensor divider, the important thing is that they both have the same value. Another point is that the three connectors that can be seen on the left side of the sEMG, is the socket where the, earlier mentioned, cable with the electrodes goes connected. For the signal reading, pin 34 was used, but whichever pin wished, if it works as an ADC, could be used.

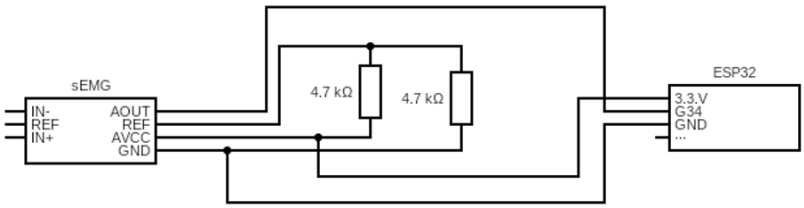


Figure 4.12.- Circuit scheme of the first assembly. *Source: own source.*

Once this assembly was built, some experimental measures were taken to see which results were obtained and how to face the next steps. After correcting some small firmware errors, the signals were going to be measured and recorded, but there was still a problem. This structure was unstable and the constant shaking of the cables both on the breadboard and on the sEMG was affecting the measures. To solve these stability issues, the soldered cables were replaced with some fixed male pin strip, *See Figure 4.13*.

Figure 4.13.- New soldering to the sEMG. *Source: own source.*

After having soldered the pins to the sEMG, all the assembly was moved from the breadboard to a prototyping board, *See Figure 4.14*, to make the whole assembly more stable once it was confirmed that it was working properly.

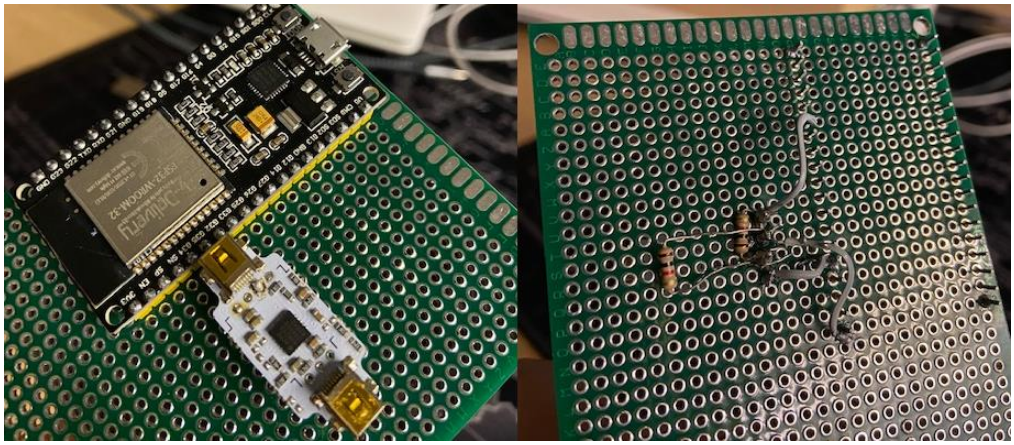


Figure 4.14.- Assembly soldered in a prototyping board. Top (left side) and the bottom (right side). *Source: own source.*

After finishing this second assembly, it was tested again by recording some example measures and the signal that was being obtained this time was much better. Also, without those cable movements mentioned before, the signal result was not having any interruption or non-sense values. Although it was working well, the sensor was still not completely stable, as the pins were only soldered on one side, at the bottom. Ideally, the pin should have been inserted through the holes of the sEMG and soldered so that it would have been stable but the diameter of the used pins was bigger than the diameter of the holes in the sensor.

After getting thinner male pin strips that could get through the sensor holes, they were soldered again as wished since the beginning, *See Figure 4.15*. A notable mention is that while removing the old soldering, a piece of rack leading to the AVCC had fallen off. It could be repaired by soldering a small piece of cable in the broken part, as if it were a bridge.

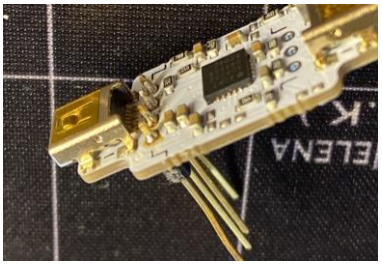


Figure 4.15.- New male pin strips soldered to the sEMG. *Source: own source.*

Besides this new soldering on the sEMG, a button was implemented in the assembly, *See Figure 4.16* and *Figure 4.17*, with the goal of simplifying the measuring process. The idea behind implementing this button was to be able to start and stop the recording of a measure by pressing this button, creating an interruption and moving to the next gesture measure. Theoretically, this should have been the final assembly of the project.

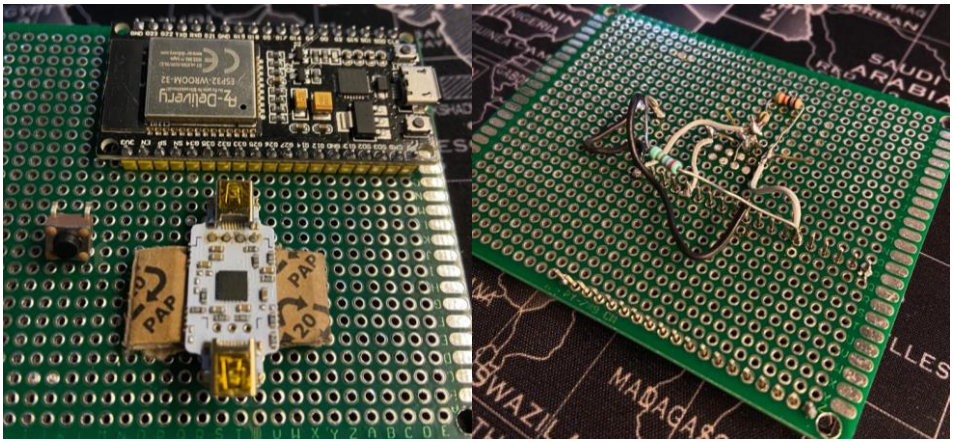


Figure 4.16.- Second assembly with new sEMG soldering and button. Top (left) and bottom (right). *Source: own source.*

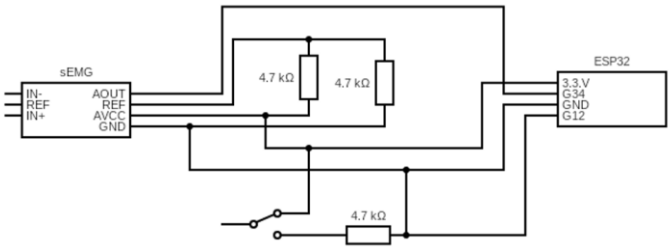


Figure 4.17.- Circuit scheme of the last assembly with the button for the measuring. *Source: own source.*

While testing this last assembly, there appeared a few execution errors with the implementation of the button and the interruptions. The problem was not due to the assembly, because all connections were checked twice, and all components and cables were well soldered and perfectly working. The problem was with the firmware, there were code errors that needed to be solved.

After considering several new assembly options and trying to have the firmware code errors fixed, the final assembly model had been chosen. The final assembly would look like the second assembly, but without the button, like in *Figure 4.14*. This decision had been taken due to a cleaner firmware code being written and it worked automatically, so there was no need to implement the button.

The decision of replacing the button will be done after having finished recording all the measures due to the high sensitivity of the assembly after having damaged the sensor and, wasting 3 weeks by ordering a new sensor was not a luxury that could be afforded.

4.2. **Firmware**

Firmware is the permanent software programmed into a read-only memory, in this case the ESP32 microcontroller; another definition might be an embedded software in and for a hardware. In

our case, the code that will read and record the measured data is firmware. The following subsections will explain which platform has been used to write and send this code, as well as the explanation of that code itself.

4.2.1. Background Research

Whenever any code for a microcontroller must be written, the firmware to be used depends on the microcontroller itself. As the ESP32 has been used for this thesis, the best compatible Firmware is Arduino, and it is the one that has been used for it.

The main reasons why I used Arduino for this section are, on one hand, because I have experience coding with it, because it is easy to use and due to its many libraries, that allow a wide range of different applications.

The development environment that uses Arduino is the Arduino IDE, a multiplatform application written in Java language which also admits languages like C or C++. The way it works is easy, once the code is written, it just must be charged into the microcontroller, and it will be ready to execute.

A library is a file which has been written in C or C++ language and allows using extra functions for a more specific application in addition to the functions that Arduino already has [32]. To be able to use these libraries, they must be installed before anything and, once installed, then they can be used in the code. For using them they must be called at the beginning of the code with an ***#include*** *<LibraryName.h>* command. The libraries used for this Firmware section are Wi-Fi and FirebaseESP32. The following subsections will explain what each of these libraries is used for and which functions have been used.

For more information about how the Arduino IDE works or how to get started into coding in such an environment, please check [33] and [34].

The last thing that had to be searched for was a way of sending this data automatically to the cloud so it could later be imported from another environment. For doing that, the best solution was using an online database, but in addition it had to be a real-time database because the recorded data to be sent for doing real-time classification in the future. The perfect match for that was Firebase, a free online platform for web and phone applications development and with many extra functions. One of those extra functions is a real-time online database, which is perfect for our needs. Furthermore, this database is perfectly compatible with Arduino and Python, the two software that are being used in this thesis.

4.2.2. Firmware code evolution

To get to the final Arduino code, there have been many code updates and changes that have been taking place alongside both the evolution of the assembly and the testing part. This subsection will scroll over all code updates until reaching the final code, explained in the next subsection.

The first code was created once the sEMG arrived and for that (see Attachment B1), a simple signal-reading firmware code was written, *See Figure 4.18*. That code creates a variable, *SensorPin*, and links it to the pin 34 from the microcontroller. After that a loop is executed, in that loop the sEMG analogical value is read and converted into a digital value. Then this digital value is printed on screen and the loop starts again.

```
int sensorPin = 34;

void setup() {
  Serial.begin(115200);
}

void loop(){
  float sensorValue = analogRead(sensorPin);
  delay(50);
  float voltage = ((((sensorValue/4095)-0.5)*3.3)/1009)*1000;
  Serial.println(voltage);
  }
```

Figure 4.18.- Code to measure sEMG values. *Source: own source.*

With this, it was pretended to visualize the response signal from the sEMG, its measure range, precision and resolution to set the bases of what had to be coded in advance and how the measuring process would be structured.

In the next design, *See Figure 4.19*, the WiFi.h and FirebaseESP32.h libraries were implemented, see Attachment B2. With this, the microcontroller could already connect to the Wi-Fi network and to the Firebase real-time database. What changes in this code version is the setup, the loop still works the same way as in the previous version. This time, the code connects firstly to the Wi-Fi network (previously, the user must introduce the network SSID and password) and once it is connected to the network it connects to the Firebase database (previously, the user must introduce the database URL and secret key). After the successful connection, an Okey message is sent to the Database, to a checkout node. After sending this text, the loop starts and reads, converts and shows the signal as explained before. The way these two libraries operate and all the information, regarding them, needed to know is precisely detailed in the following subsection.

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include "Secret_keys.h"

int sensorPin = 34;
const char* ssid = WIFI_SSID;
const char* password = WIFI_PASSWORD;
const char* host = FIREBASE_HOST;
const char* auth = FIREBASE_AUTH;

FirebaseData firebaseData;

void setup() {
  Serial.begin(115200);
  Serial.println();
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  while (WiFi.status() != WL_CONNECTED){
    Serial.print(".");
    delay(300);
    }
  Serial.println("WiFi succesfully connected!");

  Firebase.begin(host, auth);
  Firebase.reconnectWiFi(true);
  Firebase.setInt(firebaseData,"/EMGdata/Checkout","Okey");
```

Figure 4.19.- Code to test the libraries and Firebase. *Source: own source.*

After testing the previous code and verify that everything was working well, a new assembly was built. This new assembly had the goal of implementing a button to make and easier measuring process where, whenever the button was pressed, a new gesture was recorded. As this idea was finally refused, the code was never tested and it was not taken into consideration.

The next code was already the final and definitive code, which is explained in the following section.

4.2.3. Final firmware code

This last section pretends to explain the final code written to measure, record and send de sEMG signals from the sensor to the online real-time database and storing them there. The most relevant parts of the code are explained in subsections after an overviewed explanation of its functioning.

Once the code is loaded in the microcontroller, it connects to the Wi-Fi network that the user previously defined. While it connects to the network, a message will appear on screen to let the user know that it is trying to connect and it also notifies once it is connected. After that, it connects to the real-time database. The next order on being executed is reading the value of a variable from the database, *Measure_patients*, increasing by one its value and sending the updated value back to the database. This variable is an integer variable and its function is, on one side to keep a recount of the users that have been measured so far and, on the other side to make a little correction in the Python code further on.

Afterward, a loop that will iterate five times, once per gesture, starts. Firstly, a few messages are printed, letting the patient know that the measuring is starting and to prepare the first gesture and right after that a second loop begins. This second loop performs the measuring by reading the value that the sEMG is measuring, converting it into a digital value and sending it to the database, this loop repeats 140 times, time enough to perform the gesture twice. After these 140 iterations a message appears, letting the patient know that the gesture measuring is over and to prepare for the next gesture. Then, the process repeats for 4 more times. Finally, after having measured the 5 gestures, an acknowledgement and goodbye message appear ending both the measuring process and the code.

Besides the code explanation, the entire Arduino code with all the details can be found in Attachment B3.

4.2.3.1. Initializing

First, the libraries are called and any variable to be used during the code is defined, *See Figure 4.20*. As mentioned before, two libraries are being used in this code: Wi-Fi and FirebaseESP32.

```
//Calling the libraries
#include <WiFi.h>
#include <FirebaseESP32.h>
#include "Secret_keys.h"

//Declaring all variables
int sensorPin = 34;
int j=1;
int i=0;

const char* ssid = WIFI_SSID;
const char* password = WIFI_PASSWORD;
const char* host = FIREBASE_HOST;
const char* auth = FIREBASE_AUTH;

//Defining Firebase Data object
FirebaseData firebaseData;
```

Figure 4.20.- Initializing the firmware code. *Source: own source.*

The Wi-Fi library has the function of enabling the microcontroller to get connected to the Wi-Fi network. To execute this order, the SSID and the password of the network will have to be written down. To keep that information a bit more private and to be able to save different data for different networks, a header file (***secret_keys.h*** check Attachment B4), *See Figure 4.21*, has been created with all this information. To link this file to the code, it just needs to be called as a library. For more information about the Wi-Fi library and its classes, check [35].

```
//WIFI Credentials
//---------------------------------------
//#define WIFI_SSID "[redacted]"
//#define WIFI_PASSWORD "[redacted]"
#define WIFI_SSID "[redacted]"
#define WIFI_PASSWORD "[redacted]"

//Firebase Real-time Database Credentials
//---------------------------------------------------------------------------------------------
#define FIREBASE_HOST "https://aehrich-database-default-rtdb.europe-west1.firebasedatabase.app"
#define FIREBASE_AUTH "[redacted]"
```

Figure 4.21.- Content *of Secret_keys.h*. *Source: own source.*

The other library, FirebaseESP32, is used to connect the microcontroller and to execute all the commands related to sending the recorded data to the Firebase database. The first step to reach that connection is to insert the URL and the secret key of the database, those values can be found in specific places on the database page, *See Figure 4.22*. That information is written and stored in the file ***secret_keys.h****, See Figure 4.21,* mentioned before, next to the Wi-Fi information. For more information about how to get those values and get started into Firebase, check [36]. And for more information about the FirebaseESP32 library and its classes, check [37].

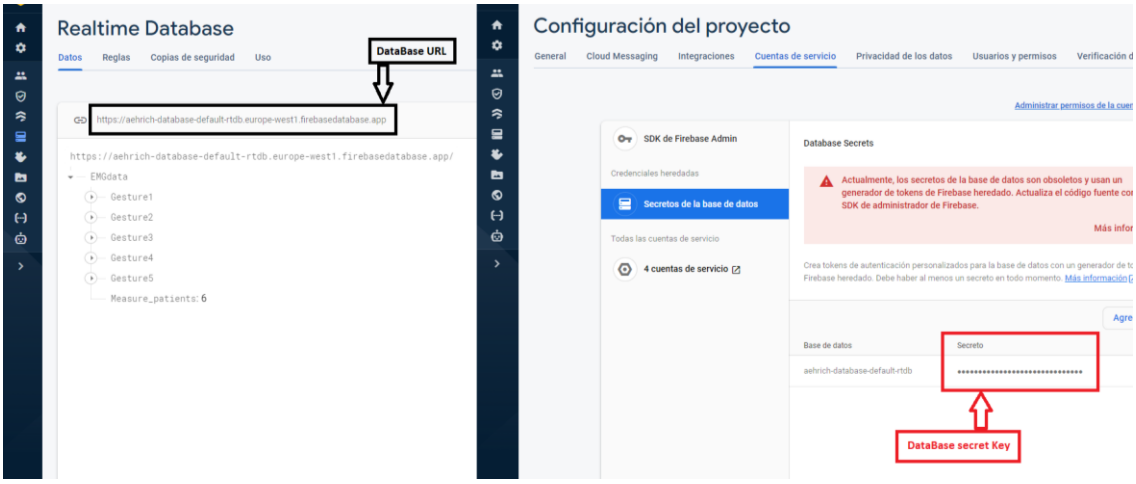


Figure 4.22.- Where to find the Database URL and Database secret Key. *Source: own source.*

The last command defines the database data object. This object will appear in every command related to Firebase.

4.2.3.2. Connecting to Wi-Fi and Firebase

Once the process is initialized, it connects to the Wi-Fi network and to the Firebase real-time database, *See Figure 4.23*. The first to be executed is the connection to the Wi-Fi network, the microcontroller reads the SSID and password that are stored in *Secret_keys.h* and it executes the connecting process. It also has an option, in the last line of the figure, that allows the microcontroller to automatically reconnect to the network in case it gets disconnected from it at any moment.

```
WiFi.begin(ssid, password);   //Reads the Wi-Fi network data stored in Secret_keys.h
Serial.print("Connecting to WiFi");
while (WiFi.status() != WL_CONNECTED){
  Serial.print(".");
  delay(300);
}

Serial.println("WiFi succesfully connected!");

Firebase.begin(host, auth); //Reads the Firebase information stored in Secret_keys.h
Firebase.reconnectWiFi(true); //In case the connexion breaks, it will try to reconnect automatically
```

Figure 4.23.- Connection to Wi-Fi Network and Firebase database. *Source: own source.*

After connecting to the Wi-Fi network, it connects to Firebase by using the same procedure as the Wi-Fi but, in this case, reading the database credentials from the same other file.

4.2.3.3. Reading data from the sensor

Once the microcontroller is connected to both Wi-Fi and Firebase, it starts reading data from the sEMG*, See Figure 4.24,* in an analogical value and storing it into a variable called *sensorValue*. After reading the data, this data is converted into a digital value, applying the conversion equation (*eq. 4.3*) and storing it into a new variable *voltage*. This new variable is a float, due to it having to read decimal numbers.

```
float sensorValue = analogRead(sensorPin);  //Reads the analogical value from the variable SensorPin
delay(30);
float voltage = ((((sensorValue/4095)-0.5)*3.3)/1009)*1000; //Converts analog value into digital
```

Figure 4.24.- Reading data from the sensor. *Source: own source.*

4.2.3.4. Sending the data to Firebase

The last step is sending the read data to the online real-time database*, See Figure 4.25*. For it, the command push is being used. What this order does is adding a value, into a specific path, to the end of the database. Besides that, there has been added an order that in case the action of sending the data to Firebase does not get executed or gets interrupted, a message will appear on the screen letting the user know that there was an error and which type of error it was.

```
if (Firebase.pushFloat(firebaseData, "/EMGdata/Gesture"+String(j), voltage)){
} //Send the digital value to firebase
else{
  Serial.println("Failed");
  Serial.println("REASON: " + firebaseData.errorReason());  //In case it does not send, it will show the error
```

Figure 4.25.- Sending data to the online real-time database. *Source: own source.*

4.2.4. Code for the evaluation of the model

This last section, shows and explains the firmware code written to create a real-time evaluation test of the chosen Machine Learning model. All the relevant information about the model and ML is explained in its correspondent sections, as this sub-section will only explain the parts regarding the firmware.

This code is a cut version of the measuring code, its goals is to record only one measure and send it to a branch in the Firebase online database called *evaluation*. This code starts the same way as all the codes, it firstly connects to the Wi-Fi network and, after that, it gets linked with the Firebase database. Then a loop starts*, See Figure 4.26,* which record the analogical values from the sensor, converts them to a digital value and sends them to the branch. Once all the values are recorded, the program ends.

The entire code can be checked on Attachment B5.

```
int i=0;

while (i<100){
  float sensorValue = analogRead(sensorPin);
  delay(30);
  float voltage = ((((sensorValue/4095)-0.5)*3.3)/1009)*1000;
  Serial.println(voltage);

  if (Firebase.pushFloat(firebaseData, "/EMGdata/evaluation", voltage)){
  }
  else{
    Serial.println("Failed");
    Serial.println("REASON: " + firebaseData.errorReason());
  }
  i++;
}
```

Figure 4.26.- Loop for reading and sending data. *Source: own source.*

4.3. **Measuring**

This last section of the Electromyography block explains and shows which gestures are chosen to measure, how those gestures have been recorded, showing snapshots of the process and the screen, and how this data has been acquired.

4.3.1. Hand-wrist gestures

The first thing needed to be done is defining which gestures will be measured and why. For that, the *MAXSENS Smart Wearable Therapeutic Glove, See Figure 4.27,* will be now introduced. This IoT solution pretends to prevent and protect upper limbs disorders and injuries from working risks [38]. This ongoing project that combines AI and medical databases is the reason for this thesis. The contribution of this project is to start with a Machine Learning gestures classification with the long-term goal of creating an algorithm capable of preventing illnesses and/or injuries.

So, if the goal is to prevent upper limb injuries and illnesses, the best option is to choose the hand-wrist gestures that are more related with those injuries [39].

Figure 4.27.- MAXSENS Smart Glove. *Source: maxsens.es*

The most used gestures are often the ones that tend to cause most injuries. So, a set of 4 wrist and hand gestures commonly used in daily living plus a neutral wrist-relaxed gesture have been chosen. This sensor has only one channel, so its electrodes can only be placed on one muscle and, therefore, measure and record the activity of that muscle. It has been tried to choose gestures that have a primary relation with the muscle the sensor's electrodes will be placed in, but as hand-wrist gestures use many muscles at the same time, not all the gestures will use the same muscles.

The two main muscles that deal with wrist gestures are: radial carpal flexor [40] and the radial carpal extensor. To decide which muscle would be used for this project, both were tested and the one that had better results would be the chosen ones.

The test consisted of recording the sEMG signal in each muscle while performing a few gestures and the muscle with a better visual response would be chosen. Firstly, the radial carpal flexor muscle was tested.

Once the electrodes have been placed in the skin surface over the radial carpal flexor muscle, *See Figure 4.28,* the signal recording starts while the five gestures are performed. The gestures that have been chosen and will be recorded are: **neutral position**, *See Figure 4.29 (a)*: the hand and the wrist are relaxed lying on the surface of the table, experimenting no effort. **Extension of fingers**, *See Figure 4.29 (b)*: in this gesture the fingers are fully extended and abducted, with the hand palm facing up. **Flexion of fingers**, *See Figure 4.29 (c)*: the gesture consists of closing the fist with force, with the thumb encircling the outer surface of the rest of the fingers. **Flexion of wrist**, *See Figure 4.29 (d)*: the wrist is flexed by tilting the hand to the inner part with the fingers contracted and totally extended. And **pronation**, *See Figure 4.29 (e)*: the wrist is rotated inwards with flexed fingers. These gestures have been specially chosen to try to obtain a good result as most of them have a direct interaction with the measured muscle.

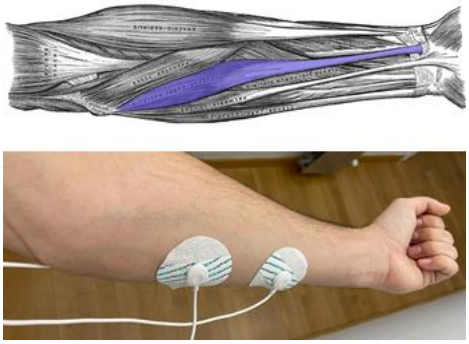


Figure 4.28.- Radial carpal flexor (top) and electrodes placement (down). *Source: Wikipedia.org and own source*

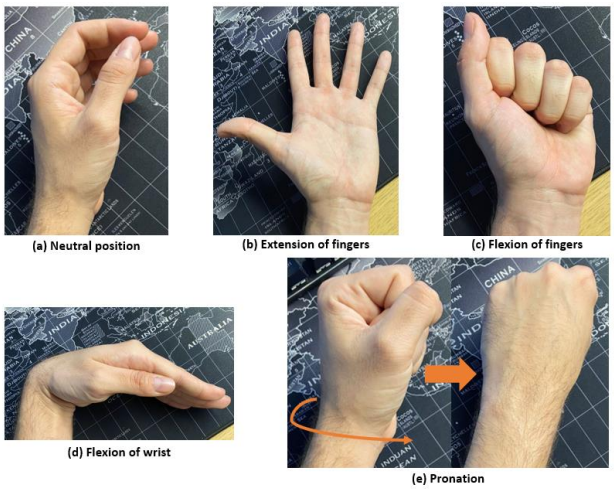


Figure 4.29.- Gestures to be measured for the first test. *Source: own source.*

The results obtained after recording all the signals*, Figure 4.30,* were not as good as expected. In the figure, five plots, one for each gesture, can be seen. It is also remarkable to mention that each gesture has been repeated three times and that each gesture has 100 data values, so each plot has a total of 300 data values.

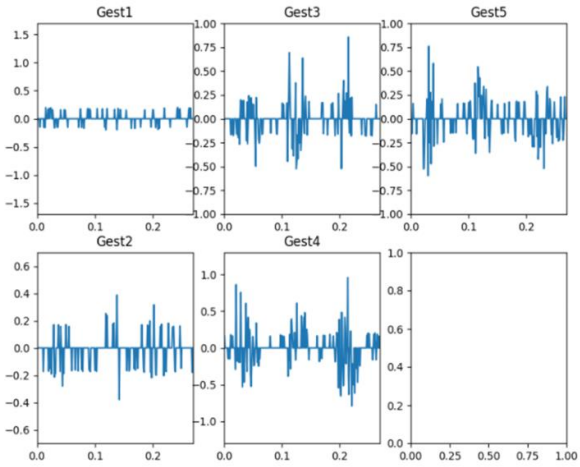


Figure 4.30.- Results of recording gestures in the radial carpal flexor (x-mV, y-sec.). *Source: own source.*

The second part of the test had the same procedure but, adapting some of the gestures to main function gestures of, this time, the radial carpal extensor. The electrodes were placed on the outside part of the arm*, Figure 4.31.* The gestures for this test are: **neutral position**, *See Figure 4.32 (a)*: the hand and the wrist are relaxed lying on the surface of the table, with no effort. **Extension of fingers**, *See Figure 4.32 (b)*: in this gesture the fingers are fully extended and abducted, with the hand palm facing up. **Flexion of fingers**, *See Figure 4.32 (c)*: the gesture consists of closing the fist with force, with the thumb encircling the outer surface of the rest of the fingers. **Extension of wrist**, *See Figure 4.32 (d)*: the wrist is extended by tilting the hand to the outer part with the fingers contracted and totally extended. And **ulnar deviation**, *See Figure 4.32 (e)*: the hand is rotated inwards with contracted and extended fingers.

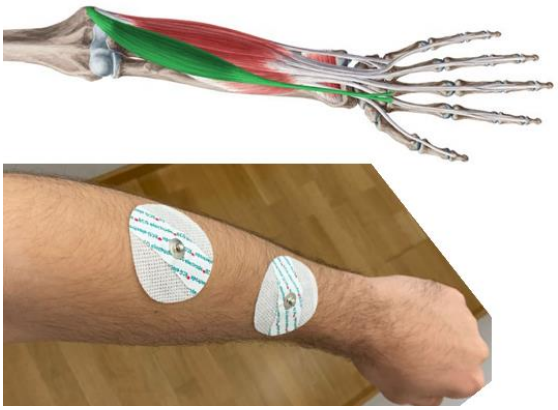


Figure 4.31.- Radial carpal extensor (top) and electrodes placement (down). *Source: kenhub.com and own source.*

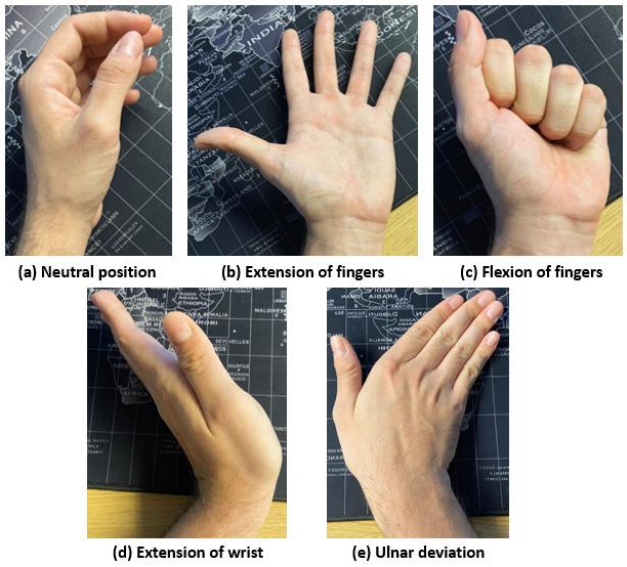


Figure 4.32.- Gestures to be measured for the second test. *Source: own source.*

After recording this set of gestures, the obtained results, *Figure 4.33,* were much better than the ones obtained in the first test. The signal was much clearer and the difference between executing the gesture and the resting were much more notable. In this case, the measured data values have been 200, so 2 repetitions per gesture.

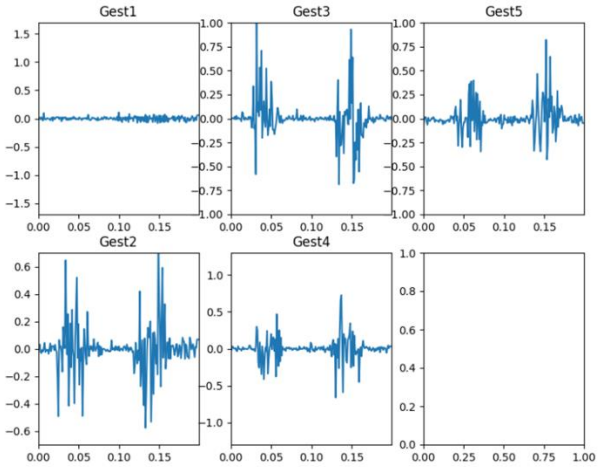


Figure 4.33.- Results of recording gestures in the radial carpal extensor (x-mV, y-sec.). *Source: own source.*

After having performed the test and comparing both results, it has been decided to start the measuring and recording to the patients to create the dataset, using the radial carpal extensor as the muscle where the electrodes of the sEMG will be placed. The already defined gestures will be the ones recorded.

4.3.2. Data acquisition procedure

The acquisition of the signals is the most important part of this thesis, because without a good dataset there is nothing reliable to be done. The test will only be executed three times per person, this way an excessive amount of repeated data that can affect the results is avoided. Furthermore, to be neutral, the same procedure and in the same order has been applied to all the patients. The following steps show the process of acquiring data from the patients:

0. Insert the local Wi-Fi and Firebase database credentials in the Arduino code, *See Figure 4.34*. That way the microcontroller can establish connection with the net and send the data to Firebase.

```
//WIFI Credentials
//---------------------------------------------------------------
#define WIFI_SSID "Example"
#define WIFI_PASSWORD "123test123"

//Firebase Real-time Database Credentials
//---------------------------------------------------------------
#define FIREBASE_HOST "https://example-db.firebasedatabase.app"
#define FIREBASE_AUTH "123invented_key123"
```

Figure 4.34.- Wi-Fi and Firebase credentials*. Source: own source.*

1. Stick the electrodes on the patient's arm surface, *See Figure 4.29*, and show the patient the gestures that he will have to do and ask him to repeat them. The patient must know that each gesture will only be repeated one time and he or she must try to perform as precisely as possible.

2. After repeating the gestures, ask the patient to relax the arm for 5 seconds, meanwhile start running the code and checking that everything is ready to start.

3. Once the code is running, the patient will start doing all the gestures whenever he is told to. All steps the patient must follow (when to do the gesture, when to rest, when to wait…) will appear on screen. Anyway, the executor should be supervising the entire process in case anything goes wrong or the patient has any question.

4. After measuring all the gestures three times, slightly remove the electrodes from the patient's arm and place them on his other arm and repeat all the procedure twice again.

5. Lastly, remove the electrodes from the other arm, stop running the code, ask the patient if everything went well and check that the data has been uploaded correctly to the Firebase real-time database.

To be noticed that the recording of one gesture lasts 14 seconds. During this lapse of time, the gesture must be repeated two times. During these 14 seconds, approximately 100 data values will be recorded.

The goal is to perform the test in a range of 30 to 40 people. Which will mean that there will be 900 to 1200 gestures recorded, *equations 4 and 5*.

$$30\ to\ 40\ people\ *\ 2\ arms * 5\ gest.\ * 3\ rounds\ =\ \mathbf{900\ to\ 1200\ rounds} \quad \textit{(Eq. 4.)}$$

$$900\ to\ 1200\ rounds * 1\ {}^{gest.}/_{round}\ =\mathbf{900\ to\ 1200\ gest.} \quad \textit{(Eq. 5.)}$$

Overall, if everything went as predicted, once finished measuring there will be between 90.000 and 120.000 data values from the sEMG signal, *equation 6*.

$$900\ to\ 1200\ rounds\ * 100\ data\ val. = \mathbf{90.000\ to\ 120.000\ data\ val.} \quad \textit{(Eq. 6.)}$$

4.3.3. Acquired data

During the acquisition procedure, some basic characteristics from the patients have been registered to keep a track of the time of data that is being recorded. This basic information has been the gender and the age of the patient, and if the patient shows any type of injury on any of its arms. This information might be relevant when it comes to analyse the obtained results of the classification model.

The total number of measured patients has been 30 so, as explained in the earlier section, there is a total of 90.000 data values and 900 registered gestures and, from these 900, 180 recordings of each gesture.

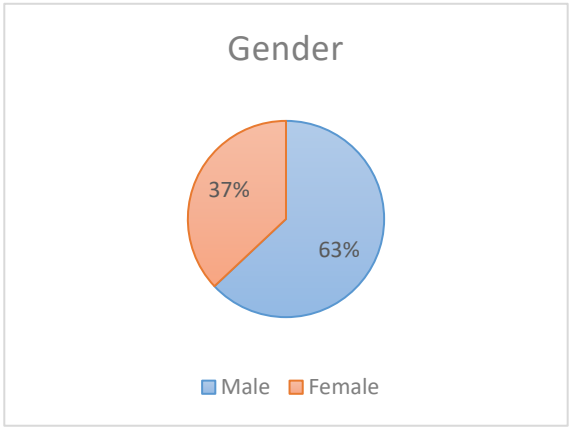


Figure 4.35.- Gender distribution of the patients. *Source: own.*

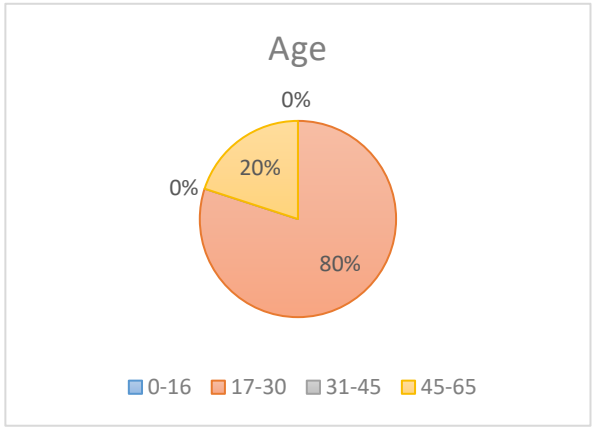


Figure 4.36.- Age distribution of the patients. *Source: own.*

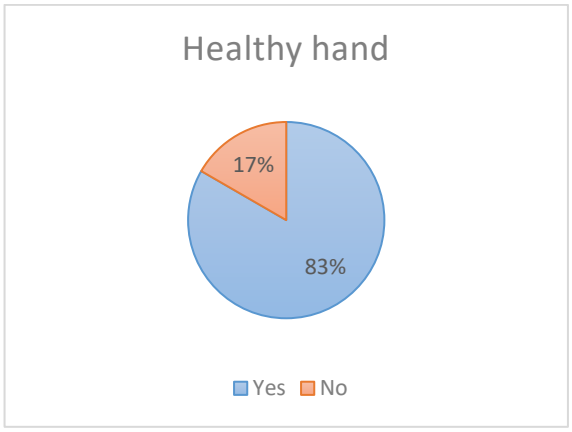


Figure 4.37.- Distribution of arms health. *Source: own.*

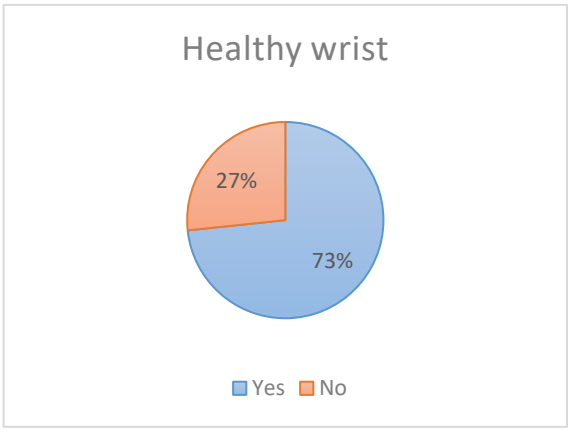


Figure 4.38.- Distribution of wrist health. *Source: own.*

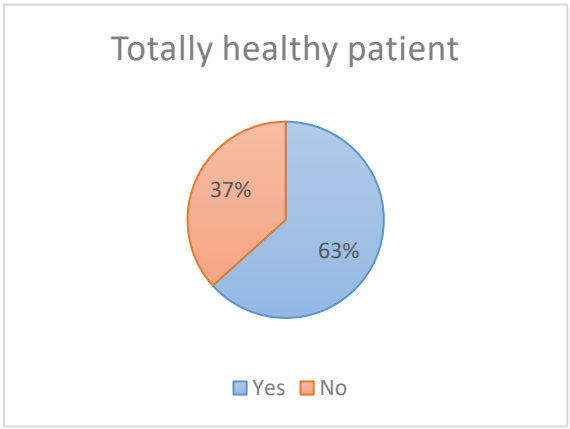


Figure 4.39.- Distribution of the global health of the patient. *Source: own.*

From *Figure 4.35* to *Figure 4.39* the recapped information from the measured patients is shown. Although many patients have healthy hands and wrists, the result of the total healthy patients has a higher rate of people with lesions than separately. This is due to most of the patients who had a wrist injury not having a hand injury and vice versa, so as both groups of patients are considered as non-healthy, the final number of this group of patients grows and equals more the balance than the separately. Another worth mentioning factor is the age distribution of the patients, as the measuring process has all been executed by myself, the target patient has been people between 17-30 years, mostly friends, and people between 45-65 years, their parents. Because of that, the age ranges considered as child, 0-16 years, and young-adults, 31-45, have no recorded measures. These factors may affect the later results of the designed classification model, the health state of the patients. The fact of looking for patients with injuries has been made on purpose as the final goal is to help monitor or predict future illnesses in patients with hand-wrist problems, so it would not have made sense to measure only healthy patients with similar recorded muscle activity patterns.

The table with all the recollected information of the patients can be found in Attachment A2.

5. Artificial Intelligence (AI)

This is the second pillar of this thesis, Artificial Intelligence. After having seen all the hardware needed for this project, as well as the measurement procedures, its data and the firmware used to it, it is time to introduce the brain of the process. This section explains what happens with the data once it is stored in the real-time database in Firebase. Firstly, some background information about AI will be given, to establish the basis of where we are and where we want to go. Then, the environment where everything has been designed and executed as well as what an environment is, why this one was chosen, etc. After that, the core of this pilar; Machine Learning. This is the branch of Artificial Intelligence that performs the classification of the sEMG signals. Lastly, the obtained results of the entire process have been interpreted, compared and analysed with the goal of obtaining firm conclusions regarding the work carried out.

5.1. Background Research

AI is the ability of a computer to do tasks usually dealt by humans since they require intelligence and discernment [41]. In other words, simulating human intelligence in programmed machines to think like them and mimic their actions. It also refers to machines that have an association with humans in the sense that they can learn and solve problems. Its ideal feature is having the ability to choose actions that have the best opportunity of achieving a goal [42]. Alongside this characteristic, Artificial Intelligence has many others which also give name to its subsets like: Deep Learning, Artificial Neural Networks, Feature Engineering, Intelligent Robots, Machine Learning, etc.

But before getting deeper into AI characteristics, where and when was this concept born and, most important, why? Well, it also started when sir Alan Turing wrote in 1950 a scientific paper: “Computing Machinery and Intelligence” after breaking the Nazi encryption in the second World War with a machine called Enigma*, See Figure 5.1.* [43]. This established the main goals and vision of AI and considered Artificial Intelligence as the branch of Computer Science aiming to answer Turing’s queries affirmatively.

Figure 5. 1.- Enigma machine.
Source: mirror.co.uk

As this technology is a branch of Computer Science, AI does also have its own set of branches and subtypes*, See Figure 5.2*. As

we can see in the figure, Artificial Intelligence encompasses other methods that get increasingly specific as the complexity of a problem increases. The biggest branch of AI is Machine Learning (ML), a method of data analysis based on the idea that a system can learn from data, identify patterns and even make decisions with a minimal human interaction [44]. ML englobes different brain-inspired methods, which try to replicate the functioning of animal brains. This sub-branch consists of Neural Networks (NN or ANN) and Spiking Neural Networks (SNN). Both are computing systems inspired by biological neural networks that create, as mentioned before, animal brains. The first one is based on a set of nodes, which imitate the neurons, able to transmit a signal to other nodes, as a synapse. This generates connections between nodes called edges, which typically have a weight that adjusts as learning progresses. On the other hand, SNN mimic natural neural networks in a more closely related way as they add the concept of time into their operating model [45]. Lastly, the deepest branch of Artificial Intelligence is Deep Learning (DL). The easiest way to understand this technique is following the branch path that leads to it, so it is considered a subfield of Machine Learning concerned with algorithms inspired by Artificial Neural Networks. The aim of this field is to making learning algorithms more efficient and contribute to make revolutionary progress in ML and AI [46].

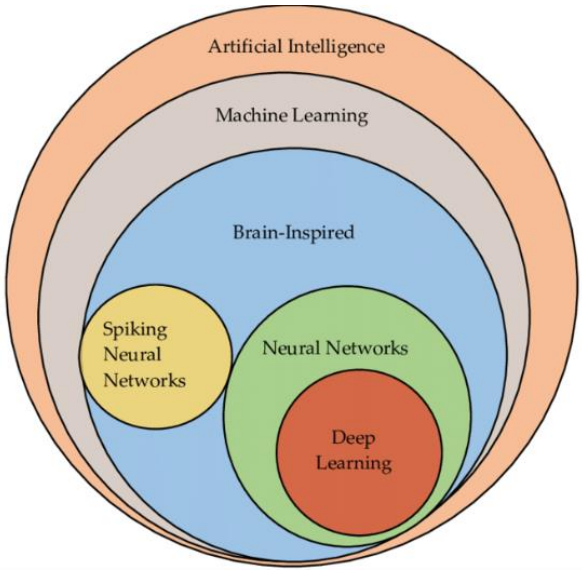


Figure 5. 2.- Taxonomy of AI. *Source: researchgate.net*

To create a technology that allows computers and machines to act intelligently scientists have faced many obstacles, which have been divided during time into subsets of problems, making a division of categories and creating new goals to achieve that main and principal goals of Artificial Intelligence. Some of those sub-problems are: problem solving, knowledge representation, planning or learning [47].

Problem solving pretends to imitate step by step the way a human being reasons when it comes to making deductions or solving any kind of puzzles, and it works, researchers have been developing it

for decades. The main inconvenience is that for big-sized problems, the time that the computers or machines need to solve those problems is astronomically big. So, nowadays priority is to reduce that algorithm's execution by making the process more efficient.

Another goal, sets on the core of AI research, knowledge representation and engineering. Representation of objects, properties, situations, states or time are among the things that machines need to master to be able to solve many problems they are expected to. By learning this, the machine creates sets of things it knows about, starting with the most generals and with more information and research, enabling the possibility of providing foundation for further knowledge.

There are plenty of other fascinating goals and characteristics that this technology has, but when it comes to solving problems, AI has a wide set of tools ready to help with those processes. AI has tools for many specific things, such as: search and optimization, logic, uncertain reasoning, learning, etc. These tools not only help develop but also, optimise the networks [48].

A few of the most well-known AI tools are: *Scikit Learn*, a Machine Learning library that includes many calculations for AI, data mining, predictive data analysis, etc. [49]. Another famous platform is *Tensorflow*, this platform from Google allows the user to set up, train and send neural systems with a noticeably big amount of data in an extremely brief time. It is used when it comes to applying profound learning calculations and more complex algorithms [50]. *Theano, Caffe, MxNet, Keras, PyTorch, CNTK, Auto ML, OpenNN* and many others are examples of infinite tools that facilitate the execution time and increase the efficiency of the processes involved in working with IA.

Artificial Intelligence is a name that sounds "cool" to everybody, but people do not really know what applications AI has beyond robots or flying cars. In today's world, this technology has been or is being integrated everywhere. Some examples of its applications are in sectors like: e-commerce (personalised shopping experience, AI-powered assistants or even fraud prevention techniques), education (smart content for learning, voice assistants, personalised learning for students with difficulties or automated administrative tasks for educators), lifestyle (from spam filters or recommendation systems to autonomous vehicles or facial recognition), navigation (GPS technology with real-time interaction and update) and many others like HR, robotics, healthcare, agriculture or gaming are some of the applications that can be seen reflected in today's world [51].

5.2. **Environment**

In AI, it is sometimes more difficult to understand the problem itself than finding a solution. When looking for a solution, we focus on two aspects: the dataset used and the AI model. But we ignore another important aspect of the problem, the **environment**. The area where the factor must

operate in, is called the environment. There are several types of AI environments available for different AI applications, some of those types are [52]:

- **Single Agent vs Multi Agent.** An agent is anything that receives objects from the environment with the help of sensors and acts on them with the help of an effector or actuator. The number of agents involved in the environment will define if it is a single (only one agent) or a multi (more than one agent) agent AI environment. The design of the agent is different for both environments.

- **Complete vs Incomplete.** The complete environment is enough to completely solve a problem. Conversely, if the system is not able to predict all movements to solve the problem it is considered as an incomplete environment.

- **Fully Observable vs Partially Observable.** In case the sensor of the agent can sense or have access to the whole state of the environment at any point in time, it will be a fully observable environment. Meanwhile, if this does not happen it will be considered as a partially observable environment. If the agent has no sensor in all environments, it is considered as unobservable.

- **Competitive vs Collaborative.** If a group of agents work against each other to maximise the outcome by implementing competitiveness to reach the goal, it is considered a competitive environment. Whereas, if those agents collaborate and work together to reach that goal, the environment is considered collaborative.

- **Static vs Dynamic.** If the environment can change as the agent thinks, it is called dynamic, otherwise it is named static. Static environments are easy to handle because the agent does not have to see the world when it decides to act. However, in a dynamic environment, agents need to see the world every time they act.

- **Discrete vs Continuous.** If an environment has a finite number of perceptions and actions, it is called a discrete environment; otherwise, it is called a continuous.

- **Deterministic vs Stochastic.** If the agent's current state and the chosen action can completely determine the next state of the environment, such an environment is called a deterministic environment. The random environment is, as its name says, random and cannot be completely determined by the agent.

- **Episodic vs Sequential.** In case it is an episodic environment, there will be a series of actions that take place once, and that action requires only the, that moment, current perception. However, in a sequential environment, the agent needs memory of past actions to determine the next best action.

Another important aspect before starting with the model itself is the **code editor**, this is an essential tool for coding, designing or writing. It allows the user to create, edit or update a file or a project, manage the contents of a directory or many other things. Formerly, there were only specific code editors for each platform, which diminished the chances of interaction between different operating systems (OS) or languages. Nowadays this is different, and all code editors are cross-platform and allow the user to use it in different OS and environments without the need of having to adapt or learn how this pristine environment or OS works. Furthermore, the user can personalise and change the behaviour of the code editor by changing some configurations and making it the most comfortable [53].

For this thesis **Visual Studio Code** (VS Code) has been chosen as the code editor. This code editor was released in 2015 by Microsoft and since then it is considered as one of the standards for software development. This editor is available on all platforms (Mac, Windows and Linux). The main reasons why this editor has been chosen are, on one hand, the previous use of this code editor in the past and, therefore, familiarity with it. On the other hand, its compatibility with every programming language and its substantial number of extensions and plugins made this code editor the perfect choice for this project. Anyway, any other code editor compatible with python could have been used for it.

After knowing which code editor has been used, it is important to know in which **programming language** this project has been written. The programming language is the way a developer communicates with the computer. This set of instructions written in a specific language (C, C++, Java, Python, etc.) allows the developer to perform specific tasks. There are three types of programming languages: **low-level**, **high-level** and **middle-level programming languages** [54].

The first ones, **low-level programming languages**, are machine-dependent languages with no need of compilers or interpreters, so it can be run fast. The two subtypes of low-level programming languages are: **Machine Language**, usually displayed in binary or hexadecimal form. An advantage of this language is that it can execute the code much faster than high-level languages since it does not need any translation as the computer directly understands that language; and **Assembly Language**, this one is designed for processors and represents a set of instructions in a human-understandable form, this language uses an assembler to convert the code to machine language.

The second ones, **high-level programming languages**, are designed for developing software programs and websites. In this case, the need of a compiler or interpreter to translate the language into machine language is mandatory as these languages are user-friendly and easy to read, write and, therefore, not in binary or hexadecimal. This language type is divided into three subtypes: **Procedural Oriented Programming language (POP)**, this language divides the codes into small processes called routines or functions and is used to create programs that can be accomplished using an editor like IDE. The advantage of this programming language is it helps the user to track the program flow in an uncomplicated way and its code is reusable for other parts of the program. Some examples of this language are: C, Basic or Pascal; **Object-Oriented Programming languages (OOP)**, this language divides programs into smaller blocks, called objects. Its principal use is the implementation of real-world entities making the program efficient and easy to use. An advantage of this language is how fast and easy its codes can be executed, modified or debugged. Examples of these languages are: C++, Java or Python; and **Natural Language**, which is a language used by machines to understand and interpret human languages while developers use it to translate, summarise automatically, segment topics, etc.

The last one, **middle-level programming languages**, are a mixture of both mentioned before and, therefore, have characteristics of both. Concerning the high-level languages, it keeps its user-friendliness and for the low-level languages it keeps a close relation to machine language. An example of this language is C++.

For this thesis, **Python** has been used as the programming language. This language is the most used in areas like: AI, machine Learning, Big Data or Robotics. Besides that, it is easy to read, understand and write; it can execute the code line-by-line making it easier to find errors in the code and it is platform-independent. For more information about Python check [55].

5.3. Machine Learning (ML)

After having checked all different Artificial Intelligence sub-branches and methods, it was confirmed that Machine Learning would be the most suitable technique to develop this thesis as it is facing a classification problem and that is one of the most common ML tasks. This section contains, firstly, some theoretical information about Machine Learning, its methods and characteristics; and secondly, the created algorithms and its explanations created to try solving the problem.

5.3.1. Background Research

There are diverse ways of training Machine Learning algorithms and each of those ways can be classified into a type. ML has many distinct types or methods, but the three more popular and

therefore more used are: **supervised learning**, **unsupervised learning** and **reinforcement learning**. To know which method adapts the best to the user's needs, two main things must be checked. On one side the type of problem (classification, clustering, etc.) and on the other side the type of data the user has (labelled or unlabelled data, the main difference between them is how many parameters of the data are completely machine-readable) [56].

Supervised learning is an Artificial Intelligence method in which computer algorithms are trained on input data that has been labelled as a specific output*, See Figure 5.3*. The model is trained until it can recognize the underlying patterns and relationships between input data and output labels, allowing it to provide accurate labelling results when presented with unprecedented data. Its two main problem types are: classification and regression. Supervised learning is training-based, during its training phase, the system receives labelled records that tell the system which outputs are associated with each specific input value. The trained model is then presented with test data: this is labelled data, but the label has not been shown to the algorithm. The goal of the test data is to measure how accurately the algorithm performs on unlabelled data [57].

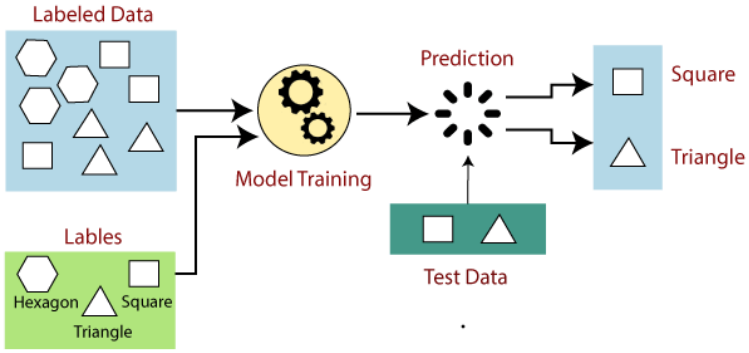


Figure 5. 3.- Example of Supervised Learning. *Source: torrellesdefoix.cat*

The second type, **unsupervised learning**, is a machine learning technique that does not require user supervision of the model*, See Figure 5.4*. Instead, the model can discover previously undiscovered patterns and information on its own. It deals with untagged data. Compared to supervised learning, unsupervised learning algorithms allow users to perform more complex processing tasks. However, unsupervised learning can be more difficult to predict than other natural learning methods. Unsupervised learning algorithms include clustering, anomaly detection, neural networks, etc. [58].

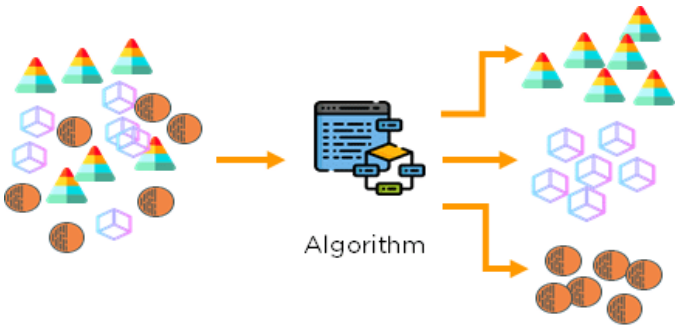


Figure 5. 4.- Unsupervised Learning example. *Source: researchgate.com*

Lastly, **reinforcement learning** is about taking the appropriate action to maximise the reward in each situation*, See Figure 5.5*. It is used by various software programs and machines to find the best possible behaviour or path that should be taken in each situation. Reinforcement learning differs from supervised learning in that in supervised learning the training data contains the answer key so the model can train itself with the correct answer whereas in reinforcement learning there is no answer but the reinforcement agent decides how to handle a given question Task. In the absence of a training dataset, it must learn from experience [59].

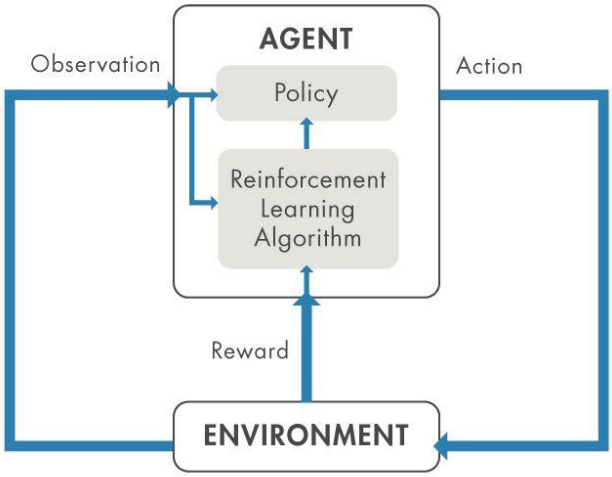


Figure 5. 5.- Explanation of Reinforcement Learning. *Source: mathworks.com*

The following figure, *See Figure 5.6.,* shows a short schematic summary of these three Machine Learning types and its inputs and outputs.

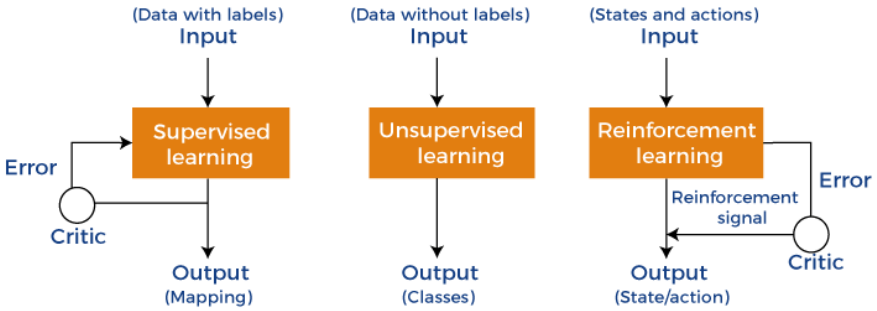


Figure 5. 6.- Classification of ML types. *Source: javatpoint.com*

No matter which type of Machine Learning is being used to solve a problem, the user will have to use a method to train the algorithm. This method will be chosen depending on the ML type, the data and the model itself. There are plenty of ML methods, but a bunch of the most famous and used are: **regression**, **classification**, **clustering**, **dimensionality reduction** or **neural networks** [60]. Each model is characteristic and used in one of the three Machine Learning types [61].

There are many models used for **Supervised Learning**. This type is the easiest to understand, and they just need a dataset and a function to be trained to develop them. Its models are classified into regression and classification.

The **regression** method is classified into the supervised learning type and its goal is to predict or explain a specific value based on a previous dataset (i.e., predicting the price of a property with a previous pricing dataset or predict the energy consumption of a home). The most common models used to apply regression to a model are:

- **Linear Regression**. It is considered the simplest method as it uses the linear regression equation*, See Equation 7,* and to train to be trained it just needs inputs of *(x, y)* values. Then the model calculates a line that minimises the distance between those given data points and the calculated line, *See Figure 5.7*.

$$\boldsymbol{y = m * x + b} \qquad \textbf{(Eq. 7)}$$

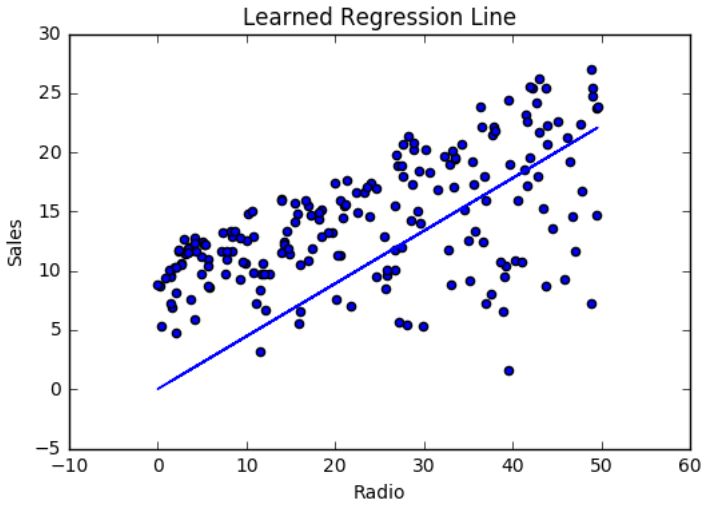


Figure 5. 7.- Example of linear regression*. Source: ml-cheatsheet.readthedoc.io*

- **Decision Tree**. This model can be used for both regression and classification and its functioning consist of a structure with the shape of a tree, *See Figure 5.8.,* where decisions with its corresponding answers (outcomes) until it reaches the searched answer. This is a very intuitive method and visual method; besides that, it is extremely easy to implement as it just needs to add branches and nodes for it to be extended. On the other hand, this method is not fully accurate or, at least, needs an excessively considerable number of nodes and branches to be more accurate.

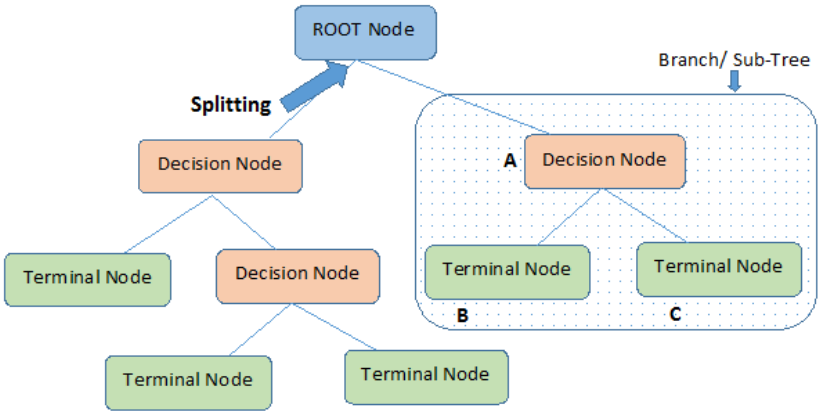


Figure 5. 8.- Decision Tree structure. *Source: medium.com*

- **Random Forest**. Its name explains what this method consists of this model is based on many decision trees, therefore it is called forest*, See Figure 5.9*. Each tree reaches an answer (outcome) and the mean or average outcome from all trees will be considered as the result. As it happens with Decision Trees, this method can also be used for regression and classification.

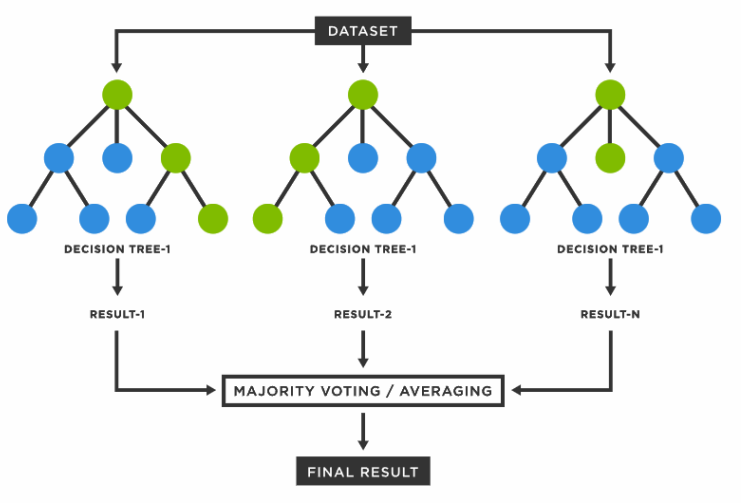


Figure 5. 9.- Structure of a Random Forest. *Source: tibco.com*[40]

- **Neural Networks**. They have the goal of catching non-linear patterns in data by adding layers of characteristics to the model*, See Figure 5.10*. This structure pretends to work like a human brain by bouncing data around the nodes (that imitate neurons) until the information reaches the output. This model is technically a sub branch of Machine Learning and has its own classification into different subtypes, as already explained in earlier sections.

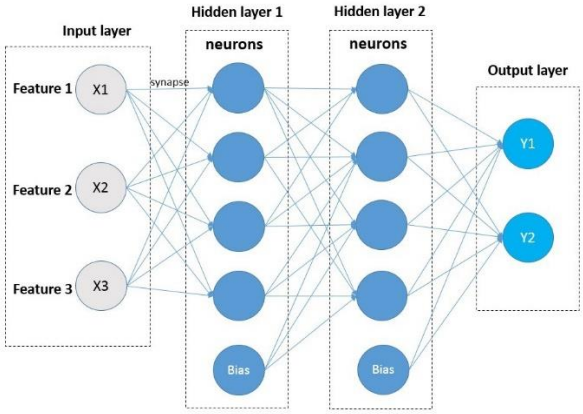


Figure 5. 10.- Neural Network example. *Source: miro.medium.com*

Classification is another supervised learning method which pretends to predict a class value (i.e., predict if a customer will buy a product or classify images depending on what their content is). Its models pretend to make conclusions after observing data. This thesis is a clear example of a classification problem. Two ways of classifying can be different in Machine Learning: binary classification (when the output can only be between 2 options i.e., 1 or 2, yes or no, etc.) and multi-class classification (when there are multiple classes, more than two, to be classified). Some of the typical algorithms to solve these problems are:

- **Logistic Regression**. This regression algorithm can be also used for classification and works like linear regression, but in this case, it predicts the output as a probability in the range between 0 and 1 using a sigmoid function, *See Equation 8* and *Figure 5.11*, being 1 a 100% of accuracy. The model applies the function to each class and the one with a highest accuracy is the output of the predicted value.

$$S(x) = \frac{1}{1+e^{-x}} \qquad \textbf{(Eq. 8)}$$

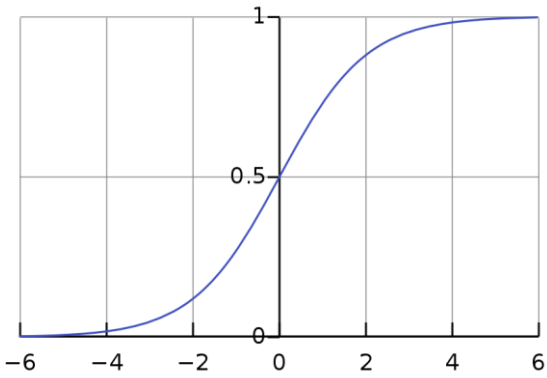


Figure 5. 11.- Sigmoid function representation. *Source: wikipedia.org*

- **Support Vector Machine (SVM)**. This model can also be used on regression tasks although it is mostly used on classification problems. The model finds the boundaries (hyperplane) that adjust to the best decision in a N-dimensions problem and the algorithm extracts the vectors, from these boundaries, that delimits them*, See Figure 5.12*.

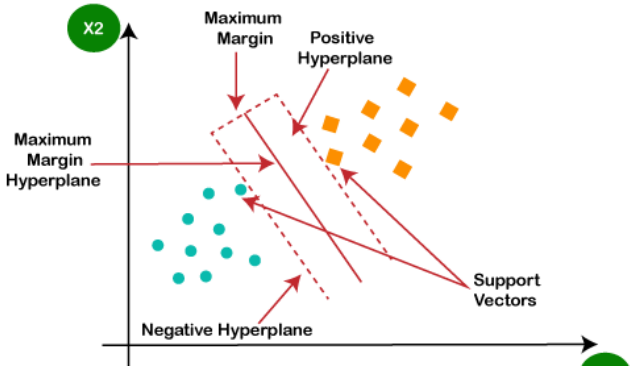


Figure 5. 12.- SVM example. *Source: javatpoint.com*

- **Naïve Bayes**. This model is based on the Bayes theorem, *See Equation 8,* an immensely popular statistics theorem and goes after an independent (naïve) conclusion between features. This assumes the independence of any value from another one.

$$P(y|X) = \frac{P(X|y)*P(y)}{P(X)} \qquad \textit{(Eq. 8)}$$

- **K-Nearest Neighbours**. This method assumes that there is similarity between things that lay near each other. This similarity is defined by the distance between data points, and this helps to delimitate an area where this similar point resides, *See Figure 5.13*.

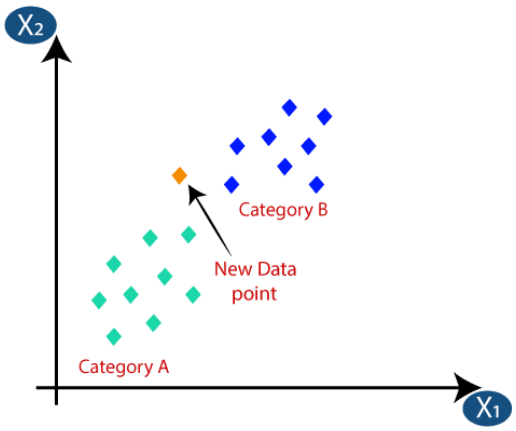


Figure 5. 13.- Example of K-Nearest Neighbours method. *Source: gruposincom.es*

When it comes to **Unsupervised Learning**, the model can learn hidden patterns from the dataset on its own. The main tasks of this model are **clustering** and **dimensionality reduction**.

Clustering, this method has as goal grouping observations with similar features by letting the algorithm define the output. The quality of the solution for a problem can only be checked via visualisation. The most famous clustering procedure is the **K-Means method**, *See Figure 5.14*. This method creates K centres within the data, then links the data point to its nearest created centre and after that it calculates the new centres of each group. Lastly, only if the centre changes the process needs to be redone. Hierarchal Clustering and DBSCAN are two other models that work fine for clustering as well.

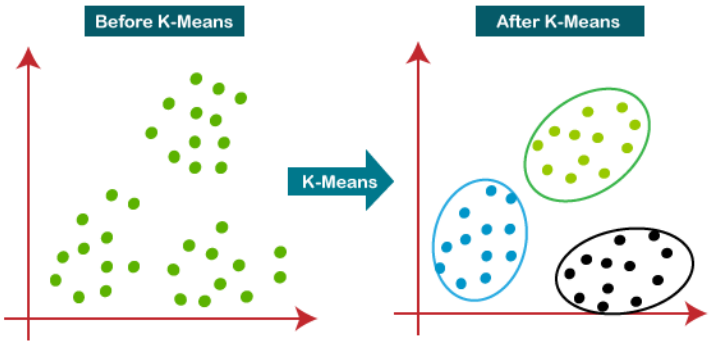


Figure 5. 14.- K-Means method example. *Source: analyticsvidhya.com*

Dimensionality reduction is used for deleting non relevant information from a dataset (i.e., when it comes to testing microchips in a process and it has thousands of features with many of them being non relevant). The aim of this method is to avoid redundant information in an excessively large set of data and focus on the

relevant features. For it, dimensionality reduction algorithms are being used. The most popular of those algorithms is the Principal Component Analysis (PCA), this algorithm decreases the size of feature space by searching for vectors that increase to the maximum the linear variation of the data. This reduces incredibly the amount of data without wasting excessive valuable information.

Last, the models applied to **Reinforcement Learning** stand out for learning actions from a given set of states to reach a goal state. This model acquires feedback signals after a state to improve the performance, this feedback is obtained by interacting with the environment. This learning model could be compared to the way humans learn. Some famous methods that use this learning are **Q-learning**, **SARSA** or **Deep Q Networks**.

The aim of **Q-learning** is learning a series of rules making the algorithm able to tell an agent which action to take depending on the circumstances. It includes Q-values for any state-action pair indicator of reward and tries to maximise the Q-value.

State-Action-Reward-State-Action (SARSA) is a method which bases its decision-making on the Markov process and uses a performed action to learn the Q-values.

Deep Q Network (DQN), is a model of **Q-learning** combined with Neural Networks, specifically inside a NN. For this model the Neural Network uses Q-values for each action based on the state.

Besides all these models there are many more which are used for more specific tasks and many more will be developed in the future. It is also notable to mention that, as every method, Machine Learning has its limitations. The most notable and worth mentioning limitation is data. A low amount of data will end up in a poor result. Besides that, the lack of good data is as important as the lack of data because as Machine Learning works by training, if the algorithm is trained with bad data or features when it comes to its performance, it will be extremely poor [62].

5.3.2. Pre-processing

This section will explain the content and how the first of the three codes of this project has been created, the pre-processing of the signal.

The pre-processing of the signal is the first and most important part of the problem. How good the data is processed and treated will be reflected in how good or, at least, reliable obtained results are. For this case, the sEMG measures raw data. This data needs to be processed before training it in the algorithm. For a good signal processing, different methods have been used to end up having a signal as clear as possible to be later evaluated.

It consists of two blocks, the signal treatment and the extraction of features. The first one consists of preparing the data to make it easy to use later in the features extractions, that second part is about calculating some characteristics that make differences between gestures to find a way for classifying them.

5.3.2.1. Signal treatment

For the signal treatment, the process has been divided into six blocks, *See Figure 5.15.,* where the data will be imported from Firebase until extracting its features to start the training of the model. During the process of explaining the steps taken to treat the signal, a representative signal segment will be shown as an illustrative example of the process.

If the rest of gestures want to be analysed, check on Attachment C9, and the entire pre-processing functions and code will be found on Attachment C1 and Attachment C2 respectively.

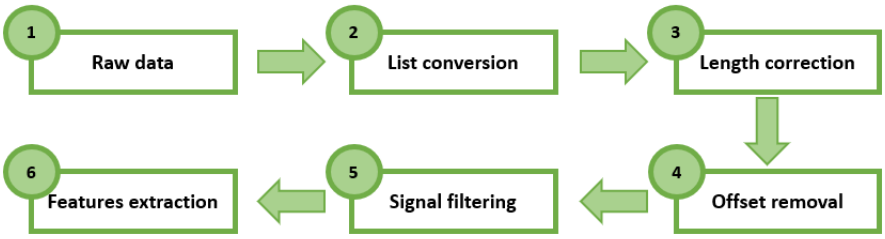


Figure 5. 15.- Data pre-processing structure. *Source: own source*

The first section, **raw data**, is responsible for importing the data from the online real-time database and saving it into variables*, See Figure 5.16.* As there have been five gestures recorded, the data of each of them will be stored into a different variable. The command that executes this order is *get*, this command belongs to a library called *pyrebase* [63]. This library was written to be used in python and allows linking python to Firebase, as well as interacting with it from python and, in our case, importing data from there. To run the command the user must specify the path of the database where the data that wants to be used is stored, then add the order *get* at the end of the line.

```
40  #Get recorded signal from Fyrebase
41  data1 = db.child("EMGdata").child("Gesture1").get() #--> gets all the content of the node (path)
42  data2 = db.child("EMGdata").child("Gesture2").get()
43  data3 = db.child("EMGdata").child("Gesture3").get()
44  data4 = db.child("EMGdata").child("Gesture4").get()
45  data5 = db.child("EMGdata").child("Gesture5").get()
```

Figure 5. 16.- Importing saved data into Python variables. *Source: own source.*

Once the raw data is stored in variables (data1, data2, data3, data4 and data5), the next step is to **convert the incoming data into lists**, so it is much easier to control that data. The data is imported in JSON format, this format saves the data in a human-readable format consisted of attribute (key)-

value pairs, *See Figure 5.17.,* the information are the values, the keys are irrelevant for the classification, they work as an identifier of each data value to make that value unique.

Figure 5. 17.- JSON format data, key (left) and value (right). *Source: own source.*

To remove those keys, a short loop that runs through all the pairs of values of the variable is executed*, See Figure 5.18*. This loop copies the value and stores it in a new variable list (gest1, gest2, gest3, gest4 and gest5) without the key. The result is five lists with only the values of each gesture.

```
#Removes key from JSON and appends in list
for val1 in data1.each():
  gest1.append(val1.val())
```

Figure 5. 18.- Loop to keep only the value from the data. *Source: own source.*

After having all values stored in lists, a **length correction** is applied. Each recorded gesture has, theoretically, a length of 100 data values but, due to an interruption caused between the ESP32 Wi-Fi library and the *analogRead* order, the microcontroller is not able to read all values at once and breaks the connection and instantly reconnects to be able to finish the recording and sending data from the microcontroller to the database. The result is some gestures recording with 99 values and other ones with 100. It is not a big deal, due to the missing value always corresponds to a repose position but, it is necessary that each gesture has 100 values for a better signal segmentation without overlapping.

The function*, See Figure 5.19*, takes the list of each gesture that has 99 values instead of 100, and every 99 data values adds a 0, until having gone through the whole list. Then stores this data in a new list called *new_gest*, which will later be recalled gest1, gest2, gest3, gest4, gest5 again.

```
#THIS FUNCTION CORRECT THE LENGTH FROM THE SIGNALS
def correct_length(gest):
    new_gest = []
    for i in range(0,len(gest),99):
        new_gest += gest[i:i+99] + [0]
    return new_gest

```

Figure 5. 19.- Length correction function. *Source: own source.*

Now that the data values are stored in lists, its visual representation looks like in *Figure 5.20.* This is a segment of the whole dataset from gesture 3 (flexion of fingers), the length of the segment is of 0.6 seconds and six gesture repetitions are discernible. When the data values increase corresponds to the execution of the gesture, meanwhile when the data values tend to 0, the muscle is reposing and it also marks the separation between gestures. It is also appreciable that the signal is not centred on the Y axis, it does not start from zero. That is due to the reference of a person will never be zero as it always has some activity and the electrodes are neither perfect.

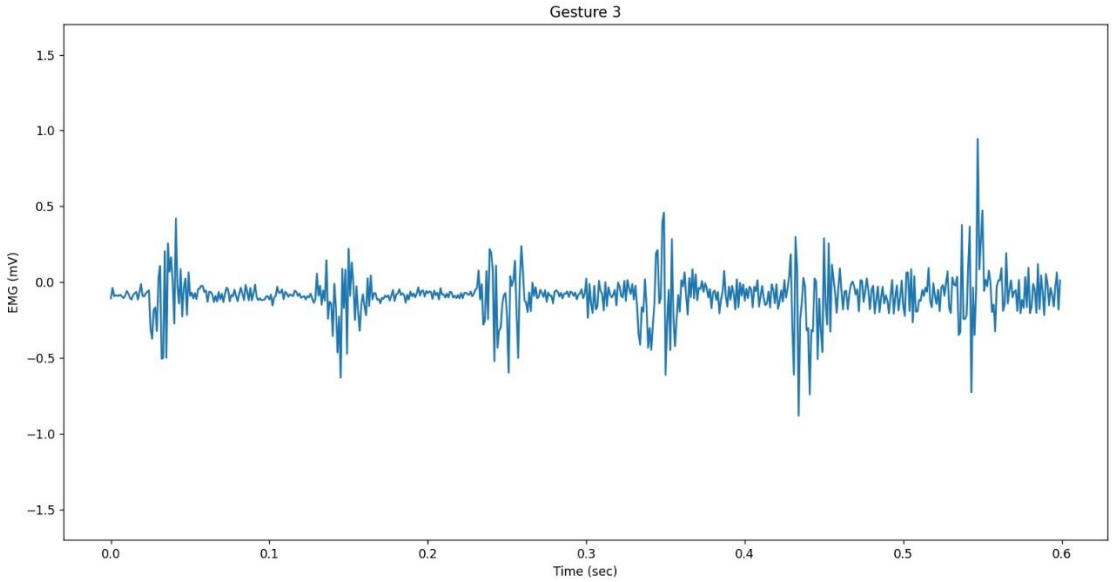


Figure 5. 20.- Segment of data after key removal and length correction. *Source: own source.*

To **centre the signal***, See Figure 5.21,* in 0 and make it equal for all recordings and gestures a function, *rmv_mean,* has been created. This function calculates the mean of the gesture (every 100 values) and subtracts it from the values. Then it appends it to a new list *emg_correctmean* which is later recalled, again, gest1, gest2, gest3, gest4 and gest5. The result of this function is, *See Figure 5.22.,* a signal centred on the Y axis by zero.

```
26   #THIS FUNCTION REMOVES THE OFFSET FROM  THE SIGNAL
27   def rmv_mean(gest, time):
28     for i in range(0,len(gest),100):
29       emg_correctmean = gest - np.mean(gest)
30     return emg_correctmean
```

Figure 5. 21.- Function to centre the signal. *Source: own source.*

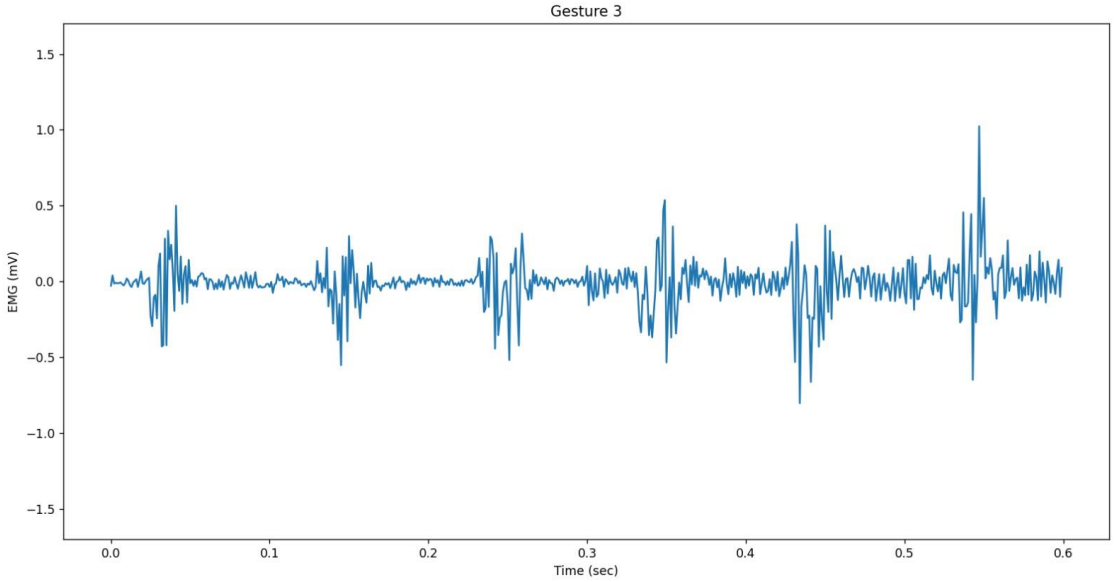


Figure 5. 22.- Same signal segment as Figure 5.19. after centring the signal. *Source: own source.*

The next step is to filter the signal to remove some noise and get a clearer wave for better features extraction. Therefore, a Butterworth bandpass filter has been created to attenuate frequencies outside of a certain range. This is a common signal processing filter. In this filter two frequencies are declared, a high-pass frequency which delimits the maximum frequency accepted, and a low-pass frequency which delimits the minimum frequency accepted. To implement it, Python has a library called *SciPy* [64] which provides mathematical algorithms for many things such as integration or differential equations and it also has a section called *SciPy. Signal* prepared specifically for signal processing.

To create the filter, a low-pass frequency and a high-pass frequency need to be added as inputs of the function. The typical frequency of raw sEMG data is between 6 and 500 Hz. Fast oscillations are usually cause by unwanted noise. To make an accurate filter, the high-pass frequency will be of 20 Hz, meanwhile the low-pass frequency will be of 450 Hz [65]. As the function from the *SciPy* library requires values between 0 and 1, they have been normalised using the Nyquist frequency, *See Equation 9*. The sample rate frequency is of 1 kHz.

$$f_n = \frac{1}{2} * f_s = 0.5 * 1kHz = 500Hz \qquad \textit{(Eq. 9)}$$

After having defined these values, the function can execute*, See Figure 5.23.* It adapts those previously introduced frequency values to, later, apply the filter with the *signal.butter* command. The filter is a 5th order filter, to achieve the maximum precision possible. After that, it saves the value in a variable. In case the filter wants to be visualized, in the input label must be written *plot=True* and then the *if plot* condition will execute. The visual result of the plot is the original signal, blue, and the filtered signal, black, *See Figure 5.24.*

```
def bp_filter(x, low_f, high_f, samplerate, plot=False):
    x = x - np.mean(x)

    low_cutoff_bp = low_f / (samplerate / 2)
    high_cutoff_bp = high_f / (samplerate / 2)

    [b, a] = signal.butter(5, [low_cutoff_bp, high_cutoff_bp], btype='bandpass')

    x_filt = signal.filtfilt(b, a, x)

    if plot:
        t = np.arange(0, len(x) / samplerate, 1 / samplerate)
        plt.plot(t, x)
        plt.plot(t, x_filt, 'k')
        plt.autoscale(tight=True)
        plt.xlabel('Time')
        plt.ylabel('Amplitude (mV)')
        plt.show()

    return x_filt
```

Figure 5. 23.- Designed Butterworth bandpass filter. *Source: own source.*

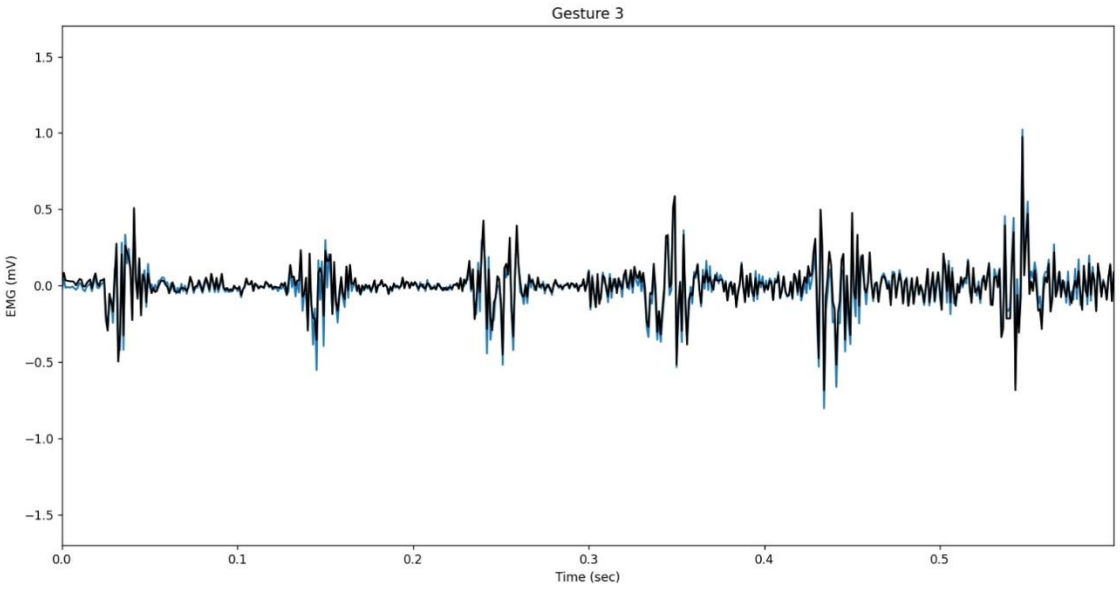


Figure 5. 24.- Visualization of the signal after the bandpass filter. *Source: own source.*

After having filtered the signal through a bandpass filter, a second filter has been applied trying to reduce even more the noisy signal. This second filter has been a notch filter. What this filter does is blocking frequencies between a given range, the stop band [66]. This stop band is between 59 and 61 Hz and is executed the same way as the band pass filter but, this time, with a 4th order gradient*, See Figure 5.25.* The results of filtering the signal for a second time, *See Figure 5.26,* show the signal after applying the bandpass filter, blue, and the signal after applying the notch filter, black. The results are less significant as the first filtering, but still some signal correction can be appreciated.

```
def notch_filter(x, samplerate, plot=False):
    x = x - np.mean(x)

    high_cutoff_notch = 59 / (samplerate / 2)
    low_cutoff_notch = 61 / (samplerate / 2)

    [b, a] = signal.butter(4, [high_cutoff_notch, low_cutoff_notch], btype='stop')

    x_filt = signal.filtfilt(b, a, x.T)

    if plot:
        t = np.arange(0, len(x) / samplerate, 1 / samplerate)
        plt.plot(t, x)
        plt.plot(t, x_filt.T, 'k')
        plt.autoscale(tight=True)
        plt.xlabel('Time')
        plt.ylabel('Amplitude (mV)')
        plt.show()

    return x_filt
```

Figure 5. 25.- Designed notch filter. *Source: own source.*

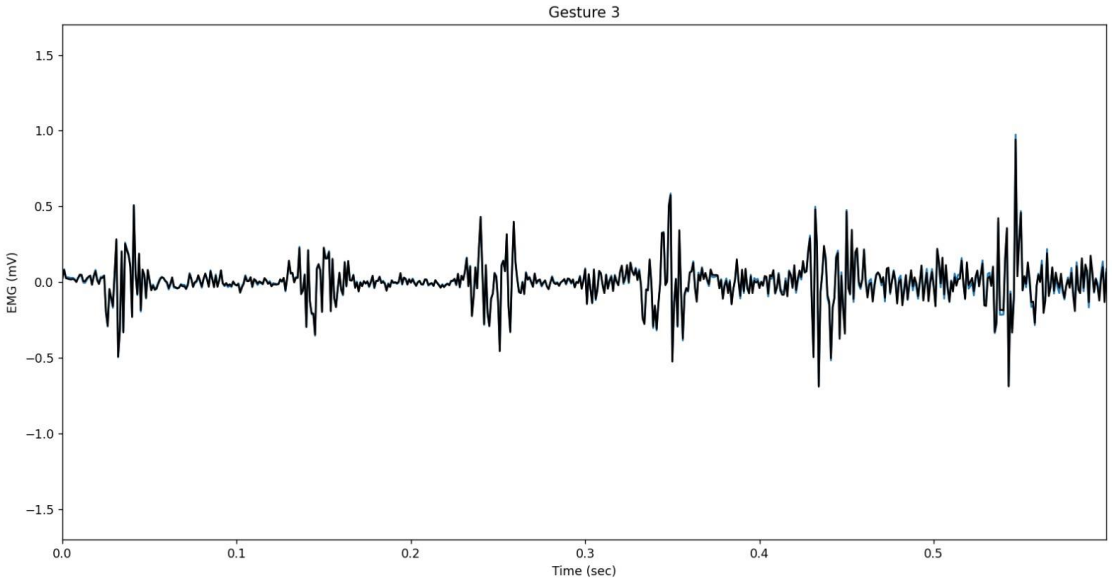


Figure 5. 26.- Visualisation of the signal after the notch filter. *Source: own source.*

Lastly, after having filtered the signals and having stored them into new variable lists, the pre-processing is considered done and the **extraction of the features** is ready to be executed.

5.3.2.2. Features extraction

Features are those values derived from the dataset that pretend to give non-redundant information to facilitate the later learning. This also avoids adding the whole dataset to the learning algorithm full of redundant and overlapping information which is not valuable for a proper training.

It is worth mentioning that for these calculations a helpful Python library for mathematical operations has been used, this library is called *NumPy* [67]. One of the advantages of this library is that it allows to perform those operations even with arrays, the format of the list that will be worked on.

There are many features that can be useful for a model but because this is a classification model from a temporal signal, specifically a signal coming from a sEMG, the most typical features to implement are:

[*A small explanation before explaining the features. N refers to the length of the list and x(n) to a value from that signal list. This value N, will appear as n in the following figures, is a variable with the function len(signal_list) which calculates the length of the list]*

Mean Absolute Value (MAV). Corresponds to the mean of the signal, in other words the average value of the dataset*, See Equation 9,* its calculation is the summation of all the absolute values of the signal divided by the length of that signal list. In Python, this feature has been calculated with the help of *NumPy*, *See Figure 5.27.*

$$MAV = \frac{1}{N}\sum_{n=1}^{N}|x(n)| \qquad \textbf{(Eq. 9)}$$

```
#calculation of MAV
mav = np.sum(np.abs(signal))/n
```

Figure 5. 27.- Equation to calculate MAV in Python. *Source: own source.*

- **Maximum Value (MAX).** As its name says, this feature is the highest value of the signal. That highest value will be, when it is close to the maximum, near 1.7 mV, as it is the maximum that the sEMG can measure. To find this value, the function *max* has been used, *See Figure 5.28.*

```
#calculation of MAX and MIN
max_val = signal.max()
```

Figure 5. 28.- Function to calculate MAX in Python. *Source: own source.*

- **Minimum Value (MIN).** This feature is exactly the opposite as the one mentioned before. It finds the lowest value of the signal; in this case it will be near -1.7 mV when it comes near its limit. And its calculation in Python uses the *min* function, *See Figure 5.29.*

```
min_val = signal.min()
```

Figure 5. 29.- *Function to calculate MAX in Python.* Source: own source.

- **Root Mean Square (RMS).** Also called quadratic mean, it is equal to the square root of the mean of the square values of the signal*, See Equation 10*. For its calculation in Python*, See Figure 5.30, NumPy* has been used to help calculate the square of the signal, the summation of the squares and the square root.

$$RMS = \sqrt{\frac{1}{N}\sum_{n=1}^{N} x(n)^2} \qquad \textbf{(Eq. 10)}$$

```
#calculation of RMS
rms = np.sqrt(np.sum(np.square(signal))/n)
```

Figure 5. 30.- Equation to calculate RMS in Python. *Source: own source.*

- **Variance (VAR).** This feature calculates the dispersion of the data points from the mean of the signal. The higher the variance is, the more dispersed the values are. Broadly speaking, the variance equals the summation of the square subtraction between a value and the mean of the signal divided by the length of the signal but, as it can be seen in *Equation 11,* it is divided between the length of the signal minus one. This is due to the calculation being done to a sample population, not to the complete dataset [68]. For Python, the variance has been calculated with *NumPy, See Figure 5.31.*

$$VAR = \frac{1}{N-1}\sum_{n=1}^{N}(x(n) - \overline{x})^2 \qquad \textbf{(Eq. 11)}$$

```
#calculation of VAR
var = np.var(signal)
```

Figure 5. 31.- Function to calculate VAR in Python*. Source: own source.*

- **Difference Absolute Mean Value (DAMV).** This feature calculates the average absolute difference of two consecutive values of the signal divided by the length of the segment minus 1, as it is not the complete signal, what it measures. For its calculation in Python*, See Figure 5.32, NumPy* has been used to do the summation and the absolute value, for the

difference two variables *signal_1* and *signal_q* were defined for: one to pick a value and, the other one to pick the following value from the last.

$$DAMV = \frac{1}{N-1}\sum_{n=1}^{N-1}|x(n+1) - x(n)| \quad \textbf{(Eq. 12)}$$

```
#calculation of DAMV
damv = (np.sum(np.abs(signal_1 - signal_q)))/(n-1)
```

Figure 5. 32.- *Equation to calculate DAMV in Python.* Source: own source.

- **Difference Variance Version (DVARV).** This feature calculates the variance but applying the concept of the DAMV of calculating the difference between a value and its following*, See Equation 13*. In addition, it is divided by the length of the signal minus 2, instead of 1 as the other two features, this because it is a version of the difference variance [69]. In Python, a combination of VAR and DAMV equations has been used, *See Figure 5.33,* again with the help of *NumPy* and these two created variables.

$$DVARV = \frac{1}{N-2}\sum_{n=1}^{N-1}(x(n+1) - x(n))^2 \quad \textbf{(Eq. 13)}$$

```
#calculation of DVARV
dvarv = (np.sum(np.square(signal_1 - signal_q)))/(n-2)
```

Figure 5. 33.- Equation to calculate DVARV in Python. *Source: own source.*

- **Integrated Absolute of Second Derivative (IASD).** This feature calculates the relative change of the second derivative of the signal, this equation is effectively applied many times to filter designs and signal mapping. It finds the summation of the absolute difference between the second derivative of the signal values*, See Equation 15*. To do the calculation in Python*, See Figure 5.34,* two new variable have been created *signal_prima_1* and *signal_prima_q* to calculate the second derivative of the signal values and then apply it to the equation which uses *NumPy* to calculate the summation and the absolute value.

$$IASD = \sum_{n=1}^{N-2}|x'(n+1) - x'(n)| \quad \textbf{(Eq. 14)}$$

```
#Signal and signal2 second derivate
signal_prima = signal_1 - signal_q
signal_prima_1 = signal_prima[1:]
signal_prima_q = signal_prima[:-1]

#calculation of IASD
iasd = np.sum(np.abs(signal_prima_1 - signal_prima_q))
```

Figure 5. 34.- Equation to calculate IASD in Python. *Source: own source.*

- **Integrated Exponential (IE).** This last feature is used to amplify large values and to suppress the small ones by calculating the exponential value of the signal values and its summation*,*

See Equation 15. In Python, *NumPy* has an *exp* function to calculate the exponential value, *See Figure 5.35.*

$$IE = \sum_{n=1}^{N} \exp\left(x(n)\right) \qquad \textbf{(Eq. 15)}$$

```
#calculation of IE
ie = np.sum(np.exp(signal))
```

Figure 5. 35.- Equation to calculate IE in Python. *Source: own source.*

After having chosen the features and prepared the equations for them, they have been all summed up in a function called *preprocessing()*. This function runs through the list of each gesture and calculates these features every 100 values, a gesture measure length, *See Figure 5.36.* After the calculation the result is appended to a list, one for every feature, for an organised classification of features.

```
#THIS FUNCTION CALCULATES THE FEATURES
def preprocessing(gest_filtered):
    l_mav = []
    l_max_val = []
    l_min_val = []
    l_rms = []
    l_var = []
    l_damv = []
    l_dvarv = []
    l_iasd = []
    l_ie = []

    for i in range(0, len(gest_filtered), 100):
        signal = gest_filtered[i:i+100]
        n = len(signal)
```

Figure 5. 36.- Beginning of the preprocessing function. *Source: own source.*

The result of executing this function is a list of lists where each sub-list is the obtained values of a feature. To structure these features datasets in a proper way, data frames from the *pandas* library have been implemented. *Pandas* is a library that provides easy-to-use data structure and data analysis tools for Python [70]. One of those tools is the *DataFrame*, a two-dimensional table able to classify data from a dictionary, list or array.

Each of the gestures recording and feature extraction has been executed separately, which means there is a list of data per gesture, five data frames have been created, one per gesture*, See Figure 5.37.* Worth mentioning is that to properly organise the dataset, a few modifications needed to be executed. The first change was to transpose the table (swap rows and columns) due to the features being supposed to be columns and they were rows. Secondly, to attribute an ascending numbering starting from 1 in the table index, this was redefined. Finally, a last column called *gesture* was added to indicate the corresponding gesture number for each gesture.

```
df_1 = pd.DataFrame(preproc_1).T
df_1.index = np.arange(1, len(df_1)+1)
df_1['gesture'] = 1
```

Figure 5. 37.- Creating a Data Frame for each gesture. *Source: own source.*

The result Data Frame looks like in *Figure 5.38.* The information of each column corresponds to each feature. The last column indicates the gesture to which the information in the row corresponds and the index (first column), indicates in an ascending numbering starting from 1 the gestures recorded.

	0	1	2	3	4	5	6	7	8	gesture
1	0.052208	0.282777	-0.249303	0.076895	0.005913	0.057070	0.007053	6.264868	100.276285	2
2	0.032655	0.202214	-0.209697	0.054477	0.002968	0.034861	0.003606	3.802451	100.180930	2
3	0.046093	0.319474	-0.318693	0.087148	0.007595	0.052109	0.010775	6.285701	100.384192	2
4	0.100602	0.615045	-0.548375	0.177379	0.031463	0.095950	0.028664	9.513587	101.614282	2
5	0.088923	0.506025	-0.476011	0.150989	0.022797	0.093112	0.022530	10.233320	101.191753	2
..	...	...	...	...	...	...	...	...	...	...
104	0.024436	0.138669	-0.142657	0.042834	0.001835	0.023378	0.001799	2.419003	100.094479	2
105	0.050110	0.302069	-0.274845	0.088645	0.007858	0.060101	0.010957	7.288752	100.394343	2
106	0.034650	0.243511	-0.228018	0.059105	0.003493	0.037332	0.003973	4.187326	100.169542	2
107	0.027498	0.122143	-0.114753	0.043431	0.001886	0.028714	0.002184	3.107497	100.097449	2
108	0.042752	0.201049	-0.222176	0.069717	0.004860	0.051057	0.006964	6.131048	100.195989	2

Figure 5. 38.- Created Data Frame for each gesture. *Source: own source.*

After having created the five data frames, one per gesture, all have been merged into one only data frame which stores all the information from all gestures together. To do so, the command *concat* from pandas is used*, See Figure 5.39.* In addition, naming to the columns has been added to know which feature corresponds each column to.

```
frames = [df_1, df_2, df_3, df_4, df_5]

df_gest = pd.concat(frames)
df_gest.columns = ['MAV', 'MAX', 'MIN', 'RMS', 'VAR', 'DAMV', 'DVARV', 'IASD', 'IE', 'gesture']
```

Figure 5. 39.- Creating a Data Frame merging all gestures. *Source: own source.*

The result is a data frame five times longer than the individual ones, with labels in the columns and the measuring numbering concatenated*, See Figure 5.40.*

	MAV	MAX	MIN	RMS	VAR	DAMV	DVARV	IASD	IE	gesture
0	0.032918	0.093443	-0.091681	0.041656	0.001735	0.039027	0.002311	4.538561	100.139951	1
1	0.006941	0.035347	-0.030433	0.009220	0.000085	0.007692	0.000103	0.822210	100.010624	1
2	0.017169	0.048073	-0.042395	0.020505	0.000420	0.020081	0.000561	2.306655	100.006763	1
3	0.029682	0.087732	-0.094755	0.036422	0.001326	0.034028	0.001711	4.034169	100.140896	1
4	0.033177	0.124172	-0.111473	0.042551	0.001810	0.038644	0.002583	4.761480	100.028415	1
..	...	...	...	...	...	...	...	...	...	...
533	0.029186	0.141600	-0.147145	0.043986	0.001935	0.030772	0.002153	3.260201	100.098731	5
534	0.033420	0.170834	-0.165381	0.053201	0.002830	0.038462	0.003586	4.648517	100.150522	5
535	0.015748	0.050441	-0.053159	0.021201	0.000449	0.016588	0.000508	1.859826	100.022508	5
536	0.021522	0.084573	-0.095409	0.032712	0.001070	0.024381	0.001463	2.892454	100.048509	5
537	0.031168	0.132806	-0.121030	0.044087	0.001944	0.033658	0.002318	4.009287	100.058001	5

Figure 5. 40.- Created data frame for all the gesture. *Source: own source.*

Lastly, after having created the data frame, a Correlation Matrix was built to visualise the relation between features. This table displays the correlation coefficients between variables, in this case the features. Each cell of that table shows the correlation between its row and column feature. Correlation matrices are used as diagnostics or inputs for advanced analyses [71]. These tables are created with *pandas*' function *corr()* and to get a visual representation from the table, the *seaborn* [72] and *matplotlib* [73] packages can be used. Those libraries are helpful for creating static, animated and interactive data visualisation in Python.

The first thing that has been done to create this matrix is removing the gesture column from the data frame, as it is non-relevant information for the features correlation*, See Figure 5.41.* After that, the matrix and its visual representation have been created.

```
df_gest = df_gest.drop(columns=['gesture'])
corrMatrix = df_gest.corr()
sns.heatmap(corrMatrix, annot=True)
plt.show()
```

Figure 5. 41.- Correlation Matrix creating and visualising commands. *Source: own source.*

The obtained result, *See Figure 5.42,* is a matrix with a correlation factor between -1 and 1. All diagonal cells are 1 because they show the correlation of each variable itself. The cells near to 1 indicate a strong correlation, which means that both features are giving similar information and that the growth of one feature implies the growth of the other one. All cells with negative values close to -1, indicate the opposite as mentioned before, prove a negative correlation and, therefore, the increase of the value of a feature implies the decrease of the other one (i.e., MAX and MIN, they are the opposite, the higher MAX is, the lower MIN will be). And the closer a cell is to 0, the less correlation between features and they can be considered independent from each other and, therefore, they both give relevant and differentiative information. As it can be observed in the figure, many cells are close to 1, which means that those features are giving too similar and non-relevant information.

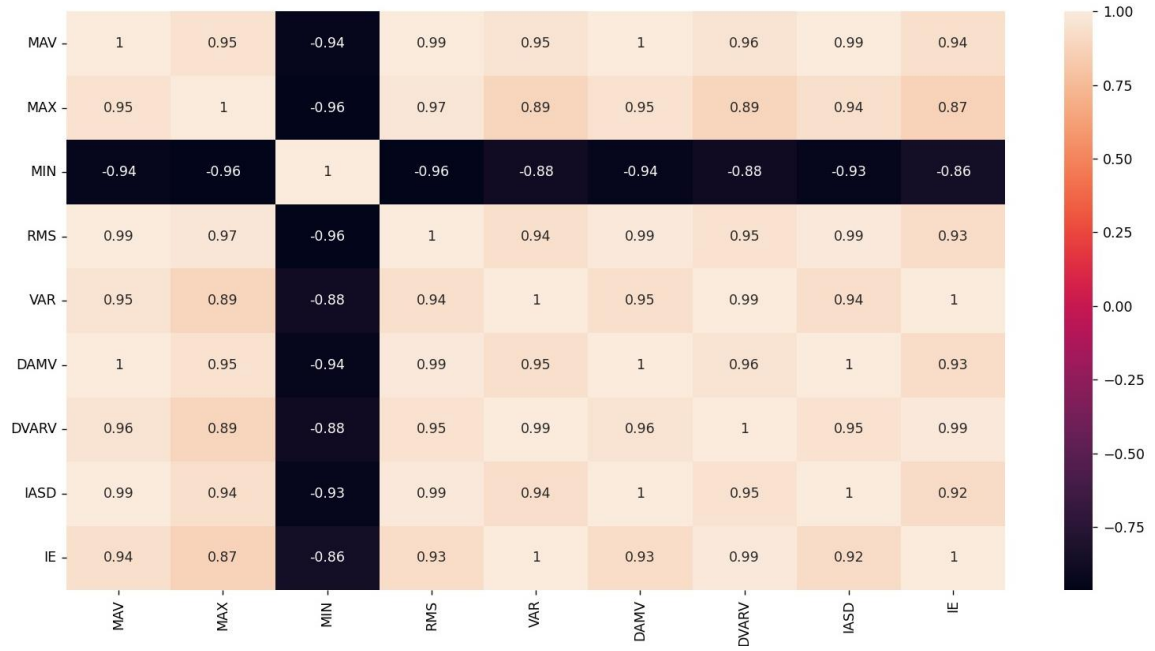


Figure 5. 42.- Obtained correlation matrix. *Source: own source.*

To try to improve this correlation between features, the data has been sampled into two parts to get the double number of features as before and, theoretically, a smaller correlation factor between features.

The first step was to split the data in a proper way, it was not just taking the whole dataset and easily splitting it into 2, that would make no sense as both feature sets would still be measuring the same, a whole gesture. The division needed to be done inside a gesture, which means to divide into two parts the measure of a gesture (100 data values). This way there would be features of the first half of the gesture recording and of the second half of it.

Then, to have this data split, the function *preprocessing* needed to be adapted to this new requirement, check Attachment C3. For it, three changes needed to be done in the function: first, the function must calculate features in groups of 50 values instead of 100 as before*, See Figure 5.43.*, and it need to be done twice, on one side for the first 50 values of the gesture and, on the other side for the second 50 values of the same gesture. For it, a second variable, *signal_2*, has been created. *Signal* will calculate all the features from the first 50 values and *signal_2* for the second 50. The second change is that new lists need to be created besides the original ones, *See Figure 5.44.* These new lists will store the values of the features of the second half of the gesture, while the original lists will store the values of the first half. The last change was to duplicate every feature calculation, so it could send the corresponding information to one list or to another, *See Figure 5.45.*

```
for i in range(0, len(gest_filtered), 100):
  signal = gest_filtered[i:i+50]
  signal2 = gest_filtered[i+50:i+100]
  n = len(signal)
```

Figure 5. 43.- Loop and variables to calculate features with the data split. *Source: own source.*

```
def preprocessing_2(gest_filtered):
  l_mav = []
  l_max_val = []
  l_min_val = []
  l_rms = []
  l_var = []
  l_damv = []
  l_dvarv = []
  l_iasd = []
  l_ie = []
  l_mav2 = []
  l_max_val2 = []
  l_min_val2 = []
  l_rms2 = []
  l_var2 = []
  l_damv2 = []
  l_dvarv2 = []
  l_iasd2 = []
  l_ie2 = []
```

Figure 5. 44.- Lists that store the calculated features. *Source: own source.*

```
#calculation of MAV
mav = np.sum(np.abs(signal))/n
l_mav.append(mav)

mav2 = np.sum(np.abs(signal2))/n
l_mav2.append(mav2)
```

Figure 5. 45.- MAV calculation for the split data. *Source: own source.*

Once this new function, *preprocessing_2()* has been executed, a new data frame has been created for it, in the same way as before, *See Figure 5.46* and *Figure 5.47.*

```
#concatenates df with all values
frame_all = [df3_1, df3_2, df3_3, df3_4, df3_5]
df_all = pd.concat(frame_all)
df_all.columns = ['MAV', 'MAX', 'MIN', 'RMS', 'VAR', 'DAMV', 'DVARV', 'IASD', 'IE', 'MAV 2', 'MAX 2', 'MIN 2', 'RMS
df_all = df_all.reset_index(drop=True)
```

Figure 5. 46.- Commands to create the new data frame. *Source: own source.*

	MAV	MAX	MIN	RMS	VAR	DAMV	DVARV	IASD	...	MIN 2	RMS 2	VAR 2	DAMV 2	DVARV 2	IASD 2	IE 2	gesture
0	0.078030	0.170499	-0.218352	0.092097	0.008396	0.107301	0.016864	8.575939	...	-0.169905	0.083766	0.007004	0.112584	0.018751	9.793405	49.993182	1
1	0.016834	0.041953	-0.045173	0.019777	0.000389	0.027189	0.001035	2.439928	...	-0.039609	0.018126	0.000328	0.016639	0.000547	1.248977	50.050444	1
2	0.029539	0.070071	-0.070066	0.034506	0.001179	0.037559	0.002236	2.845440	...	-0.059374	0.035625	0.001266	0.040054	0.002414	3.219278	50.117754	1
3	0.058828	0.103162	-0.125231	0.066256	0.004390	0.076740	0.008638	5.998386	...	-0.105218	0.059041	0.003485	0.064235	0.006558	5.175441	50.128104	1
4	0.079789	0.134764	-0.169988	0.090333	0.008157	0.126759	0.021882	10.623643	...	-0.153246	0.088229	0.007784	0.101753	0.015293	8.384114	50.234996	1
..	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
533	0.046398	0.407108	-0.191620	0.084907	0.007208	0.064467	0.015282	5.358110	...	-0.274902	0.076968	0.005922	0.073697	0.012135	6.212486	50.220514	5
534	0.030453	0.247118	-0.237816	0.065974	0.004347	0.042013	0.008339	2.903806	...	-0.363228	0.102101	0.010416	0.099345	0.019779	7.709776	50.112897	5
535	0.020410	0.110740	-0.124683	0.033071	0.001091	0.022759	0.001910	1.874157	...	-0.145020	0.074438	0.005540	0.077554	0.010260	6.423118	50.196292	5
536	0.042285	0.185002	-0.259720	0.074343	0.005527	0.060236	0.010960	4.777741	...	-0.185637	0.053836	0.002892	0.052584	0.006654	4.260954	50.192544	5
537	0.047508	0.163062	-0.175927	0.067973	0.004611	0.059382	0.008496	4.501955	...	-0.149936	0.087309	0.007618	0.078281	0.015676	6.851011	50.089168	5

Figure 5. 47.- New created data frame. *Source: own source.*

The result of this new data frame represented in a new correlation matrix, *See Figure 5.48,* is much better and more like what was expected to obtain. What has been obtained splitting the data is a double amount of feature values for each gesture repetition (two MAV values, two MAX values, two MIN values, etc.), which is reflected in better quality of features for a more accurate model.

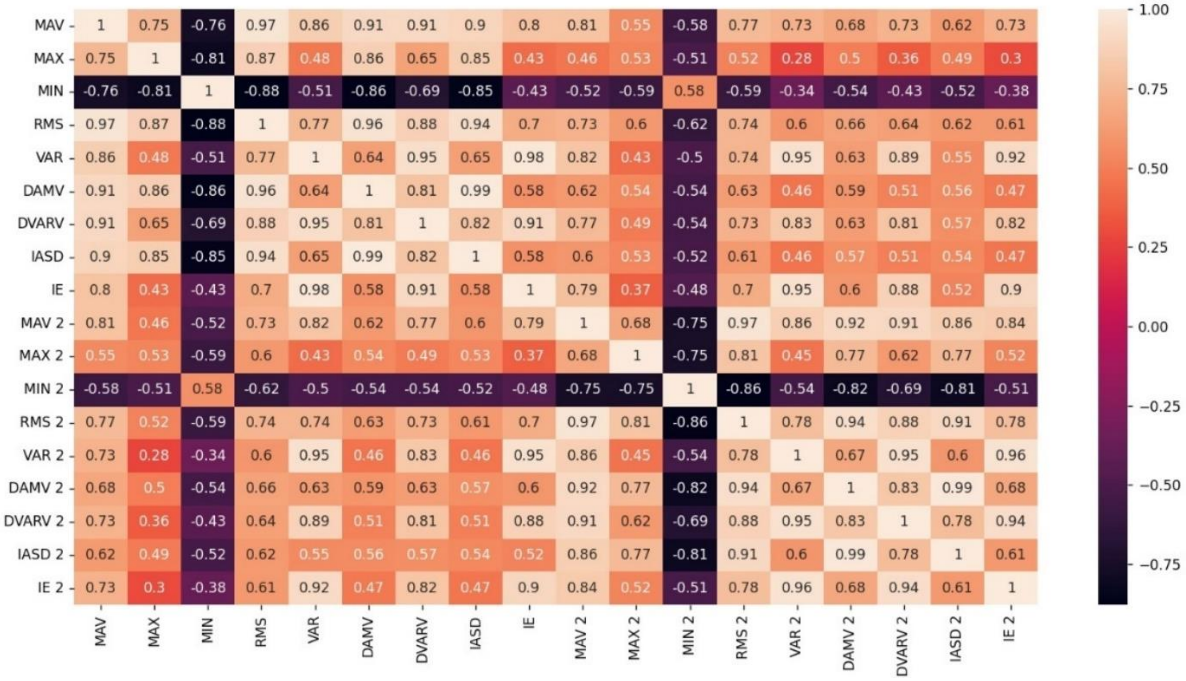


Figure 5. 48.- New correlation matrix. *Source: own source.*

After seeing the positive results of splitting the signal, the first idea that comes to mind is on splitting the data again, so instead of having a gesture repetition split into 2 parts, splitting it, for example, into 4 parts. This next step would make sense if a gesture had more data values, in other words, as a gesture repetition only consists of 100 data values, splitting this segment into 4 parts would mean having 4 segments of 25 data values each. This leads to two main problems, the first one is that having these segments with so few values will end up returning overlapped and non-relevant features. The second problem is that as the measuring consists of a rest-execute-rest process, the first and the last segments would belong to the rest part which should not provide relevant information. To confirm it, it has been tested and the result, *See Figure 5.49*, is what was expected to obtain, a matrix with high correlation values between features.

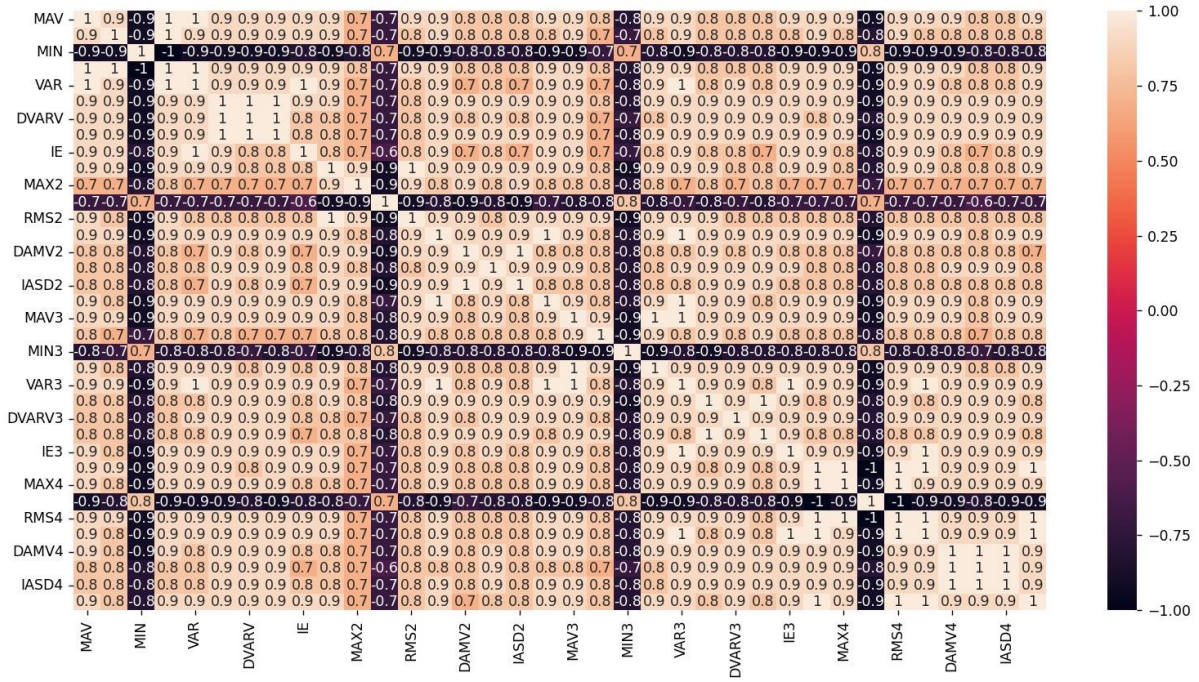


Figure 5. 49.- Correlation matrix with the signal split into 4 parts. *Source: own source.*

Lastly, before proceeding with the classification models, a small recollection of the obtained features has been designed. The aim of this is to have a visual comparison of the features from each of the both segmentations and be able to analyse the mean of the obtained features for each gesture, *See Table 5.1*. This has been obtained after executing a short script, *compare.py*, which starts reading the data frame with all the features and calculates the mean of each feature divided by the number of the gesture it is, *See Figure 50*. The entire code can be checked on the Attachment C4.

```
3   df = pd.read_csv('data.csv')
4
5   df = df.groupby('gesture',as_index= False)['MAV', 'MAX', 'MIN', 'RMS',
6   print(df)
```

Figure 5. 50.- Code to create the comparison between features. *Source: own source.*

	Gesture	MAV	MAX	MIN	RMS	VAR	DAMV	DVARV	IASD	IE
First 50 values	**1**	0,021	0,058	-0,058	0,026	0,002	0,023	0,002	1,810	50,047
	2	0,091	0,411	-0,411	0,145	0,028	0,118	0,050	9,415	50,643
	3	0,072	0,381	-0,336	0,117	0,018	0,097	0,033	7,684	50,482
	4	0,095	0,458	-0,493	0,155	0,028	0,126	0,052	10,068	50,684
	5	0,055	0,255	-0,254	0,085	0,010	0,073	0,017	5,790	50,176
Last 50 values	**1**	0,022	0,096	-0,061	0,030	0,003	0,026	0,004	2,008	50,116
	2	0,068	0,316	-0,330	0,108	0,015	0,085	0,025	6,685	50,404
	3	0,062	0,308	-0,288	0,098	0,012	0,080	0,021	6,335	50,316

	4	0,089	0,423	-0,434	0,142	0,024	0,117	0,043	9,268	50,659
	5	0,060	0,226	-0,250	0,086	0,009	0,078	0,015	6,137	50,277

Table 5. 1.- Comparison of the obtained features.

Figure 5. 51.- Visualisation of the content of Table 5.1. *Source: own source.*

Figure 5.51 is a visual representation of the content of *Table 5.1.* where the blue series represent the features for the first 50 values for each gesture and the orange lines represent the features for the last 50 values for each gesture. With this, it has been pretended to observe typical characteristics from the features and the segments. As it can be observed, the results of each segment have similar tendencies, as it was expected. For example, the mean of the maximum values (MAX) and minimum values (MIN) have similar but opposite values for its corresponding gestures. It is worth mentioning that the IASD and the IE have been divided into 10 and 100 correspondingly to fit better the figure.

With this ends the features extraction section and, with it, the signal pre-processing. Now the data is ready to be trained in a Machine Learning classification model.

5.3.3. Training models

After having pre-processed the signal and having organised and stored it as wished for a proper use, the training of the model is able to start. As it has been explained in earlier sections, among the most used models for supervised learning classification are logistic regression, random forests, Support Vector Machines (SVM), Naïve Bayes or K-Nearest Neighbours. To know which models, must be used, the information of the flowchart given by scikit-learn [74] has been used, *See Figure 5.52*. The logistic regression model does not appear in the flowchart because it is a classification model that is always taken into consideration as the starting point. To sum up, for this classification case, the models that fit the most due to the amount of data measured and the type of data dealing with are: Logistic Regression, Support Vector Classifier and K-Nearest Neighbours.

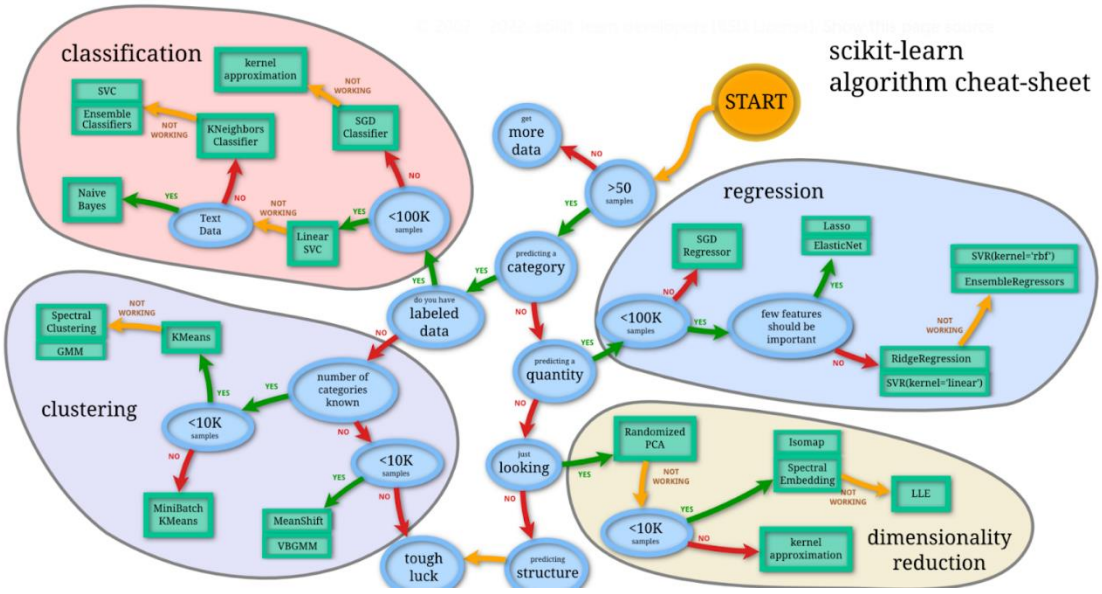


Figure 5. 52.- Flowchart to decide which ML model to use. *Source: scikit-learn.org*

To write the algorithms of each model, the library *scikit-learn* has been used [75]. This is the most famous Machine Learning library for Python, it allows to write classification, regression, clustering, dimensionality reduction, model selection and pre-processing models in a simple and efficient way.

To measure the accuracy of the models, the train-test method has been used. This method consists on splitting the data into a training set and a testing set. Then the model is trained using the training set and, subsequently, tested with the testing set. The amount of data that belongs to the training set and to the data set are not specific and should be chosen depending on the project's objectives. For this case, as it a model with a non-excessive amount of data and without big computational costs in

training nor testing, the distribution could have been done as wished. For it, two train-test distributions have been used during the entire process to evaluate its accuracy as the data amount was growing. The first distribution was 80% training and 20% testing and, the second distribution was 67% training and 33% testing [76].

Once this is defined, the next step is to split the data into the input data (X) and the output data (y), *See Figure 5.53*. For this case, the input data are all the features from the data frame and the output data the gesture number. The first to be done is import the obtained data frame from the pre-processing, for it a *pandas* function *read_csv* has been used. Obviously, the data frame has been saved into a .csv file on the preprocessing script, using the *to_csv* function. Then, the values of the gestures are store in the y variable and the features are stored in the X variable*, See Figure 5.54.* To get only the values of the gestures in variable y, the function *pop* has been used. This function returns the items and drops them from the data frame, leaving so a data frame without the gesture column for the variable X.

```
df = pd.read_csv('data.csv')

y = df.pop('gesture').values
X = df.values
```

Figure 5. 53.- Splitting data in input and output. *Source: own source.*

```
[[ 7.80300373e-02  1.70502212e-01 -2.18349166e-01 ...  1.87512603e-02
   9.79340494e+00  4.99933255e+01]
 [ 1.68333912e-02  4.19554778e-02 -4.51696741e-02 ...  5.46601817e-04
   1.24897658e+00  5.00505874e+01]
 [ 2.95385981e-02  7.00735793e-02 -7.00628068e-02 ...  2.41354669e-03
   3.21927804e+00  5.01178980e+01]
 ...
 [ 4.28679081e-02  3.82326440e-01 -2.15370763e-01 ...  3.03922629e-02
   9.90051073e+00  5.01885571e+01]
 [ 4.47312568e-02  1.65819406e-01 -2.94901265e-01 ...  1.01949734e-02
   5.27467101e+00  5.02360114e+01]
 [ 4.80572368e-02  2.17260587e-01 -1.75222737e-01 ...  4.82671162e-03
   3.18621189e+00  4.94644866e+01]]
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
 3 3 3 3 3 3 3 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4 4
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5 5]
```

Figure 5. 54.- X variable (top) and y variable (down). *Source: own source.*

After having the data split in inputs and outputs, the train-test method is defined. For it, the *scikit-learn* function named *train_test_split* is used, *See Figure 5.55.* Both the X and y variables are divided

into train and test, the size of the test is specified in the *test_size* (must be a value in the range between 0 and 1) and then, a *random_state* is specified. This is due to the *train_test_split* splits the data randomly every time the model is run and, as it is wished to execute the model with different learning methods and different test sizes, it should always be tested with the same data. To do so, a *random_state* (42) is specified and it will be used for all different models.

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
```

Figure 5. 55.- Train-test function with 33% of data in the test set. *Source: own source.*

The next step that has been done is defining a *pipeline*. This class allows sticking different processes into a single estimator and keeping the code much cleaner and compressed. This pipeline is saved as a variable *pipe*, *See Figure 5.56.* The first added to the pipeline is a scaling function *StandardScaler*, this function subtracts the mean from each feature and scales it to unit variance [77]. Then second component added is the training model (for the figure example it is Logistic Regression). With the *fit* model, the training of the model is executed. It also allows to make predictions and evaluate the model.

```
pipe = Pipeline([('scaler', StandardScaler()), ('log', LogisticRegression())])
pipe.fit(X_train, y_train)
```

Figure 5. 56.- Creating pipeline. *Source: own source.*

The following step is to check the accuracy*, See Equation 16,* of the model. This is defined as the fraction of right predictions of the model. For it, the *predict* function is used. The accuracy of a model is usually checked in the test set because it shows the model's generalisation performance and represents well the reliability of the model so, the predictions will be applied on the *X_test* variable*, See Figure 5.57.* Then, these predictions are evaluated by using the *accuracy_score* command. This checks the accuracy by comparing the *y_test* and the *y_pred* (predicted values).

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Tota\ number\ of\ predictions} \qquad \textbf{(Eq. 16)}$$

```
y_pred = pipe.predict(X_test)

accuracy1 = accuracy_score(y_test, y_pred)
```

Figure 5. 57.- Making and evaluating predictions. *Source: own source.*

As mentioned earlier, different classification models have been created. The only change between models is the class being used and the component written down into the *pipeline*, *See Figure 5.58.*

```
pipe = Pipeline([('scaler', StandardScaler()), ('log', LogisticRegression())])

pipe = Pipeline([('scaler', StandardScaler()), ('svc', SVC())])

pipe = Pipeline([('scaler', StandardScaler()), ('knn', KNeighborsClassifier(n_neighbors=5))])
```

Figure 5. 58.- Pipelines of the three models used. *Source: own source.*

A last mention, it is observable in *Figure 5.53.,* that the K-Neighbours *pipeline* has an extra element compared to the other ones. This element, *n_neighbors,* is the number of neighbours being used in the model and, as there are five gestures to be classified, the correspondent value for this *n_neighbors* element is 5.

With this, the training of the model is complete. Now what can be done is to calculate different elements to evaluate the efficiency and accuracy of the model in a more comprehensive way. Many things can be used to check this but, in this case, the **accuracy of the training set**, a **classification report**, a **confusion matrix** and a **randomness evaluation** will be used.

The **accuracy of the training set** shows how the model can classify gestures during the training, with the training set. This is a helpful metric because it helps to detect if there is an overfitting in the data set. Overfitting refers to an excess of non-relevant features that give excessive similar information. To avoid the overfitting, the correlation matrix has been built previously but, it is important to check it again when it comes to the practical part. The process to calculate this accuracy is the same as for the test set but changing the *X_test* for the *X_train* as input. If the obtained result is a value that differs from the test accuracy, there is an overfitting problem in the model.

Before proceeding to the **classification report** and the **confusion matrix**, four concepts need to be explained: True Positive (TP), False Positive (FP), False Negative (FN) and True Negative (TN)*, See Figure 59.* The figure shows a binary confusion matrix which allows the visualisation of the performance of an algorithm [78].

- **TP:** the test result indicates the presence of condition when there is one. For example, when a test for a disease shows positive and the patient has that disease.

- **FP:** the result indicates the presence of condition when there is none. For example, when a test for a disease shows up false when there is no disease.

- **FN:** opposite as FP, this time the result indicates no presence of condition when there is. For example, when a test for a disease shows up negative but the patient does have that disease.

- **TN:** the test result indicates the presence of condition when there is none. For example, when a test for a disease shows up positive when the patient does not have that disease.

Manual counting / Machine learing	True	False
True	True Positive (TP)	False Positive (FP)
False	False Negative (FN)	True Negative (TN)

Figure 5. 59.- Binary confusion matrix. *Source: researchgate.com*

The **classification report** is a famous performance evaluator of metrics usually used for classification problems. This report returns a table that displays the following performance metrics of the model [79]:

- **Precision.** Is the proportion of positive correct identifications and is calculated as the ratio of true positives to the summation of true and false positives, *See Equation 17.*

$$Precision = \frac{TP}{TP+FP} \qquad \textbf{(Eq. 17)}$$

- **Recall.** Is the proportion of real positives identified as it and is calculated as the ratio of true positives to the summation of true positives and false negatives*, See Equation 18*.

$$Recall = \frac{TP}{TP+FN} \qquad \textbf{(Eq. 18)}$$

- **F1 score.** Is the weighted harmonic mean of precision and recall, *See Equation 19*. The closer this value is to 1, the better the model's performance is.

$$F1 - score = \frac{2*precision*recall}{precision**+recall} \qquad \textbf{(Eq. 19)}$$

- **Support.** This value just diagnoses the performance evaluation process and is the number of occurrences of a class in the dataset.

To create the **classification report**, the *classification_report* class from *scikit-learning* is used and the *y_test* and *y_pred* need to be added as inputs*, See Figure 5.60*.

```
print(classification_report(y_test, y_pred))
```

Figure 5. 60.- Show classification report. *Source: own source.*

The obtained result*, See Figure 5.61.,* is a table with all the performance metrics calculated for each class, in this case for each gesture. This helps to visualise which of the classes are performing better and which are performing worst.

	precision	recall	f1-score	support
1	0.87	0.93	0.90	56
2	0.45	0.49	0.47	45
3	0.42	0.33	0.37	46
4	0.45	0.41	0.43	41
5	0.46	0.51	0.48	41
accuracy			0.55	229
macro avg	0.53	0.53	0.53	229
weighted avg	0.55	0.55	0.55	229

Figure 5. 61.- Output of the classification report. *Source: own source.*

A **confusion matrix** allows to visualize the performance of the classification model and get a better idea of the performance of the model. The matrix shows the time a gesture has been well predicted and the times it has not, it also shows how many times, and with which other gesture, a gesture has been misclassified, *See Figure 5.63.*

To create the matrix, the *confusion_matrix* class from *scikit-learning* has been used. It only requires to add the same inputs as for the classification report*, See Figure 5.62.*

```
conf_mat = confusion_matrix(y_test, y_pred)
print(conf_mat)
```

Figure 5. 62.- Confusion matrix command. *Source: own source.*

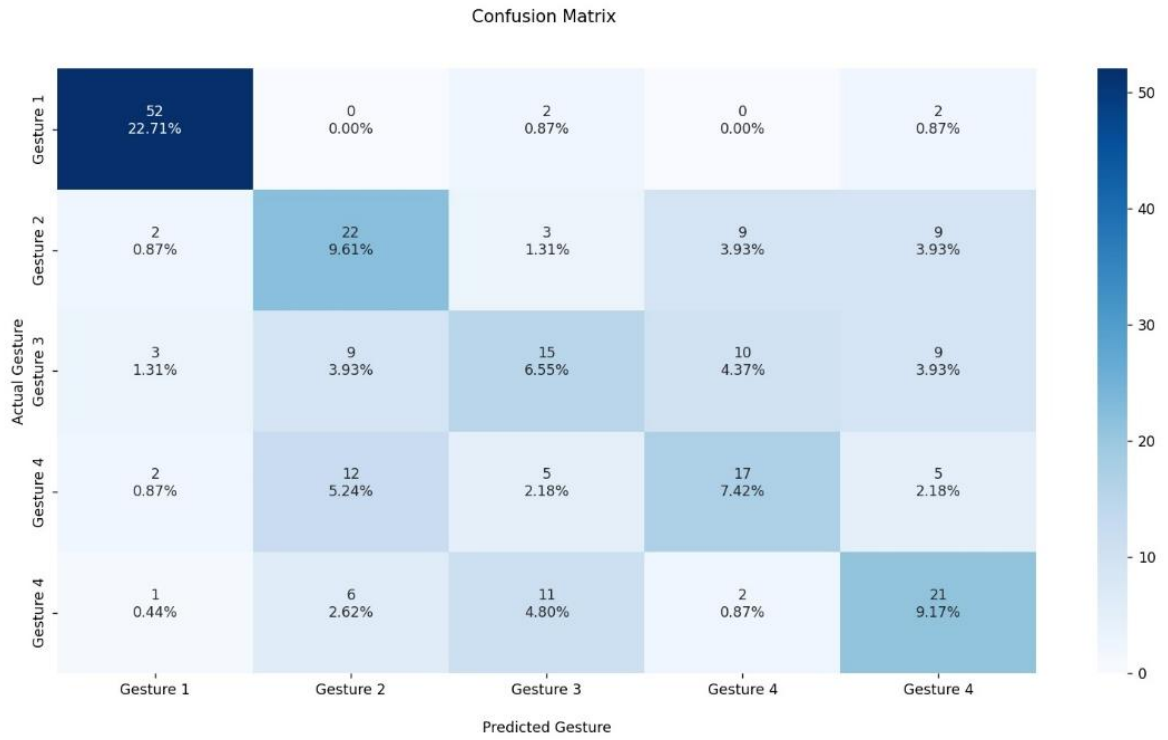


Figure 5. 63.- Visual result of a confusion matrix. *Source: own source.*

The last method used to check the reliability of the model has been a **randomness evaluation**. This evaluation pretends to demonstrate the precision of the model by testing how close it gets to the theoretical random accuracy value. This means, as there are five gestures to classify, the randomness is considered as an accuracy of 20% (100%/5) so, the closer the obtained accuracy is to 20%, the better precision does the model have. To test it, the accuracy of a random value has been calculated*, See Figure 5.64.* The model prints a random value from 1 to 5*, y_random,* and executes it as many times as *y_test* times the number appears on the set and it finally predicts its accuracy.

```
y_random = [randint(1,5) for _ in range(len(y_test))]
accuracy3 = accuracy_score(y_test, y_random)
print(accuracy3)
```

Figure 5. 64.- Randomness evaluation. *Source: own source.*

With all these elements, it should be enough to test the efficiency and reliability of the model and to learn from its results which parts from the model, signal processing or measures can be improved.

5.3.4. Real-time evaluation

One of the objectives of this project was to design a real-time evaluation of the most efficient Machine Learning classification model. To obtain that, two codes have been created which allow this

real-time classification from a real-time measure imported directly from Firebase and, that one, coming from the sEMG. Both complete codes can be found in the Attachment C8.

The first code, *real_model.py*, is an adapted version of the most efficient classification model to a real live implementation. The difference with the testing model is that no train-test is needed. As the model is being implemented there is no more need of training and testing the model to check its accuracy, as its accuracy and evaluation elements are already known and checked. The only needed for this code is the classification model, Logistic Regression*, See Figure 5.65.*

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

df = pd.read_csv('data.csv')
y = df.pop('gesture').values
X = df.values

pipe = Pipeline([('scaler', StandardScaler()), ('log', LogisticRegression(random_state=0, max_iter=1000))])
pipe.fit(X, y)

import pickle
file_pipe = open('log_reg.pkl', 'wb')
pickle.dump(pipe, file_pipe)
```

Figure 5. 65.- Code of the real model. *Source: own source.*

The second code, *evaluation.py,* englobes all the processes that have been described and explained so far. It is like a recapitulation of the whole section of this report regarding Python. The code is divided into two blocks: the **signal processing** and the **signal classification**.

Signal processing*, See Figure 5.66***.** This part of the code connects to the online real-time database from Firebase, then extracts the values from the directory of the database where the measures are stored and, then, it executes all the functions from the pre-processing section. It calculates the features, creates the data frame and then starts the classification block.

```
firebaseConfig = {
  "apiKey": "AIzaSyAwBffvO_waRFkEFjO8tJpDE4z8erMECQ0",
  "authDomain": "aehrich-database.firebaseapp.com",
  "projectId": "aehrich-database",
  "databaseURL": "https://aehrich-database-default-rtdb.europe-west1.firebasedatabase.app/",
  "storageBucket": "aehrich-database.appspot.com",
  "messagingSenderId": "710072352888",
  "appId": "1:710072352888:web:7aace7bb41b5847dcc6d8d"
}

firebase = pyrebase.initialize_app(firebaseConfig)
db = firebase.database()
data = db.child("EMGdata").child("evaluation").get() #--> gets all the content of the node (path)
gest= []
for val in data.each():
  gest.append(val.val())

gest = fnc.correct_length(gest)
time = np.array([i/1000 for i in range(0,len(gest),1)])
gest1= fnc.rmv_mean(gest,time)
gest_filtered = fnc.bp_filter(gest, 20, 450, 1000, plot=False)
gest_filtered = fnc.notch_filter(gest_filtered, 1000, plot=False)
preproc = fnc.preprocessing_2(gest_filtered)
df = pd.DataFrame(preproc).T
df.index = np.arange(1, len(df)+1)
df.columns = ['MAV', 'MAX', 'MIN', 'RMS', 'VAR', 'DAMV', 'DVARV', 'IASD', 'IE']
df_gest = df.reset_index(drop=True)
```

Figure 5. 66.- First block of the evaluation code. *Source: own source*.

Signal classification*, See Figure 5.67.* After having stored the features in the data frame, it defines the X variable (values of the data frame) and then calls the classification model created for the evaluation. This is done with the *pickle* module [80]. It is the standard way of serialising objects in Python and it can be also used to serialise a Machine Learning algorithm and save it to a file so it can later be loaded on another code to continue making predictions. After deserializing the model, it is executed and a prediction of the gesture is made. Once the model predicts the gesture that has been recorded, an image of the predicted gesture appears on screen, indicating the number of the gesture and its name, *See Figure 5.68*. After that, the terminal prints a list with the probabilities that the model would have classified the measured gesture as any of the five gestures, *See Figure 5.69*. This last information is helpful to detect where the model fails the most in each gesture and by which other gesture confuses it. The last step is removing the content from the Firebase database branch where the measured data was stored and put it on 0 to enable starting a new real-time evaluation.

```
47  X = df.values
48
49  filehandler = open('log_reg.pkl', 'rb')
50  pipe = pickle.load(filehandler)
51  y = pipe.predict(X)
52  y2 = pipe.predict_proba(X)
53
54  print('Gesture: ' + str(y))
55
56
57
58  np.set_printoptions(precision=2, suppress=True)
59  print('The probability of having predicted each gesture is: ')
60  print(y2*100)
61
62
63  db.child('EMGdata').child('evaluation').remove()
64  db.child('EMGdata').child('evaluation').set(0)
```

Figure 5. 67.- Second block of the evaluation code. *Source: own source.*

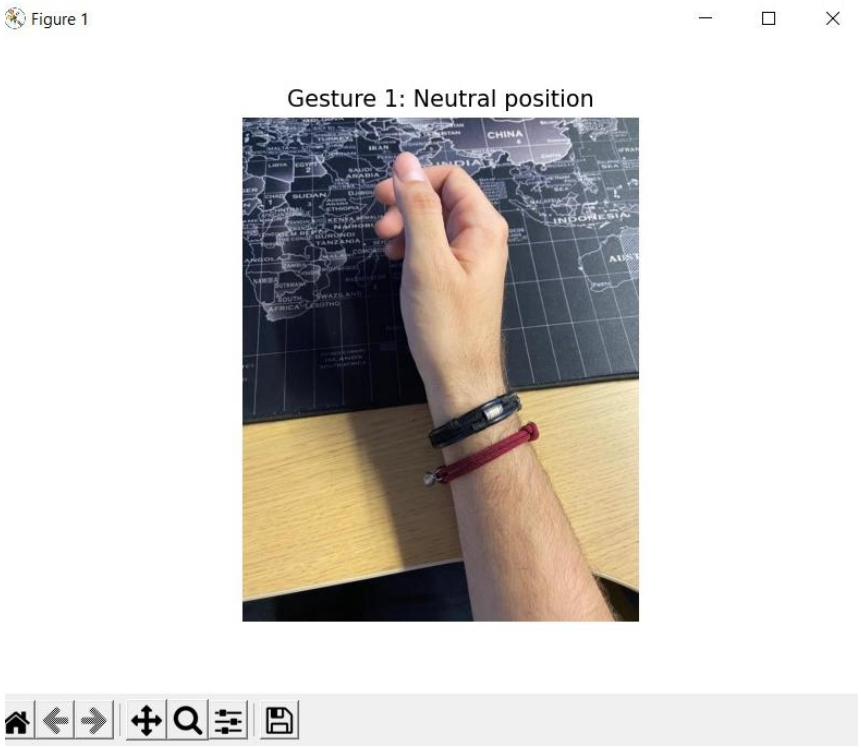


Figure 5. 68.- Output of the evaluation code, showing an image with the predicted gesture. *Source: own source.*

```
Gesture: [1]
The probability of having predicted each gesture is:
[[96.69  0.67  0.9   0.02  1.71]]
```

Figure 5. 69.- Output of the probabilities of predicting each gesture. *Source: own source.*

5.4. **Results**

After having designed and tested all the models, it is time to extract results of them and analyse and compare them to conclude which of the models suits best to the problems requirements. For it, each model will be evaluated separately and, at the end, all three models' evaluations will be compared, so conclusions can be extracted from those evaluations.

But first, a mention about the measured data needs to be done. The goal of the project was to measure an amount of data between 30 and 40 patients which, reflected on data values, meant a total amount between 90.000 and 120.000 data values. The final number of measured patients has been 30, which means that there have been 90.000 data values measured to apply to the classification models. So, whenever it is mentioned the 100% of measured data, it refers to this 90.000 measured values.

A last mention regarding the randomness evaluation is that this element was designed to check the precision of each model for each dataset test-size when it came to analyse randomness and its corresponding results are listed in the attachment, but they have not been considered as a relevant information to end up choosing one model or other.

Attachment C10 shows screenshots from all the obtained results and confusion matrices for all the models.

5.4.1. Logistic Regression

This model was the first one designed (and can be checked on Attachment C5), when 20% of the whole data set was measured. Once the classification model was designed, its accuracy was measured as more data was measured, exactly every 5% increase in data, the accuracy of the model was re-measured. With this, the aim was to record the evolution of the model to evaluate if the accuracy was growing as the data set was growing. This evolution has been recorded with a test size of 20%, on one side and with a test-size of 33%, on the other side. So, it could also be checked which test size obtains better results.

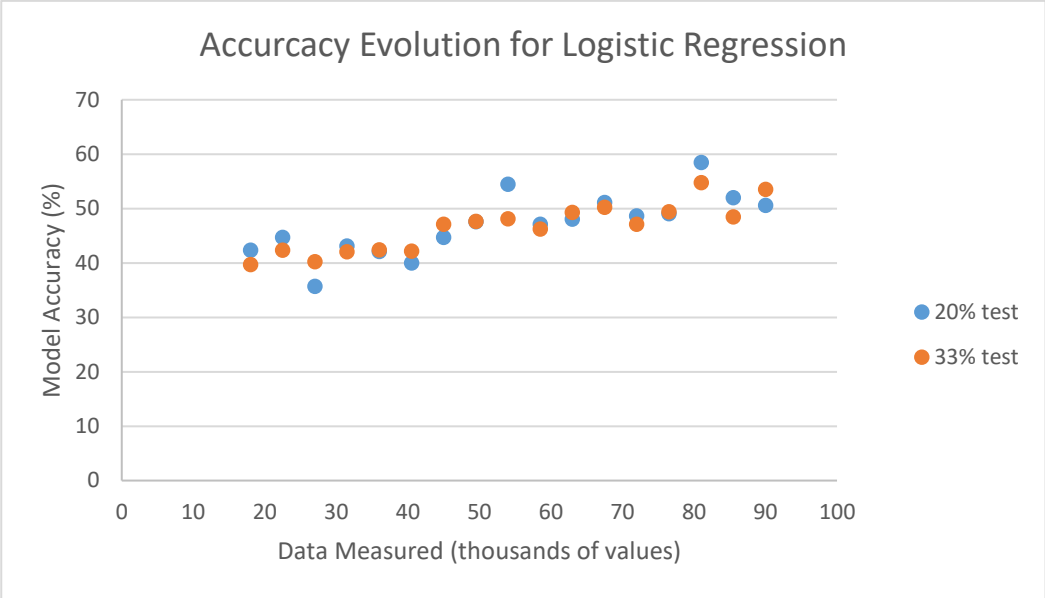


Figure 5. 70.- Visualisation of the evolution of the accuracy for the logistic regression model. *Source: own source.*

The *Figure 5.70*, shows a visual representation of the previous mentioned evolution of the accuracy of the logistic regression model with two different test sizes. The first observable fact is that as the amount of measured data increases, the accuracy does also increase. Although it is not a constant increase, as it has its difficulties, it can be confirmed that the accuracy of the model increases when the amount of data increases. Regarding to the test sizes, the 33% test size has a minim accuracy of 39,68% and a maximum accuracy of 54,74%. Compared to the 20% test size accuracies, 35,71% as the minimum accuracy and 58,43% as the maximum accuracy, its accuracy can be considered as more constant growing as the 20% test size, but with a less accurate peak performance. Although the growth of the 33% test size accuracy is more constant, what is searched in a model is the effectiveness and, the one that has the maximum accuracy is the 20% test size. It can be expected for the model to continue increasing its accuracy as long as the dataset gets bigger.

Despite this, the relevant value once all the measures are done is the one with the 100% of measured data, and that values are 50,56% and 53,54% of accuracy for the 20% and 33% test size respectively. The next step has been to evaluate those accuracies applying the earlier mentioned elements. *Table 5.2* shows the accuracies and the random evaluation for both sizes, an accuracy gap has been added, which is the subtraction of both accuracies. *Table 5.3* shows the classification report with the most relevant information.

Logistic Regression	**Test-size**	
	20%	**33%**
Test Accuracy (%)	50,56	53,54

Train Accuracy (%)	54,46	53,07
Accuracy Gap (%)	3,90	0,47

Table 5. 2.- Result table for the logistic regression model evaluation.

Logistic Regression		**Test-size**									
		20%					**33%**				
Gesture		**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
Classification Report	*Precision*	0,76	0,52	0,26	0,47	0,42	0,75	0,60	0,40	0,50	0,40
	Recall	0,91	0,37	0,17	0,50	0,62	0,93	0,44	0,21	0,55	0,61
	F1-score	0,83	0,43	0,20	0,48	0,50	0,83	0,50	0,28	0,52	0,48

Table 5. 3.- Classification report for the logistic regression model.

5.4.2. Support Vector Classifier (SVC)

The SVC, subgroup of the SVM, is the second classification model designed after the logistic regression (it can be checked on Attachment C6). Even though it was developed later than the first model, the aim of observing the development of accuracy as the data set was growing was also there. So, for this model, the evolution of the accuracy starts with the data set on 25% of the total measures recorded and has also been tested every 5% increase of the data. Again, it has been executed for a test-size of 33%, and for a test-size of 20%.

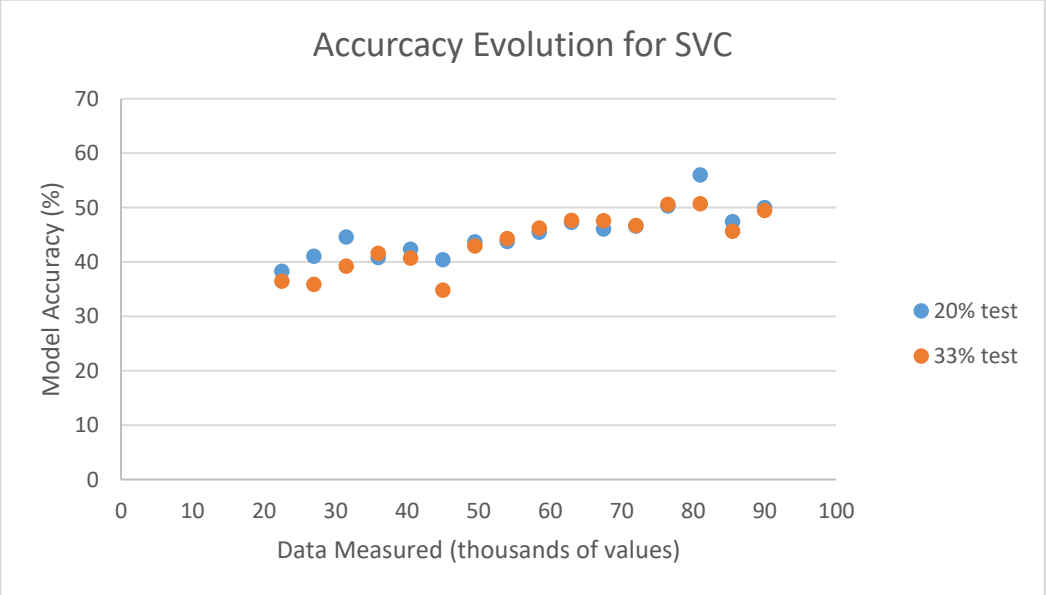


Figure 5. 71.- Visualisation of the evolution of the accuracy for the SVC model. *Source: own source.*

Figure 5.71 shows the evolution of both test sizes for the SVC model. It is observable that the accuracy of both sets increases with the increase of measured data. This time, the growth of the 20% test size set has a better performance than the 33% test size. The highest reached accuracy of the

model has been 56,02%, the 20% test size, and the lowest one stands for a 34,83%, the 33% test size. *Table 5.4* shows the results of the evaluation for both test sizes and *Table 5.5* shows the classification report for, also, the both test size sets.

SVC	Test-size	
	20%	33%
Test Accuracy (%)	50,00	49,49
Train Accuracy (%)	54,18	53,24
Accuracy Gap (%)	4,18	3,75

Table 5. 4.- Result table for the SVC model evaluation.

SVC		Test-size									
		20%					33%				
Gesture		1	2	3	4	5	1	2	3	4	5
Classification Report	*Precision*	0,88	0,58	0,33	0,48	0,36	0,84	0,83	0,46	0,34	0,36
	Recall	0,80	0,16	0,17	0,78	0,71	0,84	0,08	0,18	0,74	0,75
	F1-score	0,84	0,25	0,22	0,60	0,48	0,84	0,14	0,26	0,54	0,49

Table 5. 5.- Classification report for the SVC model.

5.4.3. K-Nearest Neighbours (KNN)

This was the last model on be implemented (can be checked on Attachment C7). This model was first tested once the 100% of the data was measured, on one side because it was just developed later than the other two models and, on the other side because this model requires a bigger data set than the other two to be executed. Because of that, there is no record of the evolution of this model. In case more measures were taken in the future, this model could be tested as long the data set increases and track its evolution. Once the model has been evaluated, *See Table 5.6* and *Table 5.7*, it is observable that the highest accuracy is for the 33% test-size set with a 49,16% of accuracy. It is also worth mentioning that this model has a higher accuracy gap than the previous ones, which leads to an overlapping problem.

KNN	Test-size	
	20%	33%
Test Accuracy (%)	48,33	49,16
Train Accuracy (%)	66,30	65,39
Accuracy Gap (%)	17,97	16,23

Table 5. 6.- Result table for the KNN model evaluation.

KNN	Test-size

		20%					33%				
Gesture		**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
Classification Report	*Precision*	0,89	0,41	0,24	0,45	0,45	0,87	0,39	0,26	0,43	0,46
	Recall	0,91	0,49	0,25	0,47	0,29	0,93	0,42	0,18	0,49	0,48
	F1-score	0,90	0,45	0,24	0,46	0,36	0,90	0,40	0,21	0,46	0,47

Table 5. 7.- Classification report for the KNN model.

5.4.4. Comparison

After having tested and evaluated all three models, the decision of compare them between each other to analyse the obtained information and check which would perform better was taken. But before starting the comparison, a last implementation wanted to be taken to see if there was any improve on the models.

This implementation consisted on cutting all signals that where above the expected maximum and minimum value measurable, which is the range of the sensor and which corresponds to ±1,64 mV. This has been done because the data acquisition is a human process and there could have happened any unintentional human error, like pressing the measuring button by accident when there was no patient or starting the measuring process before placing the electrodes on the skin surface, a broken electrode, any many other conditions that could have affected to some measures. To solve this, two commands have been implemented at the end of the signal preprocessing code*, See Figure 5.72*, after having created the data frame with all features and gestures and just before converting the data frame into a .csv file.

```
df_all= df_all[df_all['MAX']<1.64].reset_index(drop = True)
df_all= df_all[df_all['MIN']>-1.64].reset_index(drop = True)
```

Figure 5. 72.- Command to remove unwanted signals from the data frame. *Source: own source.*

The function of this command is to check the features of the maximum and minimum values and those that exceed the maximum or the minim will be removed from the data frame leaving so, just the signals within the real range.

After applying this new implementation, the models have all been retested and evaluated as in the previous section. The *Table 5.8* and *Table 5.9* show the obtained results from the three models. The first table shows the evaluation results and the second one shows the classification report from all three models and for both test size sets.

	Logistic Regression	**SVC**	**KNN**

Test-size	20%	33%	20%	33%	20%	33%
Test Accuracy (%)	60,80	55,52	57,95	54,83	51,70	51,03
Train Accuracy (%)	56,13	56,12	62,11	61,56	64,39	62,76
Accuracy Gap (%)	4,67	0,60	4,16	6,73	12,69	11,73

Table 5. 8.- Result table for the three models evaluation with the implementation.

Classification Report		Test-size									
		20%					33%				
Gesture		1	2	3	4	5	1	2	3	4	5
Logistic Regression	*Precision*	0,90	0,44	0,54	0,59	0,52	0,89	0,37	0,49	0,47	0,49
	Recall	0,97	0,34	0,48	0,47	0,76	0,93	0,28	0,39	0,47	0,68
	F1-score	0,93	0,39	0,51	0,53	0,62	0,91	0,32	0,43	0,47	0,57
SVC	*Precision*	0,91	0,38	0,50	0,56	0,49	0,93	0,31	0,55	0,43	0,52
	Recall	0,89	0,23	0,42	0,57	0,76	0,86	0,21	0,31	0,60	0,75
	F1-score	0,90	0,29	0,46	0,57	0,60	0,89	0,25	0,39	0,50	0,61
KNN	*Precision*	0,94	0,40	0,26	0,59	0,44	0,96	0,37	0,29	0,43	0,52
	Recall	0,92	0,46	0,32	0,42	0,44	0,92	0,46	0,27	0,40	0,50
	F1-score	0,93	0,43	0,29	0,49	0,44	0,94	0,41	0,28	0,42	0,51

Table 5. 9.- Classification report for the three models with the implementation.

In first sight, these results already look much better than the previous ones, but to have a proper comparison, each model will be compared to extract the own conclusions per model. After the most efficient set from each model is chosen, those three choices will lastly be compared to finally get to the most efficient classification model.

The first model to be compared is the logistic regression model, *See Table 5.10* and *Table 5.11*. From the first table, the most obvious difference is in the test accuracy, after the implementation the accuracy has increased for both sets, but specially for the 20% test size set, where the accuracy increased almost a 10% respect the initial set. Despite the growth of the general model accuracy, the gap between accuracies has been increased, the difference is still less than a 5% which means that is not really worrying. It is worth mentioning that the highest test accuracy does also have the highest accuracy gap, or what is the same, the highest overlapping issues. So, in general, taking in consideration the evaluations, the final sets look to be more optimal than the initial ones, although a worst random evaluation and a higher accuracy gap.

Regarding the classification reports, second table, it is important to compare it gesture by gesture because it determines which gestures is classified the best and how many times this classification is done correctly. The precision is not an exceptionally reliable characteristic for what it is observed,

due to this factor has a high dependency on how many times a gesture has been detected. For example, if the model classifies 8 gestures as *gesture 5* and from these 8 gestures the prediction was correct 6 times, the classification report will show a really high value. But if then it is known that the total of *gesture 5* that had been sent to the model was of 50, then the recall will be low, as the model has only catch 6 correct *gesture 5* out of 50 that were in the set. For it, it is relevant to look in precision and recall, but the most reliable characteristic is the F1-score, as it is a combination of precision and recall. It is also particularly important to analyse the confusion matrices, because they show the number of gestures classified and misclassified (remember that the confusion matrices are in the Annex C8).

The conclusion of this, is that both final sets have a better gesture prediction and classification, in general, but the 20% test-size set is even more efficient than the 33% test-size set, due to it improves the performance of all gestures respect the 33% test-size set and it also improves in 4 out of 5 gestures respect the initial 20% test-size set.

Logistic Regression	**Initial**		**Final**	
Test-size	**20%**	**33%**	**20%**	**33%**
Test Accuracy (%)	50,56	53,54	60,80	55,52
Train Accuracy (%)	54,46	53,07	56,13	56,12
Accuracy Gap (%)	3,90	0,47	4,67	0,60

Table 5. 10.- Comparison of both logistic regression models evaluation.

Classification Report		**Test-size**									
		20%					**33%**				
Gesture		**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
Initial logistic regression	*Precision*	0,76	0,52	0,26	0,47	0,42	0,75	0,60	0,40	0,50	0,40
	Recall	0,91	0,37	0,17	0,50	0,62	0,93	0,44	0,21	0,55	0,61
	F1-score	0,83	0,43	0,20	0,48	0,50	0,83	0,50	0,28	0,52	0,48
Final logistic regression	*Precision*	0,90	0,44	0,54	0,59	0,52	0,89	0,37	0,49	0,47	0,49
	Recall	0,97	0,34	0,48	0,47	0,76	0,93	0,28	0,39	0,47	0,68
	F1-score	0,93	0,39	0,51	0,53	0,62	0,91	0,32	0,43	0,47	0,57

Table 5. 11.- Comparison of both logistic regression classification reports.

The second classification model to be compared has been the SVC, *See Table 5.12* and *Table 5.13.* and the same comparation guidelines will be followed as in the previous classification model. The first table including the evaluation characteristics shows, again, an improve in the test accuracies of the models after the implementation. The 20% test-size set is the one with a highest improve, almost 8% respect its initial set. A remarkable difference is that, this time, although the test accuracy from the final 20% test-size set has improved the most, its accuracy gap, although it is a little bit high since the beginning, it does not increase respect the initial set. For the final 33% set it does increase more.

For the classification reports, a remarkable general improve in the gestures is show, expect for gesture 4 in both final sets. Despite the gesture 4 has higher stats in the initial sets, the difference is not really high and having the rest of gestures higher in the final sets makes it no relevant. Between both final sets, the 20% test-size is, again, more efficient, but with not an extremely high difference.

Although both final sets have remarkably comparable results in the classification report, the 20% final test-size set has a higher accuracy, less accuracy gap. It is also worth mentioning that SVC has more overlapping issues than logistic regression, as its values are higher.

SVC	**Initial**		**Final**	
Test-size	**20%**	**33%**	**20%**	**33%**
Test Accuracy (%)	50,00	49,49	57,95	54,83
Train Accuracy (%)	54,18	53,24	62,11	61,56
Accuracy Gap (%)	4,18	3,75	4,16	6,73

Table 5. 12.- Comparison of both SVC models evaluation.

Classification Report		**Test-size**									
		20%					**33%**				
Gesture		**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
Initial SVC	*Precision*	0,88	0,58	0,33	0,48	0,36	0,84	0,83	0,46	0,34	0,36
	Recall	0,80	0,16	0,17	0,78	0,71	0,84	0,08	0,18	0,74	0,75
	F1-score	0,84	0,25	0,22	0,60	0,48	0,84	0,14	0,26	0,54	0,49
Final SVC	*Precision*	0,91	0,38	0,50	0,56	0,49	0,93	0,31	0,55	0,43	0,52
	Recall	0,89	0,23	0,42	0,57	0,76	0,86	0,21	0,31	0,60	0,75
	F1-score	0,90	0,29	0,46	0,57	0,60	0,89	0,25	0,39	0,50	0,61

Table 5. 13.- Comparison of both SVC classification reports.

The last classification model to be compared is the K-Nearest Neighbours*, See Tabled 5.14* and *Table 5.15.* With regards to the first table, the first obvious impression is that the general improvement of the characteristics of the model is much poorer than the previous models. Despite this, again the final test-size sets have a better accuracy. The accuracy gap improves but is high in all sets, more than 10% is already a value to take into consideration although it is not yet a big problem of overlapping. Respect of the evaluation, both final sets have related results, so the classification report will help decide which of both final sets is more efficient.

The classification report, second table, shows a significant improvement on the overall gestures' classifications, as it happens on the other models. Both final sets perform better than the initials and have a remarkably similar performance. Overall, both final sets performances are similar and, although the accuracy of the final 20% set is slightly higher than the 33% final set, this last one has less accuracy gap and better results in the classification result.

Overall, as for the final comparison all test-sizes need to be the same, as it would make no sense to compare different test-sizes and given the fact that both 20% and 33% test-sizes for the KNN model are similar, the chosen set has been the 20% final test-size model.

KNN	**Initial**		**Final**	
Test-size	**20%**	**33%**	**20%**	**33%**
Test Accuracy (%)	48,33	49,16	51,70	51,03
Train Accuracy (%)	66,30	65,39	64,39	62,76
Accuracy Gap (%)	17,97	16,23	12,69	11,73

Table 5. 14.- Comparison of both KNN models evaluation.

Classification Report		**Test-size**									
		20%					**33%**				
Gesture		**1**	**2**	**3**	**4**	**5**	**1**	**2**	**3**	**4**	**5**
Initial KNN	*Precision*	0,89	0,41	0,24	0,45	0,45	0,87	0,39	0,26	0,43	0,46
	Recall	0,91	0,49	0,25	0,47	0,29	0,93	0,42	0,18	0,49	0,48
	F1-score	0,90	0,45	0,24	0,46	0,36	0,90	0,40	0,21	0,46	0,47
Final KNN	*Precision*	0,94	0,40	0,26	0,59	0,44	0,96	0,37	0,29	0,43	0,52
	Recall	0,92	0,46	0,32	0,42	0,44	0,92	0,46	0,27	0,40	0,50
	F1-score	0,93	0,43	0,29	0,49	0,44	0,94	0,41	0,28	0,42	0,51

Table 5. 15.- Comparison of both KNN classification reports.

After having compared each classification model and having chosen the most efficient set for each model, it is time to compare them and check with classification model has been more reliable and with a better performance, *See Table 5.16* and *Table 5.17.* The first table compares the model evaluations and indicating the test size and if it is before or after the implementation. The second table compares the classification reports from each model indicating, again, the set and the type of set the values are from.

The first table indicates that the best overall accuracy is given by the Logistic Regression, although the SVC has a slightly better accuracy gap, 0,51% better. The SVC models has also the most accurate random evaluation result as it only a 0,11% far from the ideal 20%, much better than the rest of performances. The KNN has a really high accuracy gap, which indicates some overlapping issues and the lowest accuracy from all three models, this might be due to a KNN model requires a higher quantity of data to perform well. The classification report will help deciding which models is the best option: Logistic Regression or SVC.

The classification report's result show that the gestures classification performance is very distributed over the three models, and, in general ways, there is no substantial difference between models. The most remarkable differences between gestures appear on the *gesture 3* between KNN and SVC; and on the *gesture 2* between KNN and SVC. *Gesture 4 and 5* for the KNN model are also poorer on performance compared to the other two models.

Overall, takin in consideration both tables and realising that it has the best performance on the classification report, because for the gestures that this model is not the best performer the difference with the best is not as remarkable as its difference with the rest when this model has the best gesture performance and, because it has the best accuracy. The Logistic Regression classification model has been chosen as the best performance classification model for this project.

	Logistic Regression (Final)	**SVC (Final)**	**KNN (Final)**
Test-size	**20%**	**20%**	**20%**
Test Accuracy (%)	60,80	57,95	51,70
Train Accuracy (%)	56,13	62,11	64,39
Accuracy Gap (%)	4,67	4,16	12,69

Table 5. 16.- Comparation of the most efficient models evaluations.

Classification Report	**Test-size**				
	20%				
Gesture	**1**	**2**	**3**	**4**	**5**

Logistic Regression (Final)	*Precision*	0,90	0,44	0,54	0,59	0,52
	Recall	0,97	0,34	0,48	0,47	0,76
	F1-score	0,93	0,39	0,51	0,53	0,62
SVC (Final)	*Precision*	0,91	0,38	0,50	0,56	0,49
	Recall	0,89	0,23	0,42	0,57	0,76
	F1-score	0,90	0,29	0,46	0,57	0,60
KNN (Final)	*Precision*	0,94	0,40	0,26	0,59	0,44
	Recall	0,92	0,46	0,32	0,42	0,44
	F1-score	0,93	0,43	0,29	0,49	0,44

Table 5. 17.- Comparison of the most efficient models classification reports.

6. Environmental Impact Analysis

The effect of human activity has an impact on the environment and, this impact, tends to be every time more negative, although there are tendencies trying to change this aspect. For that reason, it is well considered to instil the idea that all actions have a consequence on the environment and with the right awareness, negative impact on the environment can be reduced.

The main tasks of this project have been about coding and researching, this is reflected on a higher expense of electricity.

Concerning the electronic components of the project: the components used for the assembly of the microcontroller has been minimised to maximum trying to use as less components as possible. The sEMG needs 3.3V of power supply which has been always supplied to the microcontroller from an USB cable direct from the laptop. In case a battery wants be used, it is highly recommendable to use rechargeable batteries. The microcontroller, ESP32, already includes a Wi-Fi module and an ADC, so there has been no need to buy extra modules and generate unnecessary waste.

The electrodes used for all measures were all recyclable and, after being used to its limit, properly recycled.

All the components that needed to be ordered, were ordered at the same time and to the same provider trying, so, to reduce the CO2 emissions. Another solution to reduce these emissions has been to directly going to a physical shop to buy the components.

Lastly, it is worth mentioning that this project has been designed and developed under the 2021/19/UE directive on waste of electric and electronic equipment [81].

7. Future Improvement

Some aspects to improve the typical performance of the project have been found during the elaboration process and some others at the end of it. The aim of these improvements is to apply them in future development processes to upgrade the created model. As this project has been distributed into two sections, the improvements regarding each section will be mentioned separately.

When it comes to the hardware of this project, some improvement aspects have been detected during the evolution of the design of the assembly. A first recommendation would be changing the sensor. Although this sensor is good for some aspects, it is not the best one for a real-time classification model. The main problem is that it is a single channel sensor, this allows only to record the activity of one muscle at the same time and, therefore, cannot give the best information about the muscle activity whenever a gesture is executed. A multi-channel sEMG would fit much better for a proper classification. Another aspect would be to design an own sensor, due to commercial sensors being already prepared to be used and they are not that easy to modify or use for own research. Good examples of sEMG to get inspired of which have multiple channels, are compact and perfect to incorporate on the glove would be the models designed by Delsys (Trigno Quattro Sensor*, See Figure XX (a),* and Trigno Galileo Sensor*, See Figure 7.1 (b)*) [82].

Figure 7. 1.- Trigno Quattro Sensor (a) and Trigno Galileo Sensor (b). *Source: delsys.com*

It is also important to take into consideration that all the commercial EMG sensors have their own data visualisation, monitoring and segmentation applications. This complicates, on one side an external data acquisition and, on the other side its visualisation. By creating an own sensor, an own data monitoring application could be designed making much easier the data visualisation and treatment process.

Another improvement would be the microcontroller, the ESP32 is great due to its quality-price correlation, but a microcontroller which has an ADC with higher resolution than the actual, to obtain a signal with higher resolution and values with a shorter refresh period would be optimal. Some

models that could work well might be: TMS320F2823x (Texas Instruments) or STM32H7 (ST) among others [83].

As it comes to the database, although Firebase is especially useful for storing data, it is not optimal for storing a dataset of time-signal data. The aim of Firebase is to store data for applications and webs, which is reflected in storing usernames and passwords. A more appropriate database for this project would have been, for example, Redis [84]. This database is well-known for executing data structure server functions by supporting strings, lists, sets, hashes, etc. and requires low latency and high-throughout.

On the other side, regarding the software section, the improvements aim to improve the accuracy of the model and obtain a better gesture classification. The first factor that can help to improve the models' accuracy is having more data. This means, on one side taking more measures to more patients and enlarge the database, and on the other side having more data from the same gesture. To accomplish this last factor, one solution would be having a multi-channel sEMG, as mentioned before, to get more data values from the same gesture but from different muscles, this leads to a better gesture definition. Another element that would improve the amount of data from the same gesture would be reading more data in the same amount of data as it is being read now, this can be accomplished by reducing the time lapse between measure readings and it can be done with a microcontroller with higher capacities. To improve the classification of the signal, the most useful improvement would be adding more and unique features to the model to, on one side reduce overlapping and, on the other side improve the accuracy of the model.

8. Conclusions

The goals of this project were to acquire, process and interpret data from hand-wrist gestures with a single-channel electromyographic sensor and create a Machine Learning classification algorithm. The last objective was to develop a real-time evaluation model to test how the model works when it comes to real-life. After developing it and having reached the results, some conclusions can be extracted from each section of the project.

Once the project is completed, it can be confirmed that the data acquisition is one of the most important parts. This makes the difference on having or not having a complete, efficient, and reliable dataset. Having a big amount of superior quality data is necessary for developing any Machine Learning model. This will affect the accuracy of the model and the way it will end up predicting gestures. For acquiring better quality data, the mentions from the future improvements should be taken into consideration (multi-channel sEMG, better ADC resolution, etc.).

The designed classification models have acquired a maximum accuracy of 60,80%. Although this value is not the expected in an efficient Machine Learning classification model (usually more than 86% accuracy), it is a result to be proud of considering the conditions and the components this project has been designed with. Another key factor that has affected the model's accuracy has been the chosen gestures. The electrodes of the single-channel sensor have been placed onto the radial carpal extensor muscle, but when a gesture is executed many muscles work and are activated on those gestures and some of the performed gestures have other muscles as its main performers and the used muscle is just a secondary character on its performance. This is a fact that has affected the results as those gesture signals were not that relevant nor that detachable from the others to extract some distinguishable data with respect to the other gestures. Gesture 2 and gesture 3, were two of those gestures where the used muscles were a secondary character. It is worth mentioning that having measured healthy as well as non-healthy patients might have also affected the accuracy of the model as both patient types offer different results, as it was expected. But it was a factor to take into consideration because the aim of the project is to help patients with some, or future developing, type of hand-wrist illnesses or diseases.

Regarding signal processing, the importance of segmenting, cleaning and extracting relevant information from the input raw data has been learned, as that makes the difference when it comes to training the model. Choosing good features that will offer relevant and differentiable information is one of the most important things of signal processing.

This project has helped to settle a starting point for introducing Machine Learning on the gesture recognition of the Smart Wearable Therapeutic Glove. It has also helped to identify the strengths and weaknesses of the developed parts of the project, to make future improvements.

9. Economic Analysis

This last section pretends to analyse and sum up all the costs that this project had, from the costs of engineering (the labour hours that have been needed to work on this project), the costs of the material used for the project (on one side the material needed to create the project and on the other side the indirect material needed to execute it), until the costs of the software programs used for this project. All these analyses have been separated by different tables to have a visual recapitulation of each section.

9.1. Engineering costs

This first sub-section refers to the economical compensation that an engineer working on this project would receive. The costs table has been divided by the hours spent on each of the tasks of this project, *See Table 9.1.*

Engineering costs			
Task	***Hour cost (€/h)***	***Hours (h)***	***Cost***
Previous formation	12,00	50	**600,00 €**
Assembly development	12,00	50	**600,00 €**
Data acquisition firmware	12,00	30	**360,00 €**
Data collection	12,00	50	**600,00 €**
Signal processing	12,00	140	**1.680,00 €**
Classification model	12,00	100	**1.200,00 €**
Evaluation of the model	12,00	80	**960,00 €**
Report Documentation	12,00	150	**1.800,00 €**
Total Cost			***7.800,00 €***

Table 9. 1.- *Labour cost deployment.*

9.2. Material costs

This second sub-section includes all the materials needed for the project. On one side, *See Table 9.2,* all the elements needed to make the assembly for measuring data and sending it to the database and, on the other side, *See Table 9.3,* the material needed to execute the project and to help building the assembly.

Material costs					
Material	***Provider***	***Unit cost (€/unit)***	***Shipping (€)***	***Quantity***	***Cost***
4,7 kΩ Resistors	Farnell	0,62 €	4,99 €	2	**15,78 €**
16 Male pins		9,55 €		1	
Breadboard	AZ Delivery	5,99 €	0,00 €	1	**5,99 €**
Micro USB cable	MEMORYKING	4,45 €	1,99 €	1	**6,44 €**
10 Prototyping boards	Garosa	1,25 €	0.99 €	1	**13,49 €**
ESP32	AZ Delivery	11,29 €	2,99 €	1	**14,28 €**
EMG Sensor (UC-E6 Connector)	PLUX	30,32 €	11,51 €	1	**41,83 €**
Set of 20 electrodes	Axion	14,99 €	0,00 €	1	**14,99 €**
Total Cost					***112,80 €***

Table 9. 2.- *Material needed to create the assembly costs.*

Inventoriable costs		
Material	***Provider***	***Cost***
Soldering station	Preciva	**39,99 €**
Laptop	Lenovo	**599,00 €**
Total Cost		***638,99 €***

Table 9. 3.- *Inventoriable costs.*

9.3. **License costs**

This third sub-section encompasses all kinds of software used during the project*, See Table 9.4,* an effort has been made trying to use as much free software as possible to reduce costs.

License costs	
Software	***Cost***
Arduino IDE	**0,00 €**
Firebase	**0,00 €**
VS Code	**0,00 €**
MS Office	**70,00 €**
MS Windows	**145,00 €**
TOTAL COST	***215,00 €***

Table 9. 4.- *Costs of licenses from used software during the project.*

9.4. **Final economic balance**

This final part of the economic balance is a compilation of the total of each sub-section mentioned before to visualize to total cost of the entire project plus the addition of the Value Added Tax for an engineering project, which corresponds to a 21% of the total*, See Table 9.5.*

Final Economic Balance	
Type of cost	***Cost***
Engineering cost	**7.800,00 €**
Material cost	**112,80 €**
Inventoriable cost	**638,99 €**
License cost	**215,00 €**
Cost	***8.766,79 €***
VAT Import (21%)	***1.841,03 €***
Total Cost	***10.607,82 €***

Table 9. 5.- *Total costs of the project.*

References

[1] Elements of AI. *Elements of AI* [online]. Last updated in 2022. [Checked on Jan 15, 2022]. Available on: <https://course.elementsofai.com/>.

[2] Building AI. *Elements of AI* [online]. Last updated in 2022. [Checked on Jan 29, 2022]. Available on: <https://buildingai.elementsofai.com/>.

[3] Caiming Zhang, Yang Lu. Study on artificial intelligence: The state of the art and prospects. *Journal of Industrial Information Integration* [online]. Volume 23; Sep, 2021. [Checked on Feb 15, 2022]. Available on: <https://doi.org/10.1016/j.jii.2021.100224>

[4] César Bravo, Luigi Saputelli, Francklin Rivas, Anna Gabriela Pérez, Michael Nikolaou, Georg Zangl, Neil de Guzman, Shahab Mohaghegh, and Gustavo Nunez. State-of-the-Art Application of Artificial Intelligence and Trends in the E&P Industry: A Technology Survey. *Research Gate* [online]. Mar, 2012. [Checked on Feb 15, 2022]. Available on: <https://www.researchgate.net/publication/254535146_State-of-the-Art_Application_of_Artificial_Intelligence_and_Trends_in_the_EP_Industry_A_Technology_ Survey>.

[5] Fuzzy Logic | Introduction. *Geeks for Geeks* [online]. Last update Oct 06, 2021. [Checked on Feb 15, 2022]. Available on: <https://www.geeksforgeeks.org/fuzzy-logic-introduction/>

[6] Neural Networks. *IBM* [online]. Aug 17, 2020. [Checked on Feb 15, 2022]. Available on: <https://www.ibm.com/cloud/learn/neural-networks>

[7] Evolutionary Computation - A Technology Inspired by Nature. *IEEE.tv* [online]. Oct 19, 2013. [Checked on Feb 15, 2022]. Available on: <https://ieeetv.ieee.org/technology/evolutionary_computation_a_technology_inspired_by_nature>

[8] 12 Most Useful Data Mining Applications of 2022. *UpGrad* [online]. Jan 08, 2021. [Checked on Feb 15, 2022]. Available on: < https://www.upgrad.com/blog/12-most-useful-data-mining-applications/>

[9] Herbert A. Edelstein. Introduction to Data Mining & Knowledge Discovery [Two Crows Corp]. Mar 01, 1998. [Checked on Feb 15, 2022].

[10] Ray Chaudhuri, Upta & Ray Chaudhuri, Utpal. Fundamentals of Automatic Process Control [CRC Press]. 2012. [Checked on Feb 15, 2022].

[11] Semantic Segmentation. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: <https://paperswithcode.com/task/semantic-segmentation>.

[12] Image Classification. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: < https://paperswithcode.com/task/image-classification>.

[13] Object Detection. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: < https://paperswithcode.com/task/object-detection>.

[14] Language Modelling. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: <https://paperswithcode.com/task/language-modelling>.

[15] Machine Translation. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: <https://paperswithcode.com/task/machine-translation>.

[16] Question Answering. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: <https://paperswithcode.com/task/question-answering>.

[17] Drug Discovery. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: <https://paperswithcode.com/task/drug-discovery>.

[18] Medical Diagnosis. *Papers with code* [online]. Last updated Jan, 2022. [Checked on Apr 23, 2022]. Available on: <https://paperswithcode.com/task/medical-diagnosis>.

[19] Raez MB, Hussain MS, Mohd-Yasin F. Techniques of EMG signal analysis: detection, processing, classification and applications. *Biol Proced* [Online]. Mar 23, 2006. [Checked on Feb 15, 2022]. Available on: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1455479/#:~:text=There%20are%20many%20applications%20for,of%20biofeedback%20or%20ergonomic%20assessment>

[20] Enea Ceolini, Charlotte Frenkel, Sumit Bam Shrestha, Gemma Taverni, Lyes Khacef, Melika Payvand and Elisa Donati. Hand-Gesture Recognition Based on EMG and Event-Based Camera Sensor Fusion: A Benchmark in Neuromorphic Computing. *Frontiers in Neuroscience* [online]. Aug 05, 2021. [Checked on Feb 15, 2022]. Available on: <https://www.frontiersin.org/articles/10.3389/fnins. 2020.000637/full>

[21] Cen, J.-Y.; Dutta, T. Development and Evaluation of a Slip Detection Algorithm for Walking on Level and Inclined Ice Surfaces. *Sensors* [online]. Mar 18, 2022. [Checked on Apr 23, 2022]. Available on: <https:// doi.org/10.3390/s22062370>.

[22] What Is EMG (Electromyography) and How Does It Work? *Imotions* [online]. Jul 24, 2018. [Checked on Jan 27, 2022]. Available on: <https://imotions.com/blog/electromyography-101/>.

[23] Electromyograms. *Cleveland Clinic* [online]. Last reviewed on Jun 12, 2016. [Checked on Jan 27, 2022]. Available on: <https://my.clevelandclinic.org/health/articles/4825-electromyograms>.

[24] Electromyography (EMG). *Mayo Clinic* [online]. May 21, 2019. [Checked on Jan 27, 2022]. Available on: <https://www.mayoclinic.org/tests-procedures/emg/about/pac-20393913>.

[25] Chowdhury RH, Reaz MB, Ali MA, Bakar AA, Chellappan K, Chang TG. Surface electromyography signal processing and classification techniques. *Ncbi* [online]. Sep 17, 2013. [Checked on Feb 19, 2022]. Available on: <https://www.ncbi.nlm.nih.gov/pmc/articles/PMC 3821366/>.

[26] The Wavelet Transform. *Towards Data Science* [online]. Dec 21, 2020. [Checked on Feb 27, 2022]. Available on: <https://towardsdatascience.com/the-wavelet-transform-e9cfa85d7b34>.

[27] S McLaughlin, A Stogioglou and J Fackrell. Introducing Higher Order Statistics (HOS) for the Detection of Nonlinearities. *UK Nonlinear News* [online]. Sep, 1995. [Checked on Feb 19, 2022] Available on: <http://www1.maths.leeds.ac.uk/applied/news.dir/issue2/hos_intro.html>.

[28] Max Lambert, Andrew Engroff, Matt Dyer, Ben Byer. Empirical Mode Decomposition. *Clear.rice.*edu [online]. Last modified on Jan 18, 2003. [Checked on Feb 19, 2022]. Available on: <https://www.clear.rice.edu/elec301/Projects02/empiricalMode.html>.

[29] What is Independent Component Analysis? *University of Helsinki, department of Computer Science* [online]. Apr, 1999. [Checked on Feb 19, 2022]. Available on: <https://www.cs.helsinki.fi/ u/ahyvarin/whatisica.shtml>.

[30] Electromyography (EMG) Sensor Data Sheet. *Bitalino* [online]. Last updated in 2016. [Checked on Jan 29, 2022]. Available on: <https://bitalino.com/storage/uploads/media/revolution-emg-sensor-datasheet-1.pdf>.

[31] ESP32-WROOM-32, Datasheet. *Espress if* [online]. Last reviewed in 2022. [Checked on Feb 23, 2022]. Available on: <https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>

[32] Arduino Libraries. *Arduino* [online]. Last reviewed in 2022. [Checked on Feb 23, 2022]. Available on: < https://www.arduino.cc/en/main/libraries>

[33] EVERYTHING YOU NEED TO KNOW ABOUT ARDUINO CODE. *Circuit.io* [online]. March 11, 2018. [Checked on Feb 27, 2022]. Available on: <https://www.circuito.io/blog/ arduino-code/>.

[34] Arduino Documentation. *Arduino.cc* [online]. Last updated in 2022. [Checked on Feb 27, 2022]. Available on: <https://docs.arduino.cc/>.

[35] Wi-Fi. *Arduino* [online]. Last updated in 2022. [Checked on Mar 07, 2022]. Available on: <https://www.arduino.cc/reference/en/libraries/wifi/>.

[36] Empezando con Firebase (Realtime Database & Authentication). *medium* [online]. Jun 17, 2018. [Checked on Mar 07, 2022]. Available on: <https://medium.com/@margalida.kaska nte/empezando-con-firebase-realtime-database-authentication-a5c54b3b67d6>.

[37] Firebase.h. *GitHub* [online]. Last update Feb 04, 2022. [Checked on Mar 07, 2022]. Available on: <github.com/mobizt/Firebase-ESP32/blob/master/src/FirebaseESP32.h>.

[38] MAXSENS. *Maxsens* [online]. Last updated in 2021. [Checked on Mar 07, 2022]. Available on: <https://maxsens.es/>.

[39] Mona Lisa Delva, Kim Lajoie, Mahta Khoshnam and Carlo Menon. Wrist worn wearables based on force myography: on the significance of user anthropometry. *ResearchGate* [online]. Jun 12, 2020. [Checked on Mar 07, 2022]. Available on: <https://biomedicalengineeringonline.biomed central.com /articles/10.1186/s12938-020-00789-w>.

[40] Músculo flexor radial del carpo. *Wikipedia* [online]. Last update in Jan 08, 2021. [Checked on Mar 14, 2022]. Available on: <https://es.wikipedia.org/wiki/M%C3%BAsculo_flexor_radial_del_ carpo>.

[41] Artificial Intelligence (AI). *Investopedia* [online]. Last updated on Mar 08, 2021. [Checked on Apr 18, 2022]. Available on: <https://www.investopedia.com/terms/a/artificial-intelligence-ai. asp>.

[42] Artificial intelligence. *Britannica* [online]. Last updated on Mar 18, 2022. [Checked on Apr 18, 2022]. Available on: <https://www.britannica.com/technology/artificial-intelligence>.

[43] Alan Turing, el arma secreta de los aliados. *Historia, National Geographic* [online]. Last updated on Feb 17, 2021. [Checked on Apr 18, 2022]. Available on: <https://historia. nationalgeographic.com.es/a/alan-turing-arma-secreta-aliados_16352>.

[44] Machine Learning. *Sas* [online]. Last updated in 2022. [Checked on Apr 18, 2022]. Available on: <https://www.sas.com/en_us/insights/analytics/machine-learning.html>.

[45] Comparison of Artificial and Spiking Neural Networks on Digital Hardware. *Frontiersin* [online]. Apr 06, 2021. [Checked on Apr 18, 2022]. Available on: <https://www.frontiersin.org/articles/10. 3389/fnins.2021.651141/full>.

[46] What is Deep Learning? *Machine Learning Mastery* [online]. Aug 16, 2019. [Checked on Apr 18, 2022]. Available on: <https://machinelearningmastery.com/what-is-deep-learning/>.

[47] Goals of AI. *Wordpress* [online]. [Checked on Apr 18, 2022]. Available on: <https://poonam395 8.wordpress.com/goals-of-ai/>.

[48] Top 12 Artificial Intelligence Tools & Frameworks you need to know. *Edureka* [online]. Dec 16, 2021. [Checked on Apr 18, 2022]. Available on: <https://www.edureka.co/blog/top-12-artificial-intelligence-tools/>.

[49] Scikit-learn. *Scikit-learn* [online]. Last updated in 2022. [Checked on Apr 18, 2022]. Available on: <https://scikit-learn.org/stable/index.html>.

[50] TensorFlow. *TensorFlow* [online]. Last updated in 2022. [Checked on Apr 18, 2022]. Available on: <https://www.tensorflow.org/>.

[51] AI Applications: Top 14 Artificial Intelligence Applications in 2022. *SimpliLearn* [online]. Last updated in Apr 21, 2022. [Checked on Apr 18, 2022]. Available on: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/artificial-intelligence-applications>.

[52] AI - Environment types. *Csveda* [online]. Last updated in 2021. [Checked on Apr 18, 2022]. Available on:<https://csveda.com/ai-environment-types/>.

[53] Code Editor Definition, Sublime Text Tutorial. *Soject* [online]. Jul 15, 2021. [Checked on Apr 18, 2022]. Available on: <https://soject.com/code-editor-definition/>.

[54] Programming Language. *JavaTpoint* [online]. Last updated in 2021. [Checked on Apr 18, 2022]. Available on: <https://www.javatpoint.com/programming-language>.

[55] What is Python? Executive Summary. *Python* [online]. Last updated in 2022. [Checked on Apr 18, 2022]. Available on: <https://www.python.org/doc/essays/blurb/>.

[56] What is Machine Learning. *Potentia Analytics* [online]. Last updated in 2021. [Checked on Apr 18, 2022]. Available on: <https://www.potentiaco.com/what-is-machine-learning-definition-types-applications-and-examples/>.

[57] Supervised Learning. *TechTarget* [online]. Last updated in Mar 2021. [Checked on Apr 18, 2022]. Available on: <https://www.techtarget.com/searchenterpriseai/definition/supervised-learning>.

[58] Unsupervised Machine Learning. *Guru99* [online]. Last updated on Mar 08, 2022. [Checked on Apr 18, 2022]. Available on: <https://www.guru99.com/unsupervised-machine-learning. html>.

[59] Reinforcement Learning. *Geeks for Geeks* [online]. Last updated on Nov 18, 2021. [Checked on Apr 18, 2022]. Available on: <https://www.geeksforgeeks.org/what-is-reinforcement-learning/>.

[60] 10 Machine Learning Methods that Every Data Scientist Should Know. *Towards Data* Science [online]. May 1, 2019. [Checked on Apr 18, 2022]. Available on: <https://towardsdatascience.com/10-machine-learning-methods-that-every-data-scientist-should-know-3cc96e0eeee9>.

[61] Machine Learning Models. *Javatpoint* [online]. Last updated in 2021. [Checked on Apr 18, 2022]. Available on: < https://www.javatpoint.com/machine-learning-models>.

[62] The Limitations of Machine Learning. *Towards Data* Science [online]. Jul 29, 2019. [Checked on Apr 18, 2022]. Available on: <https://towardsdatascience.com/the-limitations-of-machine-learning-a00e0c3040c6>.

[63] Pyrebase. *GitHub* [online]. Feb 18, 2020. [Checked on Mar 04, 2022]. Available on: <https://github.com/thisbejim/Pyrebase>.

[64] SciPy. *SciPy* [online]. Last updated in May 18, 2022. [Checked on Mar 04, 2022]. Available on: < https://scipy.org/>.

[65] Python: Analysing EMG signals. *Scientifically* Sound [online]. Aug 18, 2016. [Checked on Mar 04, 2022]. Available on: <https://scientificallysound.org/2016/08/18/python-analysing-emg-signals-part-3/>.

[66] What is a Notch Filter? *Everything* RF [online]. Jul 25, 2021. [Checked on Mar 04, 2022]. Available on: <https://www.everythingrf.com/community/what-is-a-notch-filter>.

[67] NumPy. *NumPy* [online]. Last updated in 2022. [Checked on Mar 04, 2022]. Available on: <https://numpy.org/>.

[68] Variance Calculator. *Calculator Soup* [online]. Last reviewed on 2022. [Checked on Mar 25, 2022]. Available on: < https://www.calculatorsoup.com/calculators/statistics/variance-calculator.php>.

[69] Reza Bagherian Azhiri, Mohammad Esmaeili, Mehrdad Nourani. EMG-Based Feature Extraction and Classification for Prosthetic Hand Control. *Arxiv* [online]. Jul 1, 2021. [Checked on Mar 05, 2022]. Available on: < https://arxiv.org/pdf/2107.00733.pdf >.

[70] Pandas. *Pandas* [online]. Last updated in Apr 02, 2022. [Checked on Mar 15, 2022]. Available on: < https://pandas.pydata.org/ >.

[71] What is a Correlation Matrix? *DisplayR* [online]. Feb 20, 2019. [Checked on Apr 10, 2022]. Available on: < https://www.displayr.com/what-is-a-correlation-matrix/>.

[72] Seaborn. *Seaborn* [online]. Last updated in 2021. [Checked on Apr 10, 2022]. Available on: < https://seaborn.pydata.org/>.

[73] Matplotlib. *Matplotlib* [online]. Last updated Dec 11, 2021. [Checked on Mar 14, 2022]. Available on: < https://matplotlib.org/>.

[74] Choosing the right estimator. *Scikit*-learn [online]. Last reviewed in May, 2022. [Checked on Apr 07, 2022]. Available on: <https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html>.

[75] Scikit-learn. Scikit-learn [online]. Last updated in May, 2022. [Checked on Apr 14, 2022]. Available on: <https://scikit-learn.org/stable/index.html>.

[76] Train-Test Split for Evaluating Machine Learning Algorithms. *Machine Learning* Mastery [online]. Last updated on Aug 26, 2020. [Checked on Apr 14, 2022]. Available on: <https://machinelearningmastery.com/train-test-split-for-evaluating-machine-learning-algorithms/>.

[77] A Simple Example of Pipeline in Machine Learning with Scikit-learn. *Towards science* [online], Nov 12, 2018. [Checked on May 10, 2022]. Available on: <https://towardsdatascience.com/a-simple-example-of-pipeline-in-machine-learning-with-scikit-learn-e726ffbb6976>.

[78] Jiajia Niu, Guoshuai An, Zhen Gu, Peng Li, Qiqing Liu, Rufeng Bai, Junhong Sun & Qiuxiang Du. Analysis of sensitivity and specificity: precise recognition of neutrophils during regeneration of contused skeletal muscle in rats. *Taylor & Francis* [online]. Mar 19, 2020. [Checked on May 10, 2022]. Available on: < https://www.tandfonline.com/doi/full/10.1080/20961790.2020.1713432 >.

[79] Classification Report in Machine Learning. *The clever* programmer [online]. Jul 07, 2021. [Checked on May 10, 2022]. Available on: <https://thecleverprogrammer.com/2021/07/07/classification-report-in-machine-learning/>.

[80] Pickle – Python object serialization. Python [online]. Last updated in 2022. [Checked on May 10, 2022]. Available on: <https://docs.python.org/3/library/pickle.html>.

[81] Directive 2012/19/EU of the European Parliament and of the Council of 4 July 2012 on Waste Electrical and Electronic Equipment (WEEE). *European Union*. 2012. [Checked on May 15, 2022]. Available on: <Off J Eur Union. 2012;55(1):38–71>.

[82] Delsys. Delsys [online]. Last updated in 2022. [Checked on May 23, 2022]. Available on: <https://delsys.com/>.

[83] List of Microcontrollers with Fast Analog-to-Digital Converters. *Niconiconi's blog* [online]. June 07, 2022. [Checked on May 23, 2022]. Available on: <https://niconiconi.neocities.org/posts/list-of-mcu-with-fast-adc/>.

[84] Redis. *Redis* [online]. Last updated in 2022. [Checked on May 23, 2022]. Available on: <https://redis.io/>.

Attachment A

This annex pretends to show the Gantt diagram, which shows the organisation and execution of each part of the project, as well as the table with all the information regarding the measured patients.

A1. Gantt Diagram

Gantt_Diagram.xlsx

Task	Initial date	Ending date	Status
EMG (Hardware)	**24.01.2022**	**16.05.2022**	Finished
Background Research	24.01.2022	07.02.2022	Finished
Learn how an EMG works	07.02.2022	21.02.2022	Finished
Assembly EMG	21.02.2022	07.03.2022	Finished
Test EMG	07.03.2022	14.03.2022	Finished
Fix opeational error	14.03.2022	28.03.2022	Finished
Retest EMG	28.03.2022	28.03.2022	Finished
Reassembly for measures(Brainstorming)	11.04.2022	25.04.2022	Finished
Reassembly EMG	25.04.2022	25.04.2022	Finished
Test Reassembly EMG	25.04.2022	25.04.2022	Finished
Take measures	02.05.2022	16.05.2022	Finished
EMG (Software)	**07.02.2022**	**16.05.2022**	Finished
Research interaction ESP32 & Arduino	07.02.2022	14.02.2022	Finished
Research interaction ESP32 & Python	14.02.2022	28.02.2022	Finished
Arduino code for EMG signals	21.02.2022	28.02.2022	Finished
Research about Firebase	28.02.2022	07.03.2022	Finished
Arduino code to connect Firebase with Arduino	07.03.2022	21.03.2022	Finished
Python script to connect Firebase with Python	28.03.2022	11.04.2022	Finished
Arduino code to measure EMG	18.04.2022	02.05.2022	Finished
Arduino evaluation code	02.05.2022	16.05.2022	Finished
AI	**31.01.2022**	**30.05.2022**	Finished
Elements of AI (course)	31.01.2022	07.02.2022	Finished
Building AI (course)	07.02.2022	28.02.2022	Finished
Libraries research	07.03.2022	21.03.2022	Finished
Classification models research	21.03.2022	11.04.2022	Finished
Machine Learning libraries	11.04.2022	25.04.2022	Finished
Features research	25.04.2022	02.05.2022	Finished
First processing code	02.05.2022	09.05.2022	Finished
Second processing code	09.05.2022	16.05.2022	Finished
Logistic Regression Model	02.05.2022	09.05.2022	Finished
SVC Model	09.05.2022	09.05.2022	Finished
KNN Model	16.05.2022	16.05.2022	Finished
Implementation in processing code	16.05.2022	23.05.2022	Finished
Evaluation code	16.05.2022	30.05.2022	Finished
Report upload	**24.01.2022**	**13.06.2022**	Finished
Presentation	**20.06.2022**	**04.07.2022**	In Progress

A2. Patients information

Patients.xlsx

	Gender	Age	LAM	RAM	LAI	RAI	LWI	RWI	THP	TGR
0	-	-	-	-	-	-	-	-	-	-
1	Male	22	Yes	Yes	No	No	No	No	Yes	6
2	Female	58	Yes	Yes	Yes	No	No	No	No	6
3	Male	23	Yes	Yes	No	No	No	No	Yes	6
4	Male	23	Yes	Yes	No	No	Yes	No	No	6
5	Male	23	Yes	Yes	No	Yes	No	Yes	No	6
6	Male	25	Yes	Yes	No	No	No	No	Yes	6
7	Female	18	Yes	Yes	No	No	No	No	Yes	6
8	Male	58	Yes	Yes	No	No	No	Yes	No	6
9	Male	22	Yes	Yes	No	No	No	No	Yes	6
10	Female	23	Yes	Yes	No	No	No	No	Yes	6
11	Female	22	Yes	Yes	No	No	Yes	No	No	6
12	Male	22	Yes	Yes	No	No	No	No	Yes	6
13	Male	54	Yes	Yes	Yes	No	No	No	No	6
14	Male	53	Yes	Yes	No	No	No	No	Yes	6
15	Male	46	Yes	Yes	No	Yes	No	Yes	No	6
16	Female	48	Yes	Yes	No	No	No	No	Yes	6
17	Female	22	Yes	Yes	No	Yes	No	No	No	6
18	Female	22	Yes	Yes	No	No	No	No	Yes	6
19	Male	24	Yes	Yes	No	No	No	No	Yes	6
20	Male	24	Yes	Yes	No	No	No	No	Yes	6
21	Male	24	Yes	Yes	No	No	Yes	No	No	6
22	Male	20	Yes	Yes	No	No	No	No	Yes	6
23	Male	20	Yes	Yes	No	No	No	No	Yes	6
24	Female	28	Yes	Yes	No	No	No	Yes	No	6
25	Male	28	Yes	Yes	No	No	No	No	Yes	6
26	Female	23	Yes	Yes	Yes	No	Yes	No	No	6
27	Female	23	Yes	Yes	No	No	No	No	Yes	6
28	Male	22	Yes	Yes	Yes	No	No	No	No	6
29	Female	21	Yes	Yes	Yes	No	Yes	No	No	6
30	Male	21	Yes	Yes	Yes	Yes	No	No	Yes	6

LAM	Left Arm Measured
RAM	Right Arm Measured
LAI	Left Arm Injuries
RAI	Right Arm Injuries
LWI	Left Wrist Injury
RWI	Right Wrist Injury
THP	Totally Healthy Patient
TGR	Total Gest Recorded

Attachment B

This annex contains all the created firmware codes with the Arduino IDLE for Measuring, sending and storin data from the sEMG.

B1. Read data (read_data_v1.ino)

```
1. int sensorPin = 34;
2.
3. void setup() {
4.  Serial.begin(115200);
5. }
6.
7. void loop(){
8.  float sensorValue = analogRead(sensorPin);
9.  delay(50);
10.       float voltage = ((((sensorValue/4095)-
   0.5)*3.3)/1009)*1000;
11.       Serial.println(voltage);
12.     }
```

B2. Connect to Firebase (measuring_checkout.ino)

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include "Secret_keys.h"

int sensorPin = 34;
const char* ssid = WIFI_SSID;
const char* password = WIFI_PASSWORD;
const char* host = FIREBASE_HOST;
const char* auth = FIREBASE_AUTH;

     FirebaseData;

     void setup() {
      Serial.begin(115200);
      Serial.println();
      WiFi.begin(ssid, password);
      Serial.print("Connecting to WiFi");
      while (WiFi.status() != WL_CONNECTED){
        Serial.print(".");
        delay(300);
        }
      Serial.println("WiFi succesfully connected!");

      Firebase.begin(host, auth);
      Firebase.reconnectWiFi(true);
      Firebase.setInt(firebaseData,"/EMGdata/Checkout","Okey
");

      int i=0;
      while (i<150){
        float sensorValue = analogRead(sensorPin);
        delay(30);
        float voltage = ((((sensorValue/4095)-
0.5)*3.3)/1009)*1000;
        Serial.println(voltage);
        i++;
       }
     }


     void loop() {}
```

B3. Final code (record_data.ino)

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include "Secret_keys.h"

int sensorPin = 34;

const char* ssid = WIFI_SSID;
const char* password = WIFI_PASSWORD;
const char* host = FIREBASE_HOST;
      const char* auth = FIREBASE_AUTH;

      FirebaseData;

      void setup() {
       Serial.begin(115200);
       Serial.println();

       WiFi.begin(ssid, password);
       Serial.print("Connecting to WiFi");
       while (WiFi.status() != WL_CONNECTED){
         Serial.print(".");
         delay(300);
       }

       Serial.println("WiFi succesfully connected!");

       Firebase.begin(host, auth);
       Firebase.reconnectWiFi(true);

       Firebase.getInt(firebaseData,"/EMGdata/Measure_patient
s");
       int patients = firebaseData.intData();
       patients++;
       Firebase.setInt(firebaseData,"/EMGdata/Measure_patient
s",patients);

       int j=1;
       int i=0;
       while (j<6){
         if (j==1){
           Serial.println("Hello! We are starting the test
soon.");
           delay(2000);
           Serial.println("Prepare the first gesture,
please.");
           delay(2000);
           Serial.println("We are starting...");
           delay(2000);
         }
         else{
```

```
47.           Serial.println("Measuring time is over.");
48.           delay(2000);
49.           Serial.println("Prepare the following gesture,
   please.");
50.           delay(2000);
51.           Serial.println("It's staring...");
52.           delay(2000);
53.         }
54.           while (i<100){
55.             float sensorValue = analogRead(sensorPin);
56.             delay(30);
57.             float voltage = ((((sensorValue/4095)-
   0.5)*3.3)/1009)*1000;
58.             Serial.println(voltage);
59.
60.             if (Firebase.pushFloat(firebaseData,
   "/EMGdata/Gesture"+String(j), voltage)){
61.             }
62.             else{
63.               Serial.println("Failed");
64.               Serial.println("REASON: " +
   firebaseData.errorReason());
65.             }
66.             i++;
67.           }
68.         i=0;
69.         j++;
70.         }
71.
72.         Serial.println("Thank you very much for your help!");
73.         delay(2000);
74.         Serial.println("Bye-Bye!");
75.         delay(2000);
76.       }
77.
78.
79.       void loop(){}
```

B4. Key saver code (Secret_keys.h)

```
1. //WIFI Credentials
2. //#define WIFI_SSID "sercommBB1422"
3. //#define WIFI_PASSWORD "9CGFCDKDUE4DKSGY"
4. #define WIFI_SSID "xxxxxxxxxxx"
5. #define WIFI_PASSWORD "xxxxxxxxx"
6.
7. //Firebase Project Credentials
8. #define FIREBASE_HOST "https://xxxxxxxx-default-rtdb.europe-
   west1.firebasedatabase.app"
9. #define FIREBASE_AUTH "xxxxxxxxxxxxxxx"
```

B5. Evaluation code (evaluation.ino)

```
#include <WiFi.h>
#include <FirebaseESP32.h>
#include "Secret_keys.h"

int sensorPin = 34;

const char* ssid = WIFI_SSID;
const char* password = WIFI_PASSWORD;
const char* host = FIREBASE_HOST;
const char* auth = FIREBASE_AUTH;

FirebaseData;

void setup() {
 Serial.begin(115200);
 Serial.println();

 WiFi.begin(ssid, password);
 Serial.print("Connecting to WiFi");
 while (WiFi.status() != WL_CONNECTED){
   Serial.print(".");
   delay(300);
 }

 Serial.println("WiFi succesfully connected!");

 Firebase.begin(host, auth);
 Firebase.reconnectWiFi(true);


 int i=0;

 while (i<100){
   float sensorValue = analogRead(sensorPin);
   delay(30);
   float voltage = ((((sensorValue/4095)-0.5)*3.3)/1009)*1000;
   Serial.println(voltage);

   if (Firebase.pushFloat(firebaseData, "/EMGdata/evaluation", voltage)){
   }
   else{
     Serial.println("Failed");
     Serial.println("REASON: " + firebaseData.errorReason());
   }
   i++;
 }
}
```

Attachment C

This last attachment contains all the created Python codes from the ML classification model, its evaluation and for the signal processing. It also contains screenshots of the signal during the pre-processing and screenshots of the obtained results from each model.

C1. Preprocessing functions (functions.py)

This attachment includes all the functions designed for the signal preprocessing models and for the evaluation models.

```
import numpy as np
from math import sqrt
import matplotlib.pyplot as plt
import scipy as sp
from scipy import signal


#THIS FUNCTION CORRECT THE LENGTH FROM THE SIGNALS
    def correct_length(gest):
     new_gest = []
     for i in range(0,len(gest),99):
       new_gest += gest[i:i+99] + [0]
     return new_gest


    #THIS FUNCTION REMOVES THE OFFSET FROM  THE SIGNAL
    def rmv_mean(gest, time):
     for i in range(0,len(gest),100):
       emg_correctmean = gest - np.mean(gest)
     return emg_correctmean


    #THIS FUNCTION CREATES A BANDPASS FILTER
    def bp_filter(x, low_f, high_f, samplerate,plot=False):
       x = x - np.mean(x)

       low_cutoff_bp = low_f / (samplerate / 2)
       high_cutoff_bp = high_f / (samplerate / 2)

       [b, a] = signal.butter(5, [low_cutoff_bp,
high_cutoff_bp], btype='bandpass')
```

```
34.         x_filt = signal.filtfilt(b, a, x)
35.
36.         if plot:
37.             t = np.arange(0, len(x) / samplerate, 1 /
    samplerate)
38.             plt.plot(t, x)
39.             plt.plot(t, x_filt, 'k')
40.             plt.autoscale(tight=True)
41.             plt.xlabel('Time')
42.             plt.ylabel('Amplitude (mV)')
43.             plt.show()
44.
45.         return x_filt
46.
47.
48.
49.     #THIS FUNCTION CREATES A NOTCH FILTER
50.     def notch_filter(x, samplerate, plot=False):
51.         x = x - np.mean(x)
52.
53.         high_cutoff_notch = 59 / (samplerate / 2)
54.         low_cutoff_notch = 61 / (samplerate / 2)
55.
56.         [b, a] = signal.butter(4, [high_cutoff_notch,
    low_cutoff_notch], btype='stop')
57.
58.         x_filt = signal.filtfilt(b, a, x.T)
59.
60.         if plot:
61.             t = np.arange(0, len(x) / samplerate, 1 /
    samplerate)
62.             time = np.array([i/1000 for i in
    range(0,1500,1)])
63.             plt.plot(time, x[0:1500])
64.             plt.plot(time, x_filt[0:1500].T, 'k')
65.             plt.autoscale(tight=True)
66.             plt.xlabel('Time')
67.             plt.ylabel('Amplitude (mV)')
68.             plt.show()
69.
70.         return x_filt
71.
72.
73.
74.     #THIS FUNCTION CALCULATES THE FEATURES
75.     def features(gest_filtered):
76.       l_mav = []
77.       l_max_val = []
78.       l_min_val = []
79.       l_rms = []
80.       l_var = []
81.       l_damv = []
```

```
82.       l_dvarv = []
83.       l_iasd = []
84.       l_ie = []
85.
86.       for i in range(0, len(gest_filtered), 100):
87.         signal = gest_filtered[i:i+100]
88.         n = len(signal)
89.
90.         #calculamos MAV
91.         mav = np.sum(np.abs(signal))/n
92.         l_mav.append(mav)
93.
94.         #calculamos max y min
95.         max_val = signal.max()
96.         l_max_val.append(max_val)
97.         min_val = signal.min()
98.         l_min_val.append(min_val)
99.
100.        #calculamos RMS
101.        rms = np.sqrt(np.sum(np.square(signal))/n)
102.        l_rms.append(rms)
103.
104.        #calculamos VAR
105.        var = np.var(signal)
106.        l_var.append(var)
107.
108.        # Derivada de signal
109.        signal_1 = signal[1:]
110.        signal_q = signal[:-1]
111.
112.        #calculamos DAMV
113.        damv = (np.sum(np.abs(signal_1 - signal_q)))/(n-1)
114.        l_damv.append(damv)
115.
116.        #calculamos DVARV
117.        dvarv = (np.sum(np.square(signal_1 - signal_q)))/(n-
    2)
118.        l_dvarv.append(dvarv)
119.
120.        #calculamos IASD
121.        signal_prima = signal_1 - signal_q
122.        signal_prima_1 = signal_prima[1:]
123.        signal_prima_q = signal_prima[:-1]
124.        iasd = np.sum(np.abs(signal_prima_1 -
    signal_prima_q))
125.        l_iasd.append(iasd)
126.
127.        #calculamos IE
128.        ie = np.sum(np.exp(signal))
129.        l_ie.append(ie)
130.
```

```
131.      return l_mav, l_max_val, l_min_val, l_rms, l_var,
   l_damv, l_dvarv, l_iasd, l_ie
132.
133.
134.
135.     #THIS FUNCTION CALCULATES THE FEATURES, WITH A SAMPLED
   SIGNAL
136.     def features_2(gest_filtered):
137.      l_mav = []
138.      l_max_val = []
139.      l_min_val = []
140.      l_rms = []
141.      l_var = []
142.      l_damv = []
143.      l_dvarv = []
144.      l_iasd = []
145.      l_ie = []
146.      l_mav2 = []
147.      l_max_val2 = []
148.      l_min_val2 = []
149.      l_rms2 = []
150.      l_var2 = []
151.      l_damv2 = []
152.      l_dvarv2 = []
153.      l_iasd2 = []
154.      l_ie2 = []
155.
156.      for i in range(0, len(gest_filtered), 100):
157.        signal = gest_filtered[i:i+50]
158.        signal2 = gest_filtered[i+50:i+100]
159.        n = len(signal)
160.
161.
162.        #calculation of MAV
163.        mav = np.sum(np.abs(signal))/n
164.        l_mav.append(mav)
165.
166.        mav2 = np.sum(np.abs(signal2))/n
167.        l_mav2.append(mav2)
168.
169.
170.        #calculation of MAX and MIN
171.        max_val = signal.max()
172.        l_max_val.append(max_val)
173.        min_val = signal.min()
174.        l_min_val.append(min_val)
175.
176.        max_val2 = signal2.max()
177.        l_max_val2.append(max_val2)
178.        min_val2 = signal2.min()
179.        l_min_val2.append(min_val2)
180.
```

```
181.
182.        #calculation of RMS
183.        rms = np.sqrt(np.sum(np.square(signal))/n)
184.        l_rms.append(rms)
185.
186.        rms2 = np.sqrt(np.sum(np.square(signal2))/n)
187.        l_rms2.append(rms2)
188.
189.
190.        #calculation of VAR
191.        var = np.var(signal)
192.        l_var.append(var)
193.
194.        var2 = np.var(signal2)
195.        l_var2.append(var2)
196.
197.
198.        #signal and signal2 derivate
199.        signal_1 = signal[1:]
200.        signal_q = signal[:-1]
201.
202.        signal2_1 = signal2[1:]
203.        signal2_q = signal2[:-1]
204.
205.
206.        #calculation of DAMV
207.        damv = (np.sum(np.abs(signal_1 - signal_q)))/(n-1)
208.        l_damv.append(damv)
209.
210.        damv2 = (np.sum(np.abs(signal2_1 - signal2_q)))/(n-
    1)
211.        l_damv2.append(damv2)
212.
213.
214.        #calculation of DVARV
215.        dvarv = (np.sum(np.square(signal_1 - signal_q)))/(n-
    2)
216.        l_dvarv.append(dvarv)
217.
218.        dvarv2 = (np.sum(np.square(signal2_1 -
    signal2_q)))/(n-2)
219.        l_dvarv2.append(dvarv2)
220.
221.
222.        #Signal and signal2 second derivate
223.        signal_prima = signal_1 - signal_q
224.        signal_prima_1 = signal_prima[1:]
225.        signal_prima_q = signal_prima[:-1]
226.
227.        signal2_prima = signal2_1 - signal2_q
228.        signal2_prima_1 = signal2_prima[1:]
229.        signal2_prima_q = signal2_prima[:-1]
```

```
230.
231.
232.        #calculation of IASD
233.        iasd = np.sum(np.abs(signal_prima_1 -
   signal_prima_q))
234.        l_iasd.append(iasd)
235.
236.        iasd2 = np.sum(np.abs(signal2_prima_1 -
   signal2_prima_q))
237.        l_iasd2.append(iasd2)
238.
239.
240.        #calculation of IE
241.        ie = np.sum(np.exp(signal))
242.        l_ie.append(ie)
243.
244.        ie2 = np.sum(np.exp(signal2))
245.        l_ie2.append(ie2)
246.
247.
248.      return l_mav, l_max_val, l_min_val, l_rms, l_var,
   l_damv, l_dvarv, l_iasd, l_ie, l_mav2, l_max_val2,
   l_min_val2, l_rms2, l_var2, l_damv2, l_dvarv2, l_iasd2, l_ie2
249.
250.
251.
252.     #THIS FUNCTION CALCULATES THE FEATURES FOR THE
   EVALUATION
253.     def preprocessing_3(gest_filtered):
254.      l_mav = []
255.      l_max_val = []
256.      l_min_val = []
257.      l_rms = []
258.      l_var = []
259.      l_damv = []
260.      l_dvarv = []
261.      l_iasd = []
262.      l_ie = []
263.      l_mav2 = []
264.      l_max_val2 = []
265.      l_min_val2 = []
266.      l_rms2 = []
267.      l_var2 = []
268.      l_damv2 = []
269.      l_dvarv2 = []
270.      l_iasd2 = []
271.      l_ie2 = []
272.
273.      for i in range(0, 100, 100):
274.        signal = gest_filtered[i:i+50]
275.        signal2 = gest_filtered[i+50:i+100]
276.        n = len(signal)
```

```
277.
278.
279.        #calculation of MAV
280.        mav = np.sum(np.abs(signal))/n
281.        l_mav.append(mav)
282.
283.        mav2 = np.sum(np.abs(signal2))/n
284.        l_mav2.append(mav2)
285.
286.
287.        #calculation of MAX and MIN
288.        max_val = signal.max()
289.        l_max_val.append(max_val)
290.        min_val = signal.min()
291.        l_min_val.append(min_val)
292.
293.        max_val2 = signal2.max()
294.        l_max_val2.append(max_val2)
295.        min_val2 = signal2.min()
296.        l_min_val2.append(min_val2)
297.
298.
299.        #calculation of RMS
300.        rms = np.sqrt(np.sum(np.square(signal))/n)
301.        l_rms.append(rms)
302.
303.        rms2 = np.sqrt(np.sum(np.square(signal2))/n)
304.        l_rms2.append(rms2)
305.
306.
307.        #calculation of VAR
308.        var = np.var(signal)
309.        l_var.append(var)
310.
311.        var2 = np.var(signal2)
312.        l_var2.append(var2)
313.
314.
315.        #signal and signal2 derivate
316.        signal_1 = signal[1:]
317.        signal_q = signal[:-1]
318.
319.        signal2_1 = signal2[1:]
320.        signal2_q = signal2[:-1]
321.
322.
323.        #calculation of DAMV
324.        damv = (np.sum(np.abs(signal_1 - signal_q)))/(n-1)
325.        l_damv.append(damv)
326.
327.        damv2 = (np.sum(np.abs(signal2_1 - signal2_q)))/(n-
    1)
```

```
328.        l_damv2.append(damv2)
329.
330.
331.        #calculation of DVARV
332.        dvarv = (np.sum(np.square(signal_1 - signal_q)))/(n-
   2)
333.        l_dvarv.append(dvarv)
334.
335.        dvarv2 = (np.sum(np.square(signal2_1 -
   signal2_q)))/(n-2)
336.        l_dvarv2.append(dvarv2)
337.
338.
339.        #Signal and signal2 second derivate
340.        signal_prima = signal_1 - signal_q
341.        signal_prima_1 = signal_prima[1:]
342.        signal_prima_q = signal_prima[:-1]
343.
344.        signal2_prima = signal2_1 - signal2_q
345.        signal2_prima_1 = signal2_prima[1:]
346.        signal2_prima_q = signal2_prima[:-1]
347.
348.
349.        #calculation of IASD
350.        iasd = np.sum(np.abs(signal_prima_1 -
   signal_prima_q))
351.        l_iasd.append(iasd)
352.
353.        iasd2 = np.sum(np.abs(signal2_prima_1 -
   signal2_prima_q))
354.        l_iasd2.append(iasd2)
355.
356.
357.        #calculation of IE
358.        ie = np.sum(np.exp(signal))
359.        l_ie.append(ie)
360.
361.        ie2 = np.sum(np.exp(signal2))
362.        l_ie2.append(ie2)
363.
364.
365.     return l_mav, l_max_val, l_min_val, l_rms, l_var,
   l_damv, l_dvarv, l_iasd, l_ie, l_mav2, l_max_val2,
   l_min_val2, l_rms2, l_var2, l_damv2, l_dvarv2, l_iasd2, l_ie2
```

C2. Signal processing (processing.py)

```
1. import pyrebase
2. import os.path
3. import matplotlib.pyplot as plt
4. import numpy as np
5. import functions as fnc
6. import scipy as sp
7. from scipy import signal
8. import pandas as pd
9. import seaborn as sns
10.
11.
12.     ##########################################################
13.     ##########################################################
14.            #LINKS TO FIREBASE AND REAL-TIME DB
15.     ##########################################################
16.     ##########################################################
17.
18.     #Credentials to link Firebase project with Python
19.     firebaseConfig = {
20.      "apiKey": "AIzaSyAwBffvO_waRFkEFjO8tJpDE4z8erMECQ0",
21.      "authDomain": "aehrich-database.firebaseapp.com",
22.      "projectId": "aehrich-database",
23.      "databaseURL": "https://aehrich-database-default-
  rtdb.europe-west1.firebasedatabase.app/",
24.      "storageBucket": "aehrich-database.appspot.com",
25.      "messagingSenderId": "710072352888",
26.      "appId": "1:710072352888:web:7aace7bb41b5847dcc6d8d"
27.     }
28.
29.     #Initializes interaction with realtime database
30.     firebase = pyrebase.initialize_app(firebaseConfig)
31.     db = firebase.database()
32.
33.
34.     ##########################################################
35.     ##########################################################
36.               #IMPORTS SIGNALS FROM FIREBASE
37.     ##########################################################
38.     ##########################################################
39.
40.     #Get recorded signal from Fyrebase
41.     data1 = db.child("EMGdata").child("Gesture1").get() #--
  > gets all the content of the node (path)
42.     data2 = db.child("EMGdata").child("Gesture2").get()
43.     data3 = db.child("EMGdata").child("Gesture3").get()
44.     data4 = db.child("EMGdata").child("Gesture4").get()
45.     data5 = db.child("EMGdata").child("Gesture5").get()
46.
47.
48.     ##########################################################
```

```
49.      ##########################################################
50.             #DELETE KEY FROM JSON AND APPEND TO LIST
51.      ##########################################################
52.      ##########################################################
53.
54.      #Lists that stores the recorded data
55.      gest1= []
56.      gest2= []
57.      gest3= []
58.      gest4= []
59.      gest5= []
60.
61.      #Removes key from JSON and appends in list
62.      for val1 in data1.each():
63.       gest1.append(val1.val())
64.
65.      for val2 in data2.each():
66.       gest2.append(val2.val())
67.
68.      for val3 in data3.each():
69.       gest3.append(val3.val())
70.
71.      for val4 in data4.each():
72.       gest4.append(val4.val())
73.
74.      for val5 in data5.each():
75.       gest5.append(val5.val())
76.
77.
78.      ##########################################################
79.      ##########################################################
80.                    #CORRECT GESTURES LENGHT
81.      ##########################################################
82.      ##########################################################
83.
84.      gest1 = fnc.correct_length(gest1)
85.      gest2 = fnc.correct_length(gest2)
86.      gest4 = fnc.correct_length(gest4)
87.      gest5 = fnc.correct_length(gest5)
88.
89.
90.      ##########################################################
91.      ##########################################################
92.                               #REMOVE OFFSET
93.      ##########################################################
94.      ##########################################################
95.
96.      #Creates time variable
97.      time = np.array([i/1000 for i in
   range(0,len(gest1),1)])
98.
99.      gest1= fnc.rmv_mean(gest1,time)
```

```
    gest2= fnc.rmv_mean(gest2,time)
    gest3= fnc.rmv_mean(gest3,time)
    gest4= fnc.rmv_mean(gest4,time)
    gest5= fnc.rmv_mean(gest5,time)


    ##########################################################
    ##########################################################
                              #FILTER
    ##########################################################
    ##########################################################

    gest_filtered_1 = fnc.bp_filter(gest1, 20, 450, 1000,
  plot=False)
    gest_filtered_2 = fnc.bp_filter(gest2, 20, 450, 1000,
  plot=False)
    gest_filtered_3 = fnc.bp_filter(gest3, 20, 450, 1000,
  plot=False)
    gest_filtered_4 = fnc.bp_filter(gest4, 20, 450, 1000,
  plot=False)
    gest_filtered_5 = fnc.bp_filter(gest5, 20, 450, 1000,
  plot=False)


    gest_filtered_1 = fnc.notch_filter(gest_filtered_1,
  1000, plot=False)
    gest_filtered_2 = fnc.notch_filter(gest_filtered_2,
  1000, plot=False)
    gest_filtered_3 = fnc.notch_filter(gest_filtered_3,
  1000, plot=False)
    gest_filtered_4 = fnc.notch_filter(gest_filtered_4,
  1000, plot=False)
    gest_filtered_5 = fnc.notch_filter(gest_filtered_5,
  1000, plot=False)


    ##########################################################
    ##########################################################
                          #PRE-PROCESSING
    ##########################################################
    ##########################################################

    preproc_1 = fnc.features(gest_filtered_1)
    preproc_2 = fnc.features(gest_filtered_2)
    preproc_3 = fnc.features(gest_filtered_3)
    preproc_4 = fnc.features(gest_filtered_4)
    preproc_5 = fnc.features(gest_filtered_5)


    ##########################################################
    ##########################################################
                          #DATAFRAME
```

```
142.     ########################################################
143.     ########################################################
144.
145.     df_1 = pd.DataFrame(preproc_1).T
146.     df_1.index = np.arange(1, len(df_1)+1)
147.     df_1['gesture'] = 1
148.
149.     df_2 = pd.DataFrame(preproc_2).T
150.     df_2.index = np.arange(1, len(df_2)+1)
151.     df_2['gesture'] = 2
152.
153.     df_3 = pd.DataFrame(preproc_3).T
154.     df_3.index = np.arange(1, len(df_3)+1)
155.     df_3['gesture'] = 3
156.
157.     df_4 = pd.DataFrame(preproc_4).T
158.     df_4.index = np.arange(1, len(df_4)+1)
159.     df_4['gesture'] = 4
160.
161.     df_5 = pd.DataFrame(preproc_5).T
162.     df_5.index = np.arange(1, len(df_5)+1)
163.     df_5['gesture'] = 5
164.
165.     frames = [df_1, df_2, df_3, df_4, df_5]
166.
167.     df_gest = pd.concat(frames)
168.     df_gest.columns = ['MAV', 'MAX', 'MIN', 'RMS', 'VAR',
    'DAMV', 'DVARV', 'IASD', 'IE', 'gesture']
169.
170.     # Generates a csv with the features to evaluate the
    model
171.     df_gest = df_gest.reset_index(drop=True)
172.     df_gest.to_csv('datos.csv', index=False)
173.
174.
175.     ########################################################
176.     ########################################################
177.                         #CORRELATION MATRIX
178.     ########################################################
179.     ########################################################
180.
181.     corrMatrix = df_gest.corr()
182.     sns.heatmap(corrMatrix, annot=True)
183.     plt.show()
```

C3. Signal processing with split data (processing_2.py)

```
1. import pyrebase
2. import os.path
3. import matplotlib.pyplot as plt
4. import numpy as np
5. import functions as fnc
6. import scipy as sp
7. from scipy import signal
8. import pandas as pd
9. import seaborn as sns
10.
11.
12.     ########################################################
13.     ########################################################
14.            #LINKS TO FIREBASE AND REAL-TIME DB
15.     ########################################################
16.     ########################################################
17.
18.     #Credentials to link Firebase project with Python
19.     firebaseConfig = {
20.      "apiKey": "AIzaSyAwBffvO_waRFkEFjO8tJpDE4z8erMECQ0",
21.      "authDomain": "aehrich-database.firebaseapp.com",
22.      "projectId": "aehrich-database",
23.      "databaseURL": "https://aehrich-database-default-
   rtdb.europe-west1.firebasedatabase.app/",
24.      "storageBucket": "aehrich-database.appspot.com",
25.      "messagingSenderId": "710072352888",
26.      "appId": "1:710072352888:web:7aace7bb41b5847dcc6d8d"
27.     }
28.
29.     #Initializes interaction with realtime database
30.     firebase = pyrebase.initialize_app(firebaseConfig)
31.     db = firebase.database()
32.
33.
34.     ########################################################
35.     ########################################################
36.               #IMPORTS SIGNALS FROM FIREBASE
37.     ########################################################
38.     ########################################################
39.
40.     #Get recorded signal from Fyrebase
41.     data1 = db.child("EMGdata").child("Gesture1").get() #--
   > gets all the content of the node (path)
42.     data2 = db.child("EMGdata").child("Gesture2").get()
43.     data3 = db.child("EMGdata").child("Gesture3").get()
44.     data4 = db.child("EMGdata").child("Gesture4").get()
45.     data5 = db.child("EMGdata").child("Gesture5").get()
46.
47.
48.     ########################################################
```

```
49.     ##########################################################
50.             #DELETE KEY FROM JSON AND APPEND TO LIST
51.     ##########################################################
52.     ##########################################################
53.
54.     #Lists that stores the recorded data
55.     gest1= []
56.     gest2= []
57.     gest3= []
58.     gest4= []
59.     gest5= []
60.
61.     #Removes key from JSON and appends in list
62.     for val1 in data1.each():
63.      gest1.append(val1.val())
64.
65.     for val2 in data2.each():
66.      gest2.append(val2.val())
67.
68.     for val3 in data3.each():
69.      gest3.append(val3.val())
70.
71.     for val4 in data4.each():
72.      gest4.append(val4.val())
73.
74.     for val5 in data5.each():
75.      gest5.append(val5.val())
76.
77.
78.     ##########################################################
79.     ##########################################################
80.                     #CORRECT GESTURES LENGHT
81.     ##########################################################
82.     ##########################################################
83.
84.     gest1 = fnc.correct_length(gest1)
85.     gest2 = fnc.correct_length(gest2)
86.     gest4 = fnc.correct_length(gest4)
87.     gest5 = fnc.correct_length(gest5)
88.
89.
90.     ##########################################################
91.     ##########################################################
92.                             #REMOVE OFFSET
93.     ##########################################################
94.     ##########################################################
95.
96.     #Creates time variable
97.     time = np.array([i/1000 for i in
   range(0,len(gest5),1)])
98.
99.
```

```
    gest1= fnc.rmv_mean(gest1,time)
    gest2= fnc.rmv_mean(gest2,time)
    gest3= fnc.rmv_mean(gest3,time)
    gest4= fnc.rmv_mean(gest4,time)
    gest5= fnc.rmv_mean(gest5,time)


    ##########################################################
    ##########################################################
                            #FILTER
    ##########################################################
    ##########################################################
    gest_filtered_1 = fnc.bp_filter(gest1, 20, 450, 1000,
  plot=False)
    gest_filtered_2 = fnc.bp_filter(gest2, 20, 450, 1000,
  plot=False)
    gest_filtered_3 = fnc.bp_filter(gest3, 20, 450, 1000,
  plot=False)
    gest_filtered_4 = fnc.bp_filter(gest4, 20, 450, 1000,
  plot=False)
    gest_filtered_5 = fnc.bp_filter(gest5, 20, 450, 1000,
  plot=False)


    gest_filtered_1 = fnc.notch_filter(gest_filtered_1,
  1000, plot=False)
    gest_filtered_2 = fnc.notch_filter(gest_filtered_2,
  1000, plot=False)
    gest_filtered_3 = fnc.notch_filter(gest_filtered_3,
  1000, plot=False)
    gest_filtered_4 = fnc.notch_filter(gest_filtered_4,
  1000, plot=False)
    gest_filtered_5 = fnc.notch_filter(gest_filtered_5,
  1000, plot=False)


    ##########################################################
    ##########################################################
                          #PRE-PROCESSING
    ##########################################################
    ##########################################################

    preproc_1 = fnc.features_2(gest_filtered_1)
    preproc_2 = fnc.features_2(gest_filtered_2)
    preproc_3 = fnc.features_2(gest_filtered_3)
    preproc_4 = fnc.features_2(gest_filtered_4)
    preproc_5 = fnc.features_2(gest_filtered_5)


    ##########################################################
    ##########################################################
```

```
142.                          #DATAFRAMES
143.     ########################################################
144.     ########################################################
145.
146.     #dataframes with all the information
147.     df3_1 = pd.DataFrame(preproc_1).T
148.     df3_1.index = np.arange(1, len(df3_1)+1)
149.     df3_1['gesture'] = 1
150.
151.     df3_2 = pd.DataFrame(preproc_2).T
152.     df3_2.index = np.arange(1, len(df3_2)+1)
153.     df3_2['gesture'] = 2
154.
155.     df3_3 = pd.DataFrame(preproc_3).T
156.     df3_3.index = np.arange(1, len(df3_3)+1)
157.     df3_3['gesture'] = 3
158.
159.     df3_4 = pd.DataFrame(preproc_4).T
160.     df3_4.index = np.arange(1, len(df3_4)+1)
161.     df3_4['gesture'] = 4
162.
163.     df3_5 = pd.DataFrame(preproc_5).T
164.     df3_5.index = np.arange(1, len(df3_5)+1)
165.     df3_5['gesture'] = 5
166.
167.
168.     #concatenates df with all values
169.     frame_all = [df3_1, df3_2, df3_3, df3_4, df3_5]
170.     df_all = pd.concat(frame_all)
171.     df_all.columns = ['MAV', 'MAX', 'MIN', 'RMS', 'VAR',
   'DAMV', 'DVARV', 'IASD', 'IE', 'MAV 2', 'MAX 2', 'MIN 2',
   'RMS 2', 'VAR 2', 'DAMV 2', 'DVARV 2', 'IASD 2', 'IE 2',
   'gesture']
172.     df_all = df_all.reset_index(drop=True)
173.
174.
175.     #removes the out of range measures
176.     df_all= df_all[df_all['MAX']<1.64].reset_index(drop =
   True)
177.     df_all= df_all[df_all['MIN']>-1.64].reset_index(drop =
   True)
178.
179.
180.     # Generates a csv with the features to evaluate the
   model
181.     df_all.to_csv('data.csv', index=False)
182.
183.
184.     ########################################################
185.     ########################################################
186.                        #CORRELATION MATRIX
187.     ########################################################
```

```
188.     ##########################################################
189.
190.     #correlation matrix for df with all values
191.     df_all = df_all.drop(columns=['gesture'])
192.     corrMatrix_all = df_all.corr()
193.     sns.heatmap(corrMatrix_all, annot=True)
194.     plt.show()
```

C4. Features comparison (compare.py)

```
1. import pandas as pd
2.
3. df = pd.read_csv('data.csv')
4.
5. df = df.groupby('gesture',as_index= False)['MAV', 'MAX',
   'MIN', 'RMS', 'VAR', 'DAMV', 'DVARV', 'IASD', 'IE', 'MAV 2',
   'MAX 2', 'MIN 2', 'RMS 2', 'VAR 2', 'DAMV 2', 'DVARV 2',
   'IASD 2', 'IE 2'].mean()
6. print(df)
7.
8. df.to_excel('resumen.xlsx')
```

C5. Logistic Regression Model (LogisticRegression.py)

```
1. import pandas as pd
2. from sklearn.model_selection import train_test_split
3. from sklearn.metrics import accuracy_score, confusion_matrix,
   classification_report
4. from sklearn.linear_model import LogisticRegression
5. from sklearn.preprocessing import StandardScaler
6. from sklearn.pipeline import Pipeline
7. import numpy as np
8. import matplotlib.pyplot as plt
9. import seaborn as sns
10.      from random import randint
11.
12.      df = pd.read_csv('data.csv')
13.      y = df.pop('gesture').values
14.      X = df.values
15.
16.      X_train, X_test, y_train, y_test = train_test_split(X,
   y, test_size=0.20, random_state=42)
17.      pipe = Pipeline([('scaler', StandardScaler()), ('log',
   LogisticRegression(random_state=0, max_iter=1000))])
18.      pipe.fit(X_train, y_train)
19.
20.      y_pred = pipe.predict(X_test)
21.      accuracy1 = accuracy_score(y_test, y_pred)
22.      print('Model: Logistic Regression')
23.      print('[Train: 80%, Test: 20%]')
24.      print('TEST ACCURACY IS:    ' + str(accuracy1*100) + '
   %')
25.
26.
27.
28.      y_train_pred = pipe.predict(X_train)
29.      accuracy2 = accuracy_score(y_train, y_train_pred)
30.      print('TRAINING ACCURACY IS:    ' + str(accuracy2*100)
   + ' %')
31.
32.
33.      y_random = [randint(1,5) for _ in range(len(y_test))]
34.      accuracy3 = accuracy_score(y_test, y_random)
35.      print('RANDOMNESS EVALUATION IS:    ' +
   str(accuracy3*100) + '%')
36.
37.
38.
39.      print('CLASSIFICATION REPORT:')
40.      print(classification_report(y_test, y_pred))
41.
42.
43.
44.      conf_mat = confusion_matrix(y_test, y_pred)
```

```
45.      print(conf_mat)
46.      group_counts = ["{0:0.0f}".format(value) for value in
   conf_mat.flatten()]
47.      group_percentages = ["{0:.2%}".format(value) for value
   in conf_mat.flatten()/np.sum(conf_mat)]
48.      labels = [f"{v1}\n{v2}\n" for v1, v2 in
   zip(group_counts,group_percentages)]
49.      labels = np.asarray(labels).reshape(5,5)
50.      ax = sns.heatmap(conf_mat, annot=labels, fmt='',
   cmap='Blues')
51.      ax.set_title('Confusion Matrix \n\n')
52.      ax.set_xlabel('\nPredicted Gesture')
53.      ax.set_ylabel('Actual Gesture')
54.      ax.xaxis.set_ticklabels(['Gesture 1','Gesture 2',
   'Gesture 3', 'Gesture 4', 'Gesture 5'])
55.      ax.yaxis.set_ticklabels(['Gesture 1','Gesture 2',
   'Gesture 3', 'Gesture 4', 'Gesture 5'])
56.      plt.show()
57.
58.
59.
60.      import pickle
61.      file_pipe = open('log_reg.obj', 'wb')
62.      pickle.dump(pipe, file_pipe)
```

C6. Support Vector Classifier Model (SVC.py)

```
import pandas as pd
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
      from random import randint

      df = pd.read_csv('data.csv')
      y = df.pop('gesture').values
      X = df.values

      X_train, X_test, y_train, y_test = train_test_split(X,
y, test_size=0.33, random_state=42)
      pipe = Pipeline([('scaler', StandardScaler()), ('svc',
SVC(random_state=0))])
      pipe.fit(X_train, y_train)

      y_pred = pipe.predict(X_test)
      accuracy1 = accuracy_score(y_test, y_pred)
      print('Model: Support Vector Classifier')
      print('[Train: 67%, Test: 33%]')
      print('TEST ACCURACY IS: ' + str(accuracy1*100) + ' %')



      y_train_pred = pipe.predict(X_train)
      accuracy2 = accuracy_score(y_train, y_train_pred)
      print('TRAINING ACCURACY IS:    ' + str(accuracy2*100)
+ ' %')


      y_random = [randint(1,5) for _ in range(len(y_test))]
      accuracy3 = accuracy_score(y_test, y_random)
      print('RANDOMNESS EVALUATION IS:    ' +
str(accuracy3*100) + '%')



      print('CLASSIFICATION REPORT:')
      print(classification_report(y_test, y_pred))



      conf_mat = confusion_matrix(y_test, y_pred)
      print(conf_mat)
```

```
46.      group_counts = ["{0:0.0f}".format(value) for value in
   conf_mat.flatten()]
47.      group_percentages = ["{0:.2%}".format(value) for value
   in conf_mat.flatten()/np.sum(conf_mat)]
48.      labels = [f"{v1}\n{v2}\n" for v1, v2 in
   zip(group_counts,group_percentages)]
49.      labels = np.asarray(labels).reshape(5,5)
50.      ax = sns.heatmap(conf_mat, annot=labels, fmt='',
   cmap='Blues')
51.      ax.set_title('Confusion Matrix \n\n')
52.      ax.set_xlabel('\nPredicted Gesture')
53.      ax.set_ylabel('Actual Gesture')
54.      ax.xaxis.set_ticklabels(['Gesture 1','Gesture 2',
   'Gesture 3', 'Gesture 4', 'Gesture 5'])
55.      ax.yaxis.set_ticklabels(['Gesture 1','Gesture 2',
   'Gesture 3', 'Gesture 4', 'Gesture 5'])
56.      plt.show()
```

C7. K-Nearest Neighbours Model (KNN.py)

```
1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. import seaborn as sns
5. from sklearn.neighbors import KNeighborsClassifier
6. from sklearn.model_selection import train_test_split
7. from sklearn.metrics import accuracy_score,
   classification_report, confusion_matrix
8. from sklearn.pipeline import Pipeline
9. from sklearn.preprocessing import StandardScaler
10.      from random import randint
11.
12.      df = pd.read_csv('data.csv')
13.
14.      y = df.pop('gesture').values
15.      X = df.values
16.      X_train, X_test, y_train, y_test = train_test_split(X,
   y, test_size=0.33, random_state=42)
17.
18.      clf = KNeighborsClassifier(n_neighbors=5)
19.      pipe = Pipeline([('scaler', StandardScaler()), ('knn',
   clf)])
20.
21.      pipe.fit(X_train, y_train)
22.
23.      y_pred = pipe.predict(X_test)
24.
25.      accuracy1 = accuracy_score(y_test, y_pred)
26.      print('Model: K-Nearest Neighbours')
27.      print('[Train: 67%, Test: 33%]')
28.      print('TEST ACCURACY IS: ' + str(accuracy1*100) + ' %')
29.
30.
31.
32.      y_train_pred = pipe.predict(X_train)
33.      accuracy2 = accuracy_score(y_train, y_train_pred)
34.      print('TRAINING ACCURACY IS:     ' + str(accuracy2*100)
   + ' %')
35.
36.
37.      y_random = [randint(1,5) for _ in range(len(y_test))]
38.      accuracy3 = accuracy_score(y_test, y_random)
39.      print('RANDOMNESS EVALUATION IS:     ' +
   str(accuracy3*100) + '%')
40.
41.
42.
43.      print('CLASSIFICATION REPORT:')
44.      print(classification_report(y_test, y_pred))
45.
```

```
46.
47.
48.      conf_mat = confusion_matrix(y_test, y_pred)
49.      print(conf_mat)
50.      group_counts = ["{0:0.0f}".format(value) for value in
   conf_mat.flatten()]
51.      group_percentages = ["{0:.2%}".format(value) for value
   in conf_mat.flatten()/np.sum(conf_mat)]
52.      labels = [f"{v1}\n{v2}\n" for v1, v2 in
   zip(group_counts,group_percentages)]
53.      labels = np.asarray(labels).reshape(5,5)
54.      ax = sns.heatmap(conf_mat, annot=labels, fmt='',
   cmap='Blues')
55.      ax.set_title('Confusion Matrix \n\n')
56.      ax.set_xlabel('\nPredicted Gesture')
57.      ax.set_ylabel('Actual Gesture')
58.      ax.xaxis.set_ticklabels(['Gesture 1','Gesture 2',
   'Gesture 3', 'Gesture 4', 'Gesture 5'])
59.      ax.yaxis.set_ticklabels(['Gesture 1','Gesture 2',
   'Gesture 3', 'Gesture 4', 'Gesture 5'])
60.      plt.show()
```

C8. Evaluation of the Model (evaluation.py)

The real classification model code:

```
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import Pipeline

df = pd.read_csv('data.csv')
y = df.pop('gesture').values
X = df.values

    pipe = Pipeline([('scaler', StandardScaler()), ('log', LogisticRegression(random_state=0, max_iter=1000))])
    pipe.fit(X, y)

    import pickle
    file_pipe = open('log_reg.pkl', 'wb')
    pickle.dump(pipe, file_pipe)
```

The evaluation code:

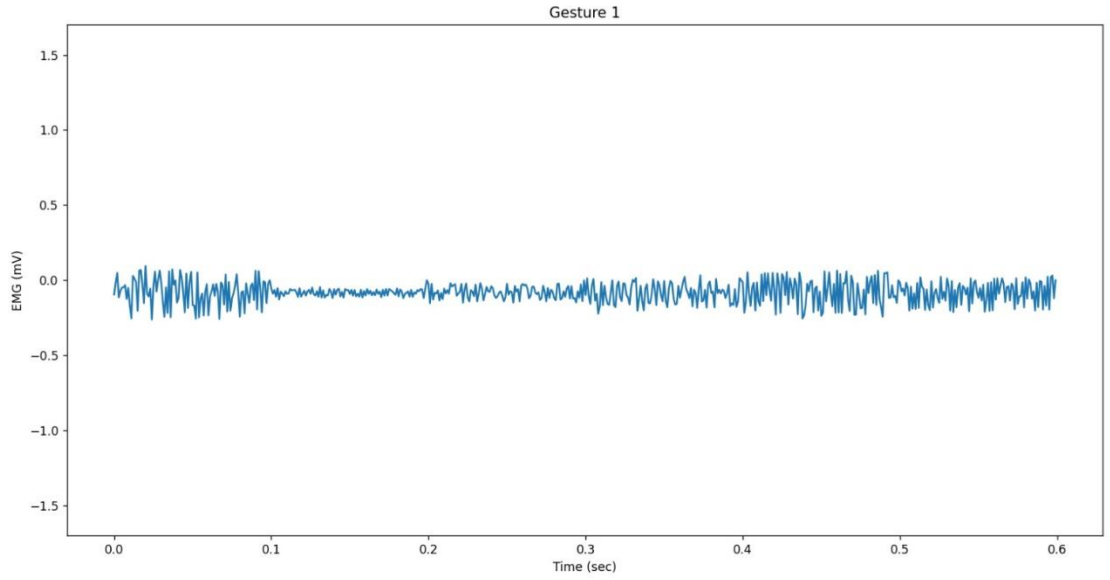
```
import pickle
import pyrebase
import os.path
import numpy as np
import functions as fnc
import scipy as sp
from scipy import signal
import pandas as pd
import matplotlib.pyplot as plt
    import matplotlib.image as mpimg

    firebaseConfig = {
      "apiKey": "AIzaSyAwBffvO_waRFkEFjO8tJpDE4z8erMECQ0",
      "authDomain": "aehrich-database.firebaseapp.com",
      "projectId": "aehrich-database",
      "databaseURL": "https://aehrich-database-default-rtdb.europe-west1.firebasedatabase.app/",
      "storageBucket": "aehrich-database.appspot.com",
      "messagingSenderId": "710072352888",
      "appId": "1:710072352888:web:7aace7bb41b5847dcc6d8d"
    }

    firebase = pyrebase.initialize_app(firebaseConfig)
    db = firebase.database()

    data = db.child("EMGdata").child("evaluation").get() #-> gets all the content of the node (path)
```

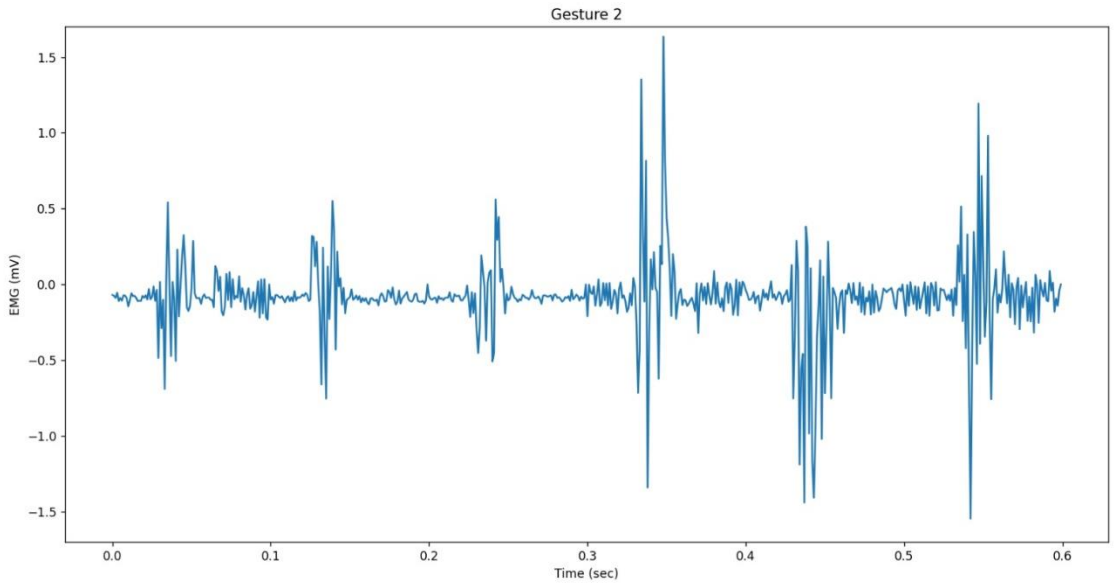
```
26.      gest= []
27.
28.      for val in data.each():
29.       gest.append(val.val())
30.
31.      gest = fnc.correct_length(gest)
32.      time = np.array([i/1000 for i in range(0,len(gest),1)])
33.      gest1= fnc.rmv_mean(gest,time)
34.
35.      gest_filtered = fnc.bp_filter(gest, 20, 450, 1000,
   plot=False)
36.      gest_filtered = fnc.notch_filter(gest_filtered, 1000,
   plot=False)
37.
38.      preproc = fnc.preprocessing_3(gest_filtered)
39.
40.      df = pd.DataFrame(preproc).T
41.      df.index = np.arange(1, len(df)+1)
42.      df.columns = ['MAV', 'MAX', 'MIN', 'RMS', 'VAR',
   'DAMV', 'DVARV', 'IASD', 'IE', 'MAV 2', 'MAX 2', 'MIN 2',
   'RMS 2', 'VAR 2', 'DAMV 2', 'DVARV 2', 'IASD 2', 'IE 2']
43.      df_gest = df.reset_index(drop=True)
44.
45.
46.
47.      X = df.values
48.
49.      filehandler = open('log_reg.pkl', 'rb')
50.      pipe = pickle.load(filehandler)
51.      y = pipe.predict(X)
52.      y2 = pipe.predict_proba(X)
53.
54.      print('Gesture: ' + str(y))
55.
56.      if y == 1:
57.       img = mpimg.imread('gest_pictures/gesture_1.jpeg')
58.       imgplot = plt.imshow(img)
59.       plt.title('Gesture 1: Neutral position')
60.       plt.axis('off')
61.       plt.show()
62.
63.      if y == 2:
64.       img = mpimg.imread('gest_pictures/gesture_2.jpeg')
65.       imgplot = plt.imshow(img)
66.       plt.title('Gesture 2: Fingers extension')
67.       plt.axis('off')
68.       plt.show()
69.
70.      if y == 3:
71.       img = mpimg.imread('gest_pictures/gesture_3.jpeg')
72.       imgplot = plt.imshow(img)
73.       plt.title('Gesture 3: Fingers flexion')
```

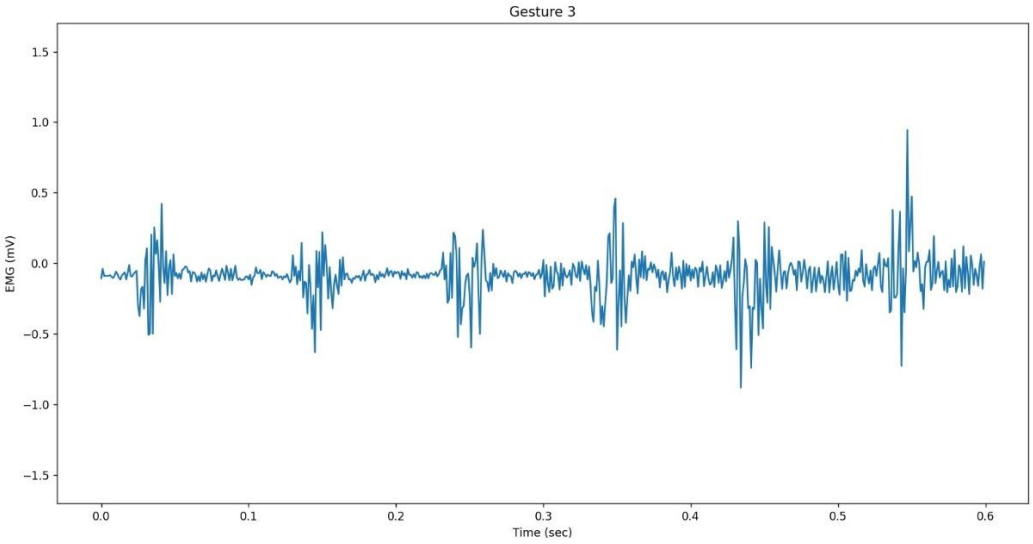
```
74.         plt.axis('off')
75.         plt.show()
76.
77.       if y == 4:
78.         img = mpimg.imread('gest_pictures/gesture_4.jpeg')
79.         imgplot = plt.imshow(img)
80.         plt.title('Gesture 4: Wrist extension')
81.         plt.axis('off')
82.         plt.show()
83.
84.       if y == 5:
85.         img = mpimg.imread('gest_pictures/gesture_5.jpeg')
86.         imgplot = plt.imshow(img)
87.         plt.title('Gesture 5: Ulnar deviation')
88.         plt.axis('off')
89.         plt.show()
90.
91.       np.set_printoptions(precision=2, suppress=True)
92.       print('The probability of having predicted each gesture
    is: ')
93.       print(y2*100)
94.
95.
96.       db.child('EMGdata').child('evaluation').remove()
97.       db.child('EMGdata').child('evaluation').set(0)
```

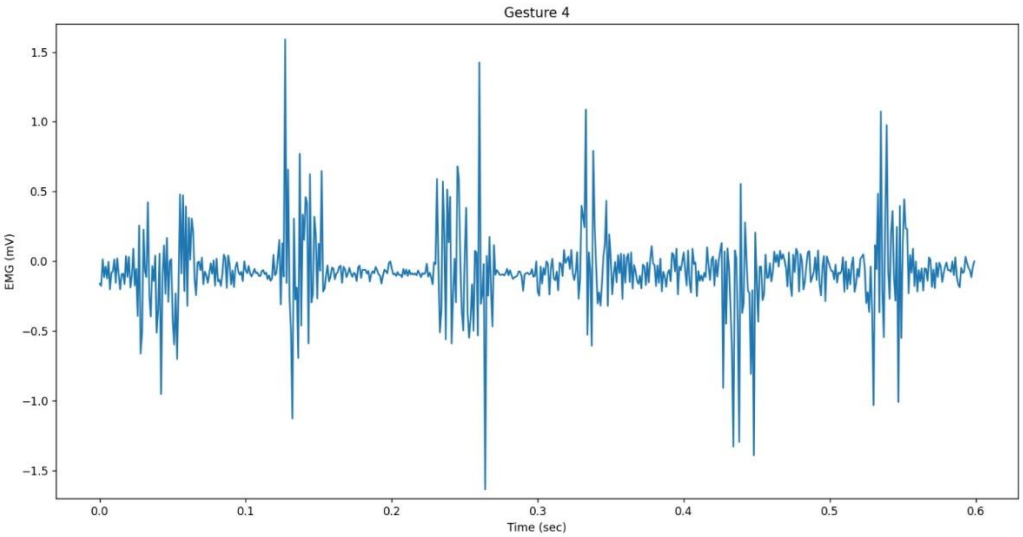
C9. Signal visual representation during the pre-processing

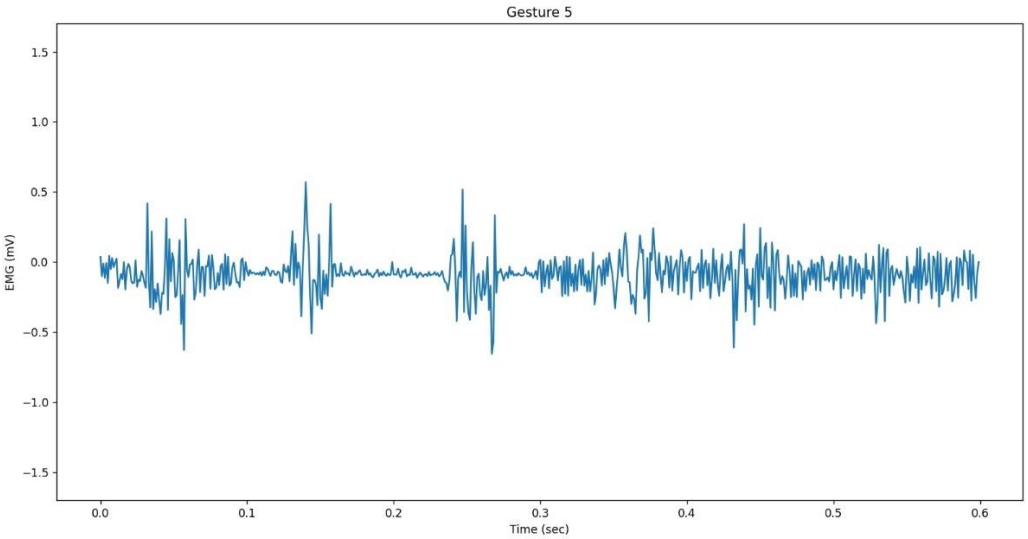
These are the images of a segment of the data of each gesture after having removed the key value and corrected the length.



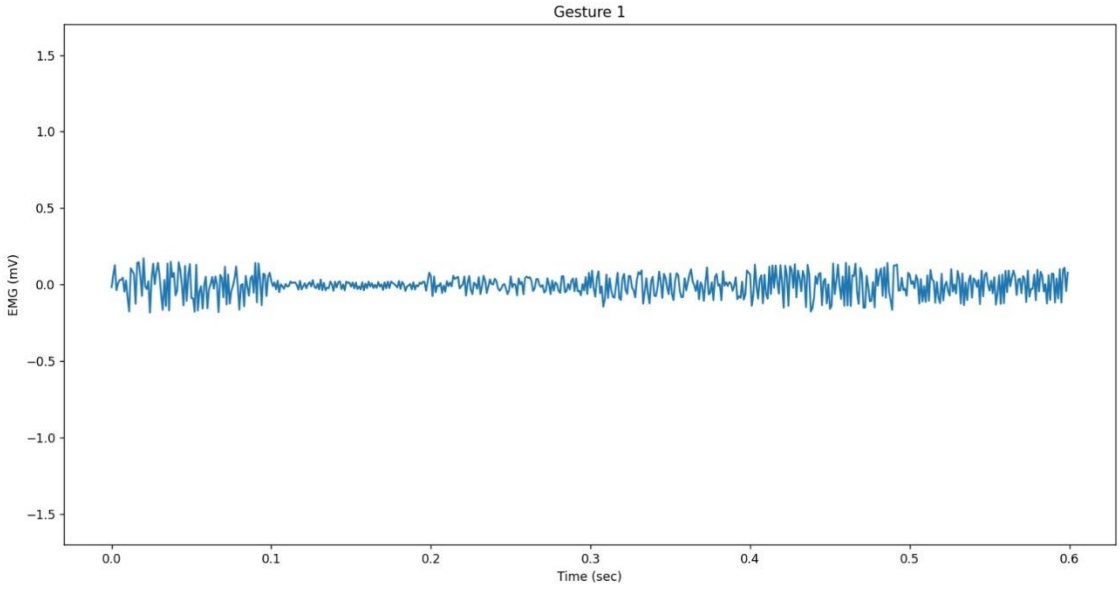


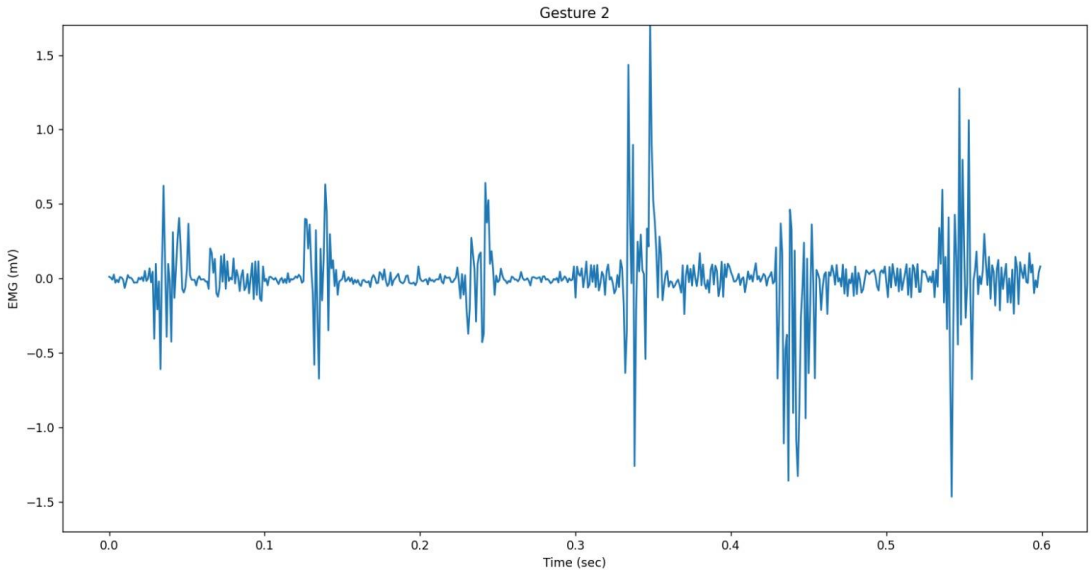

Gesture 3
EMG (mV)
Time (sec)

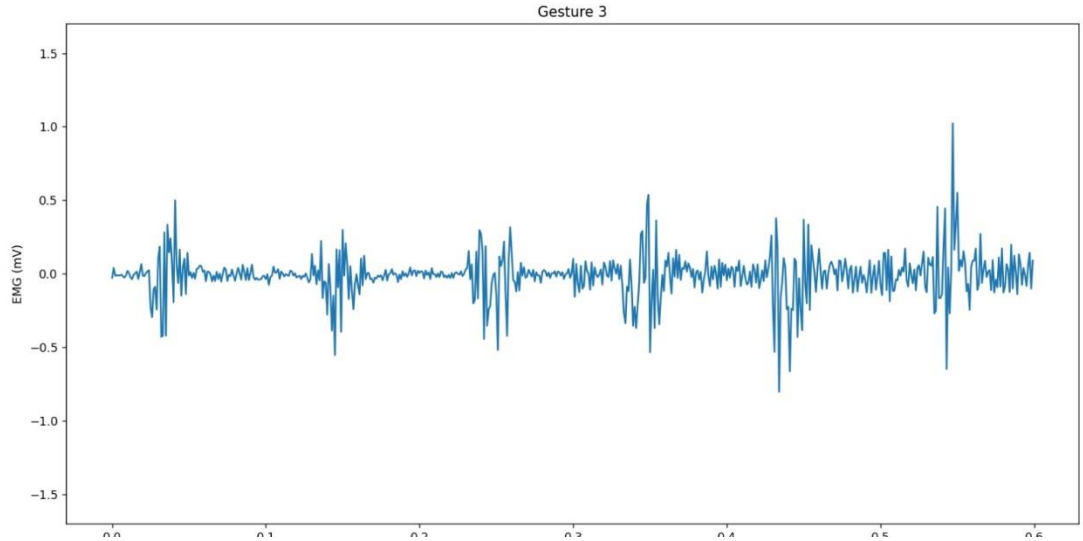

Gesture 4
EMG (mV)
Time (sec)

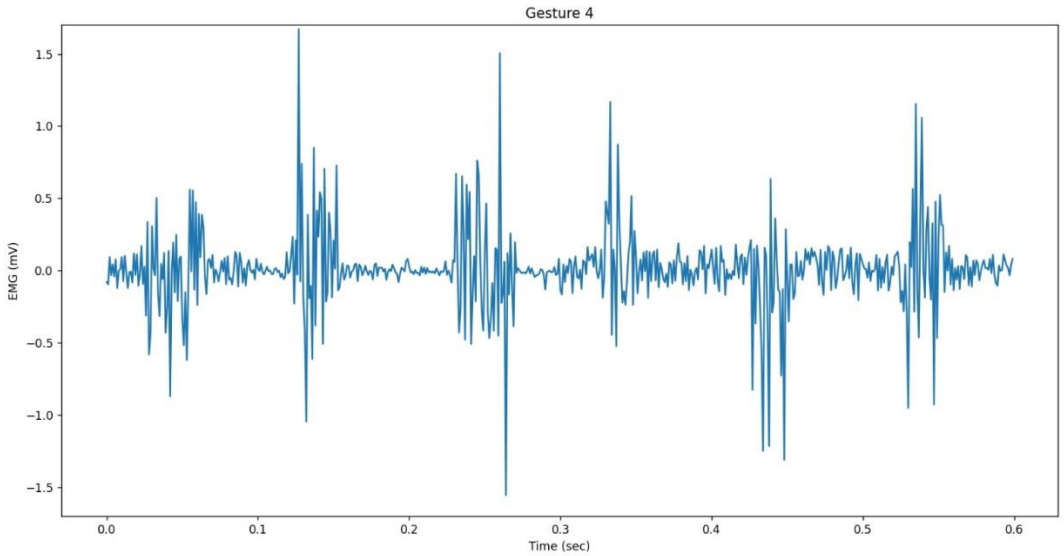

Gesture 5
EMG (mV)
Time (sec)

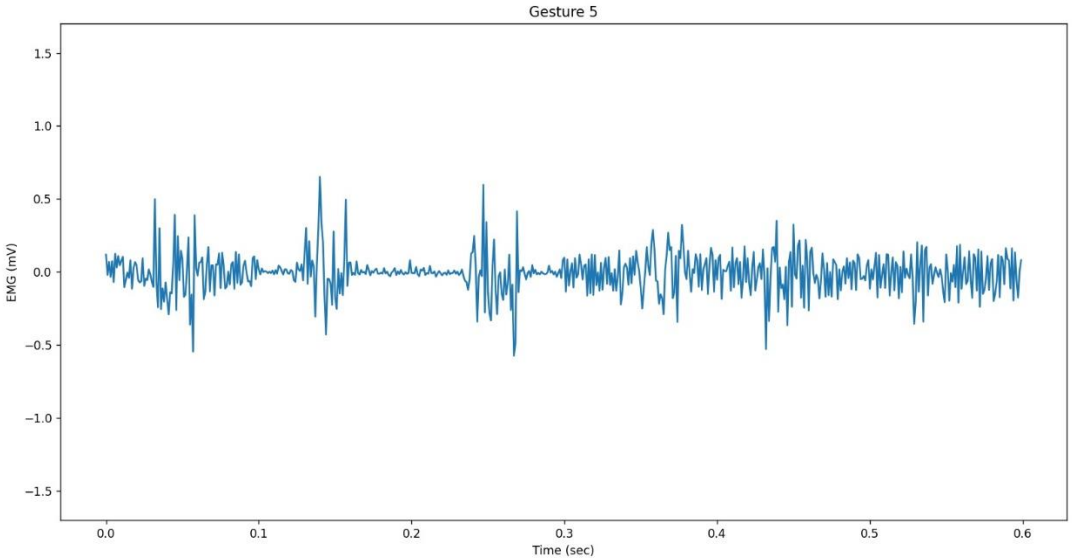
These are the images of a segment of the data of each gesture after having centred the signal to 0 on the Y axis.



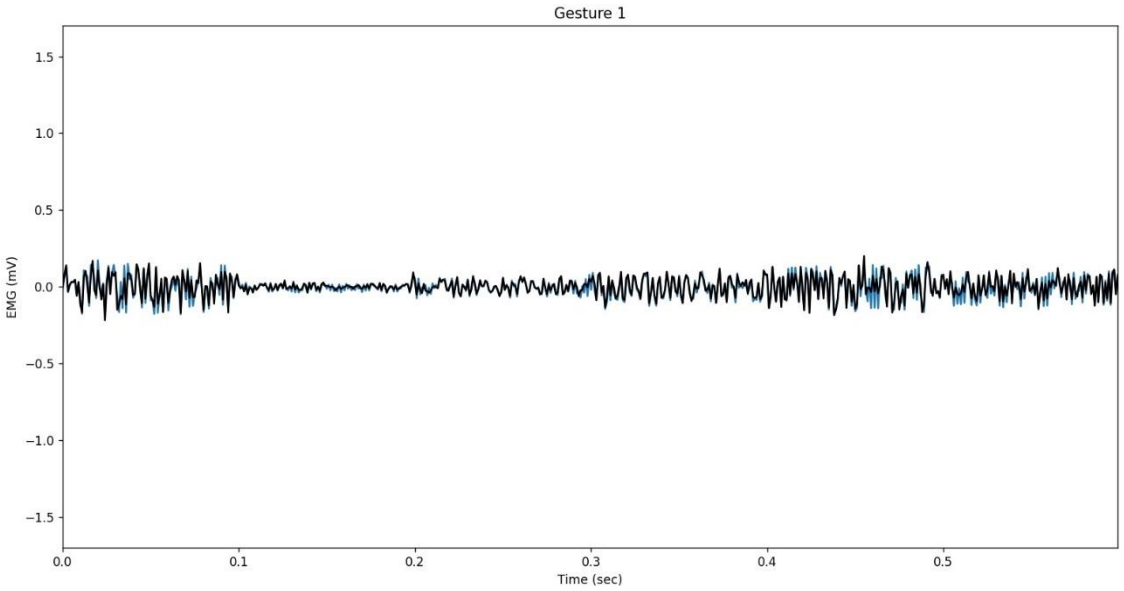


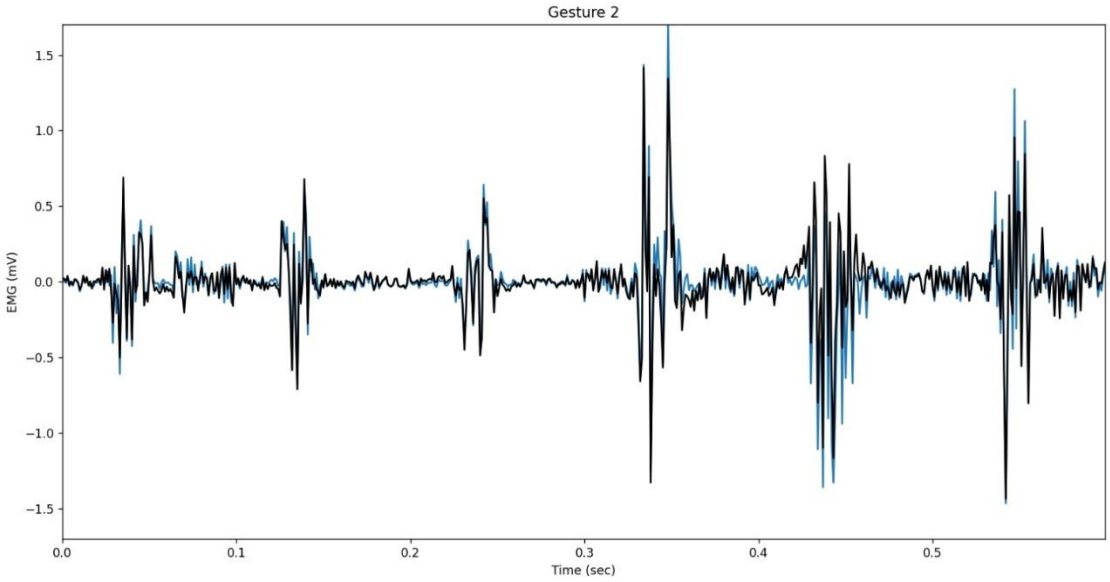

Gesture 3
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

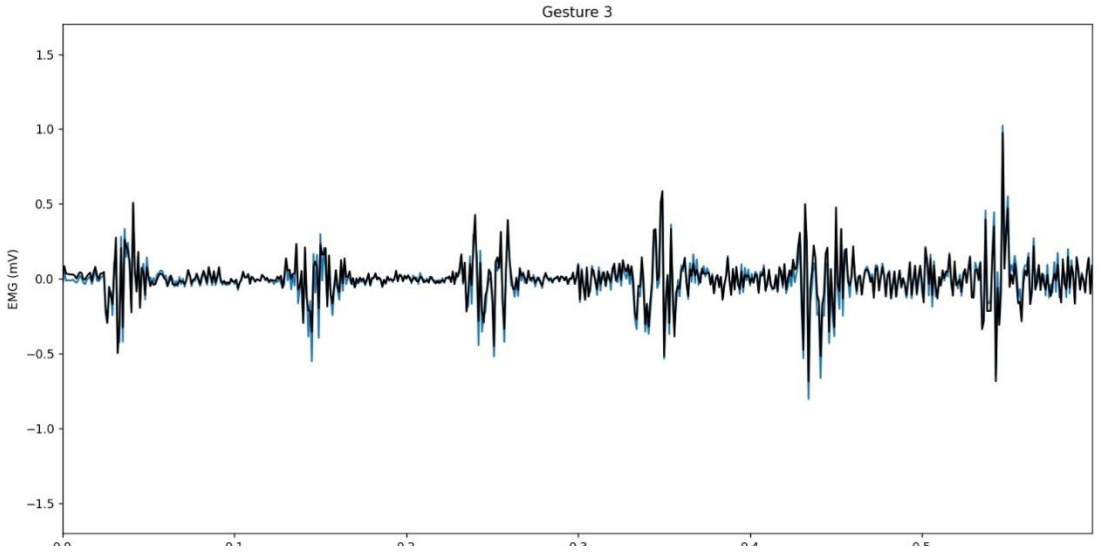

Gesture 4
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Time (sec)

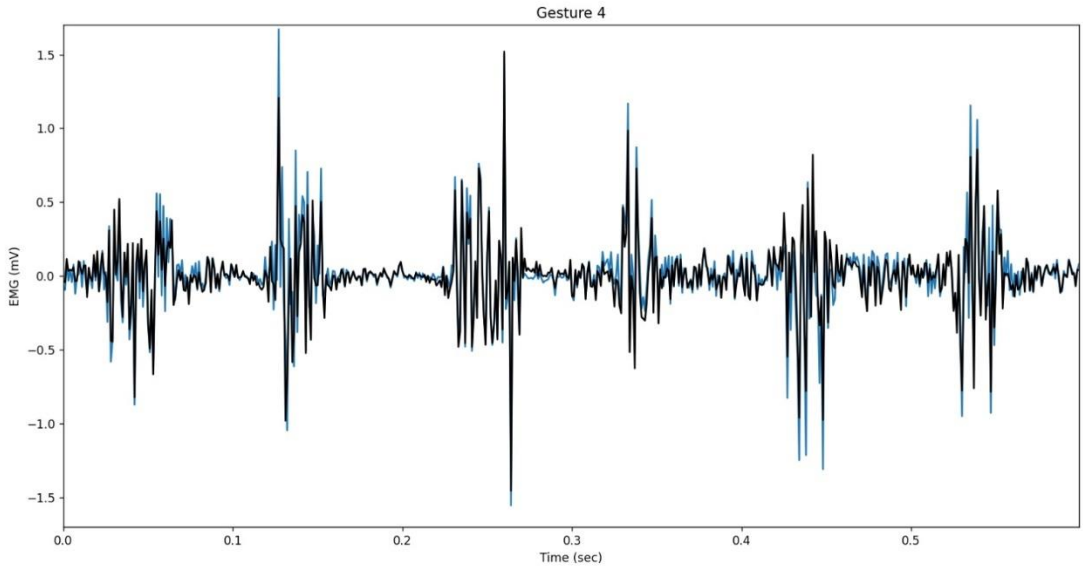

Gesture 5
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.1
0.2
0.3
0.4
0.5
0.6
Time (sec)

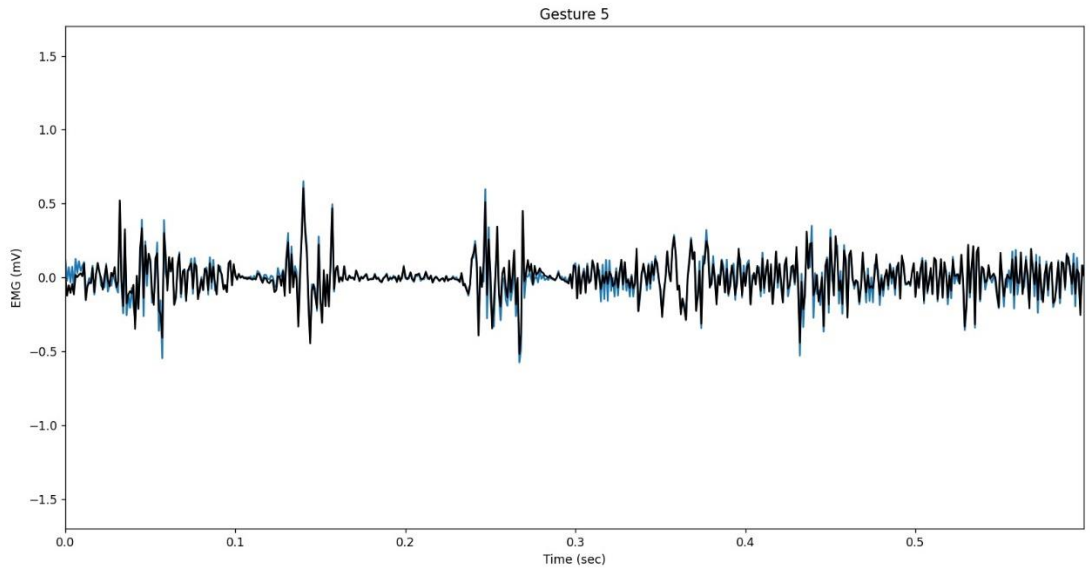
These are the images of a segment of the data of each gesture after having applied the bandpass filter.



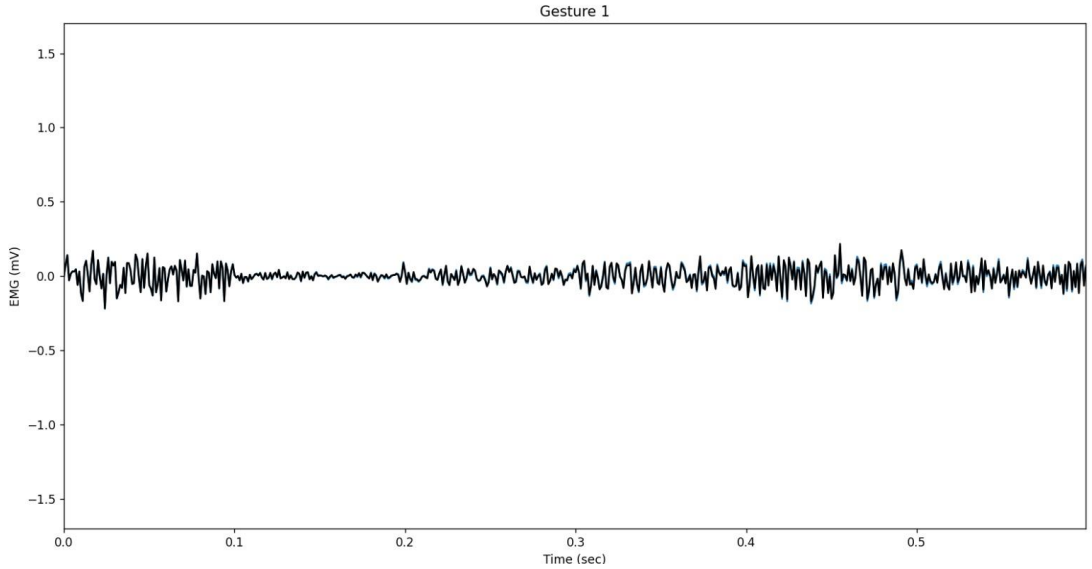


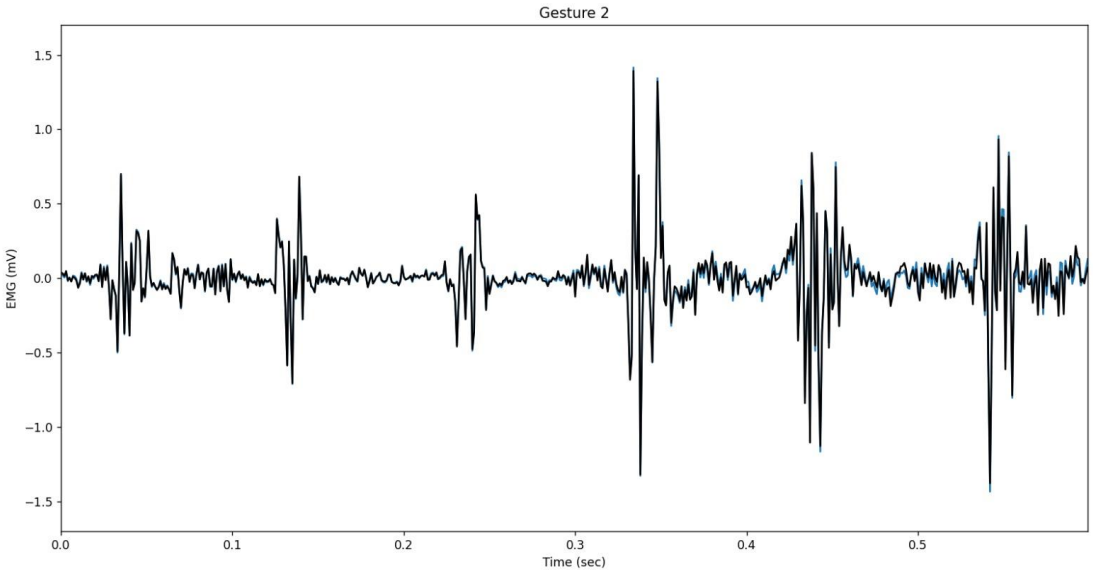

Gesture 3
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5

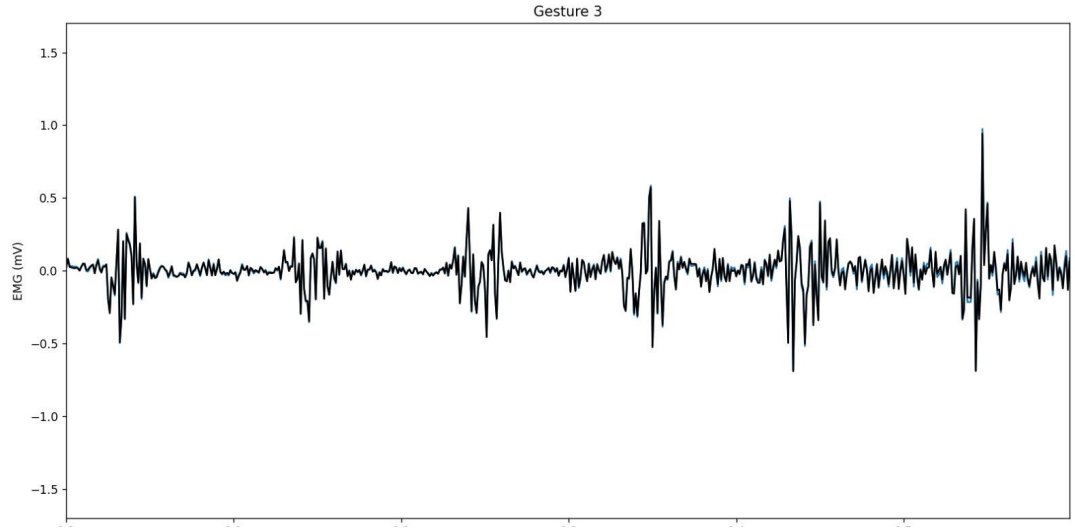

Gesture 4
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.1
0.2
0.3
0.4
0.5
Time (sec)

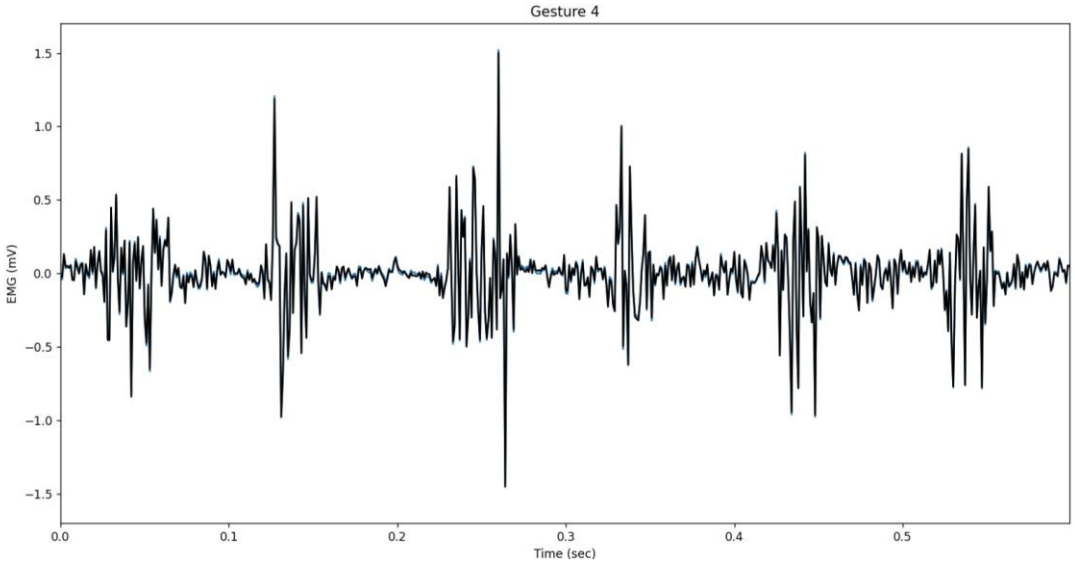

Gesture 5
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.1
0.2
0.3
0.4
0.5
Time (sec)

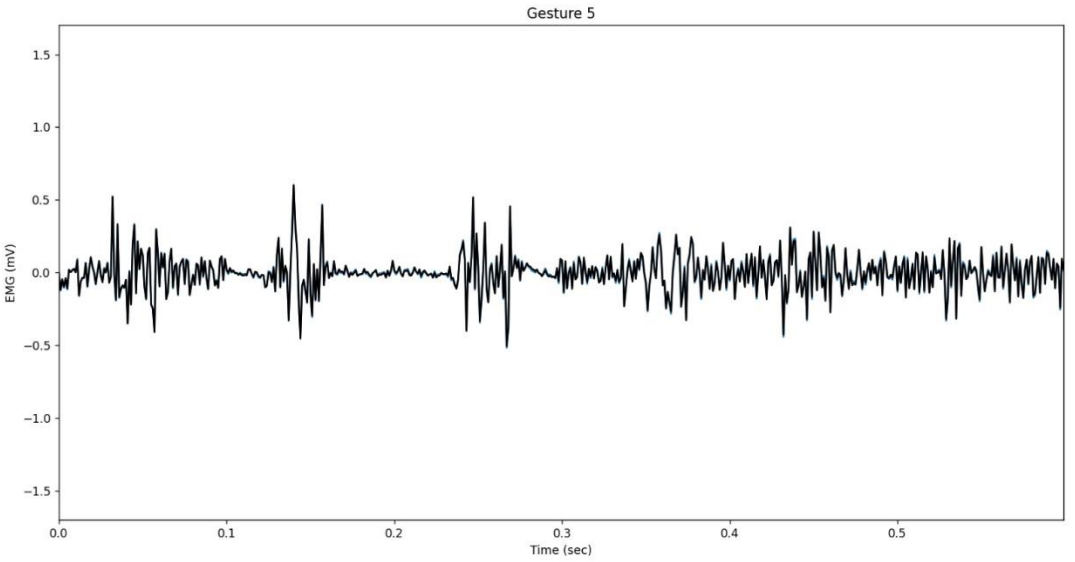
These are the images of a segment of the data of each gesture after having applied the notch filter.





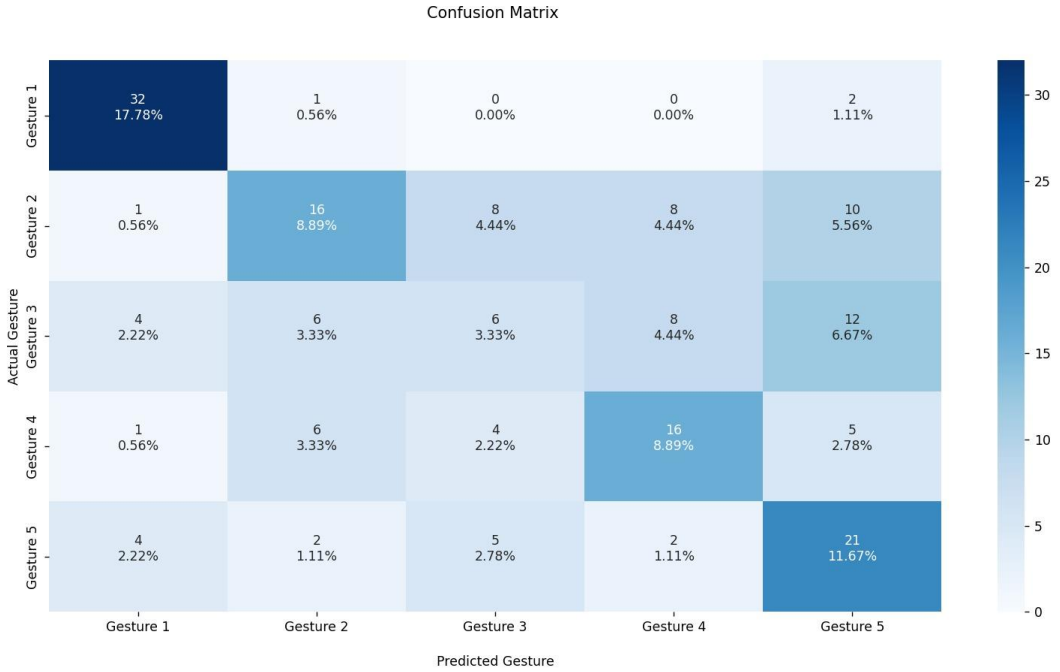

Gesture 3
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5


Gesture 4
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.1
0.2
0.3
0.4
0.5
Time (sec)


Gesture 5
EMG (mV)
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0.0
0.1
0.2
0.3
0.4
0.5
Time (sec)

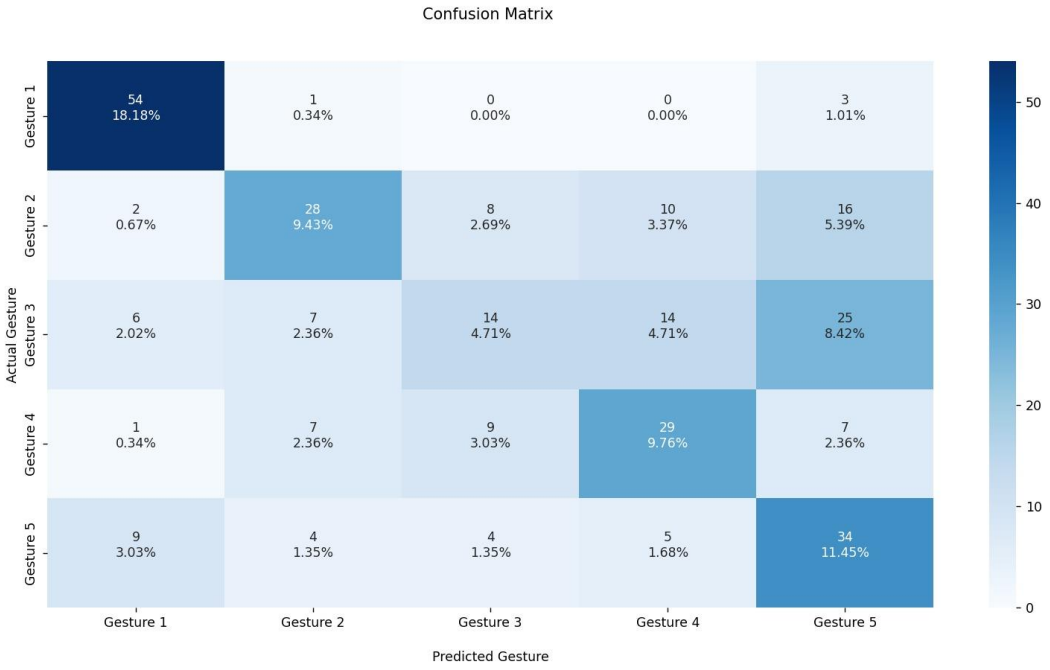
C10. Results

These are the results and confusion matrices for the Logistic Regression model with test size 20%:



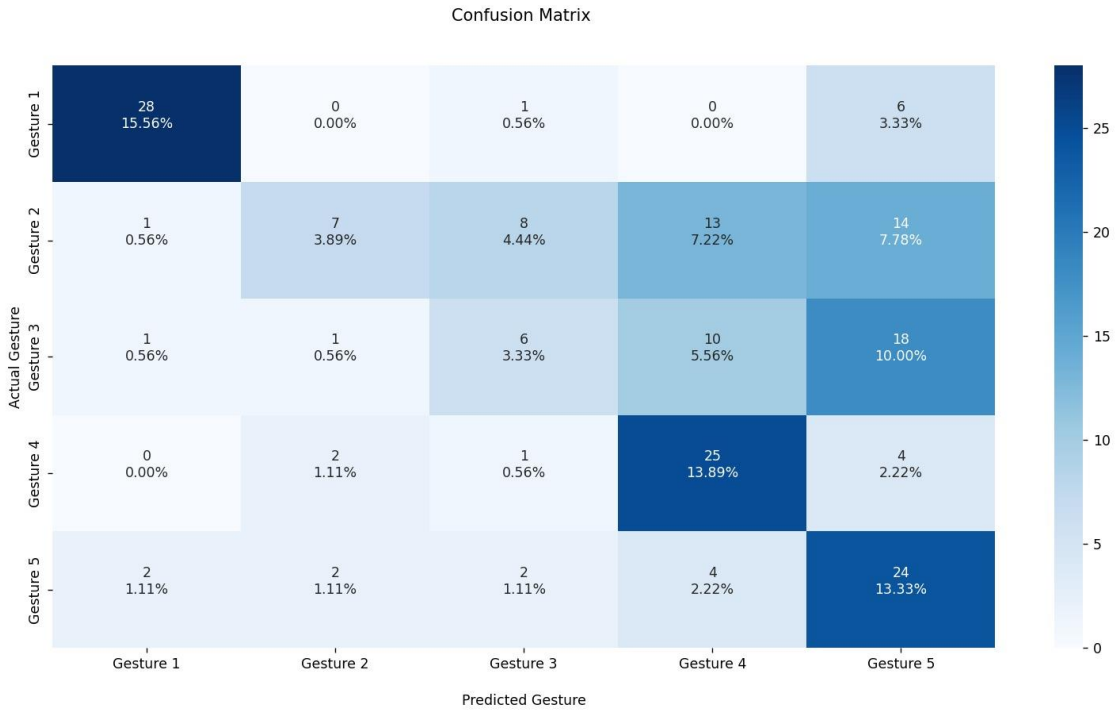

```
Model: Logistic Regression
[Train: 80%, Test: 20%]
TEST ACCURACY IS:    50.55555555555556 %
TRAINING ACCURACY IS:    54.456824512534816 %
RANDOMNESS EVALUATION IS:    19.444444444444446%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.76      0.91      0.83        35
           2       0.52      0.37      0.43        43
           3       0.26      0.17      0.20        36
           4       0.47      0.50      0.48        32
           5       0.42      0.62      0.50        34

    accuracy                           0.51       180
   macro avg       0.49      0.51      0.49       180
weighted avg       0.49      0.51      0.49       180
```

These are the results and confusion matrices for the Logistic Regression model with test size 33%:



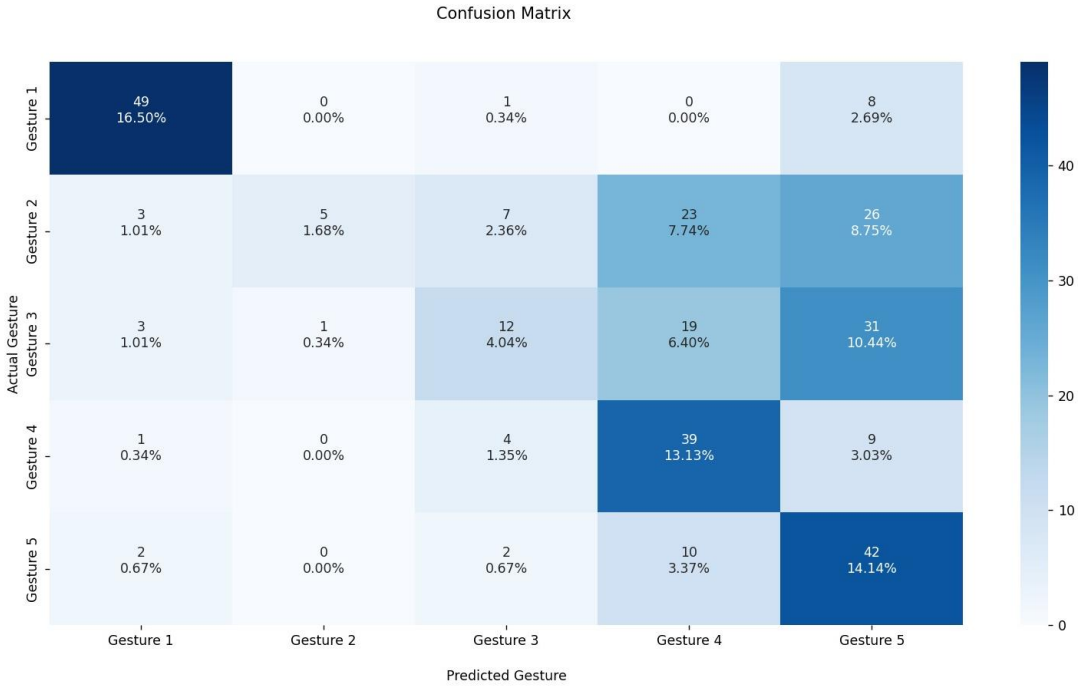

```
Model: Logistic Regression
Train: 67%, Test: 33% Of the data
TEST ACCURACY IS:        53.535353535353536 %
TRAINING ACCURACY IS:        53.07820299500832 %
RANDOMNESS EVALUATION IS     18.51851851851852%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.75      0.93      0.83        58
           2       0.60      0.44      0.50        64
           3       0.40      0.21      0.28        66
           4       0.50      0.55      0.52        53
           5       0.40      0.61      0.48        56

    accuracy                           0.54       297
   macro avg       0.53      0.55      0.52       297
weighted avg       0.53      0.54      0.52       297
```

These are the results and confusion matrices for the SVC model with test size 20%:



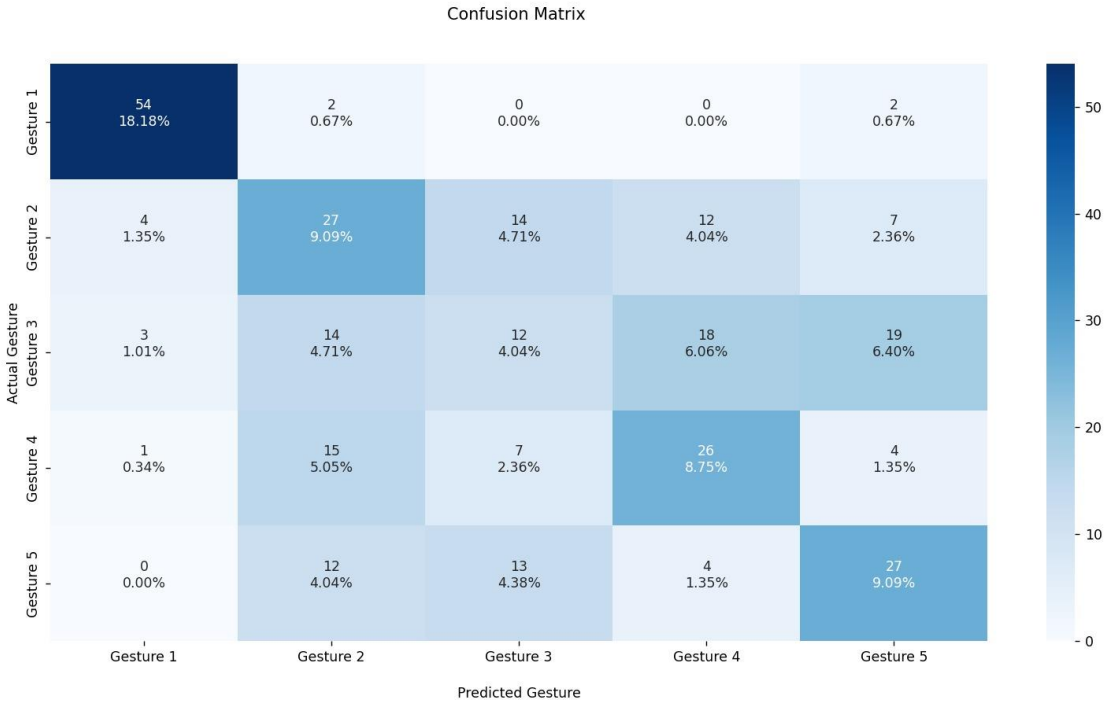

```
Model: Support Vector Classifier
[Train: 80%, Test: 20%]
TEST ACCURACY IS: 50.0 %
TRAINING ACCURACY IS:    54.1782729805014 %
RANDOMNESS EVALUATION IS:    18.88888888888889%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.88      0.80      0.84        35
           2       0.58      0.16      0.25        43
           3       0.33      0.17      0.22        36
           4       0.48      0.78      0.60        32
           5       0.36      0.71      0.48        34

    accuracy                           0.50       180
   macro avg       0.53      0.52      0.48       180
weighted avg       0.53      0.50      0.46       180
```

These are the results and confusion matrices for the SVC model with test size 33%:



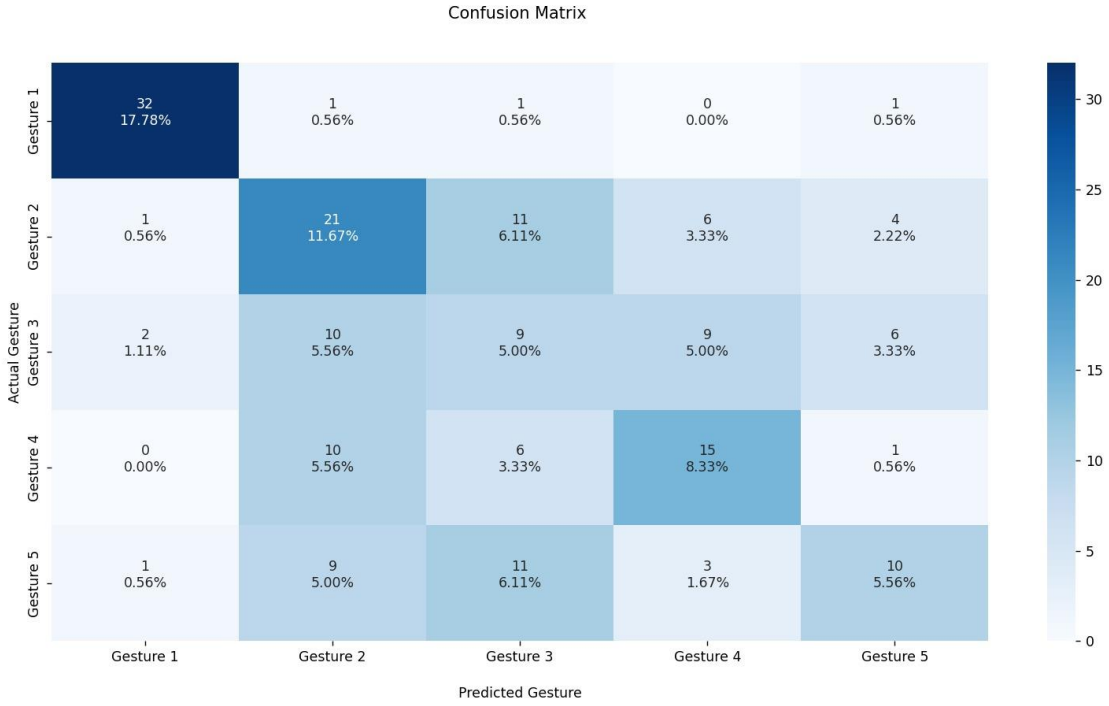

```
Model: Support Vector Classifier
Train: 67%, Test: 33% Of the data
TEST ACCURACY IS: 49.494949494949495 %
TRAINING ACCURACY IS:    53.244592346089846 %
RANDOMNESS EVALUATION IS:    21.21212121212121%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.84      0.84      0.84        58
           2       0.83      0.08      0.14        64
           3       0.46      0.18      0.26        66
           4       0.43      0.74      0.54        53
           5       0.36      0.75      0.49        56

    accuracy                           0.49       297
   macro avg       0.59      0.52      0.46       297
weighted avg       0.59      0.49      0.44       297
```

These are the results and confusion matrices for the KNN model with test size 33%:



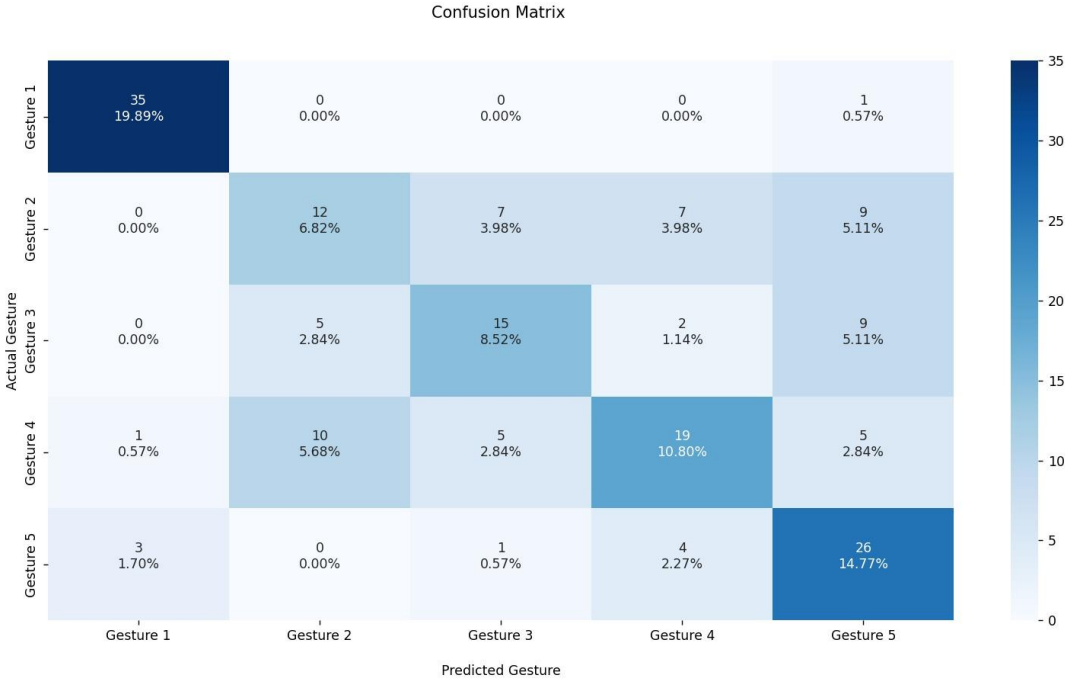

```
Model: K-Nearest Neighbours
[Train: 67%, Test: 33%]
TEST ACCURACY IS: 49.158249158249156 %
TRAINING ACCURACY IS:    65.3910149750416 %
RANDOMNESS EVALUATION IS:    17.845117845117844%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.87      0.93      0.90        58
           2       0.39      0.42      0.40        64
           3       0.26      0.18      0.21        66
           4       0.43      0.49      0.46        53
           5       0.46      0.48      0.47        56

    accuracy                           0.49       297
   macro avg       0.48      0.50      0.49       297
weighted avg       0.47      0.49      0.48       297
```

These are the results and confusion matrices for the KNN model with test size 20%:



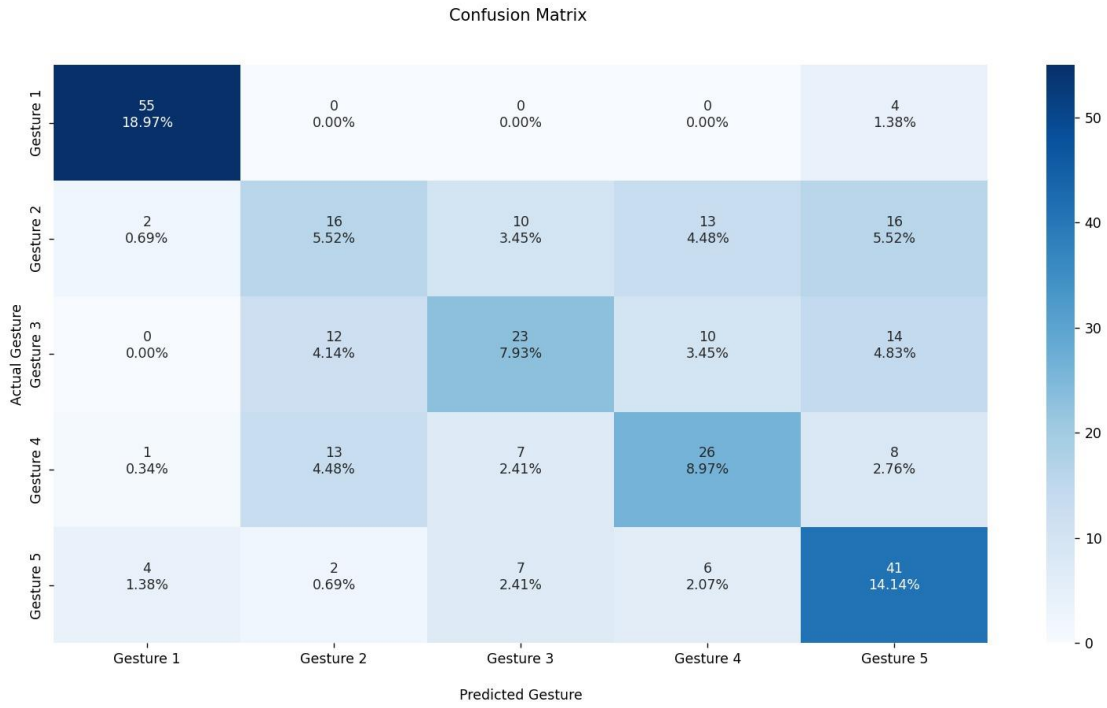

```
Model: K-Nearest Neighbours
[Train: 80%, Test: 20%]
TEST ACCURACY IS: 48.333333333333336 %
TRAINING ACCURACY IS:    66.29526462395543 %
RANDOMNESS EVALUATION IS:    21.666666666666668%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.89      0.91      0.90        35
           2       0.41      0.49      0.45        43
           3       0.24      0.25      0.24        36
           4       0.45      0.47      0.46        32
           5       0.45      0.29      0.36        34

    accuracy                           0.48       180
   macro avg       0.49      0.48      0.48       180
weighted avg       0.49      0.48      0.48       180
```

These are the results and confusion matrices for the Logistic Regression model with test size 20% with the implementation:



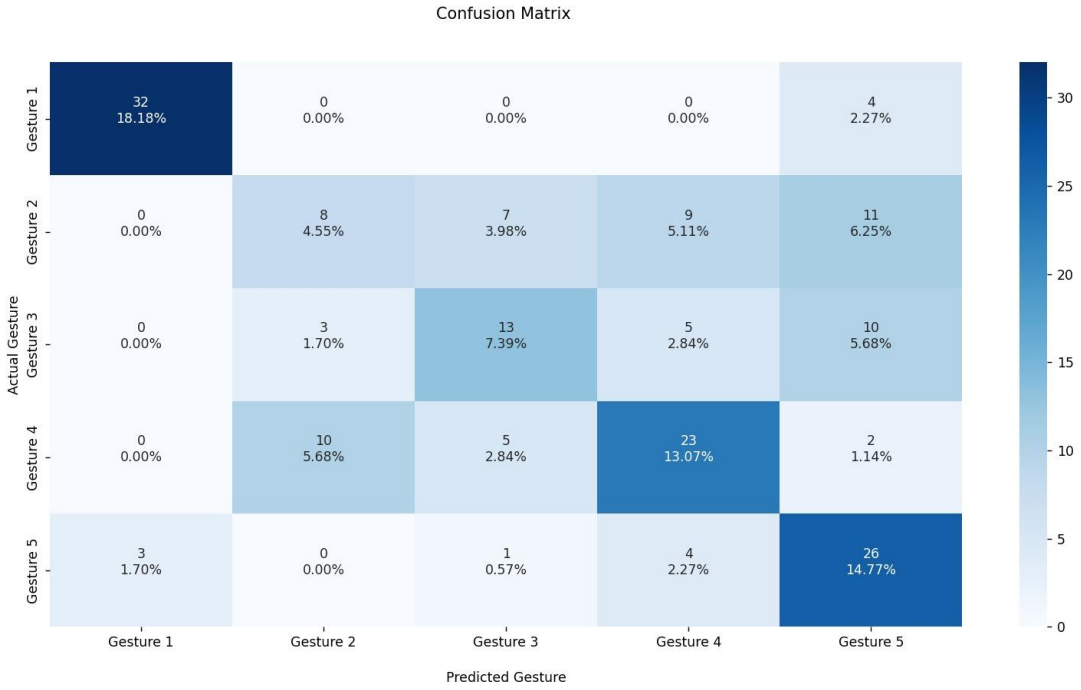

```
Model: Logistic Regression
[Train: 80%, Test: 20%]
TEST ACCURACY IS:    60.79545454545454 %
TRAINING ACCURACY IS:    56.12535612535613 %
RANDOMNESS EVALUATION IS:    15.909090909090908%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.90      0.97      0.93        36
           2       0.44      0.34      0.39        35
           3       0.54      0.48      0.51        31
           4       0.59      0.47      0.53        40
           5       0.52      0.76      0.62        34

    accuracy                           0.61       176
   macro avg       0.60      0.61      0.60       176
weighted avg       0.60      0.61      0.60       176
```

These are the results and confusion matrices for the Logistic Regression model with test size 33% with the implementation:



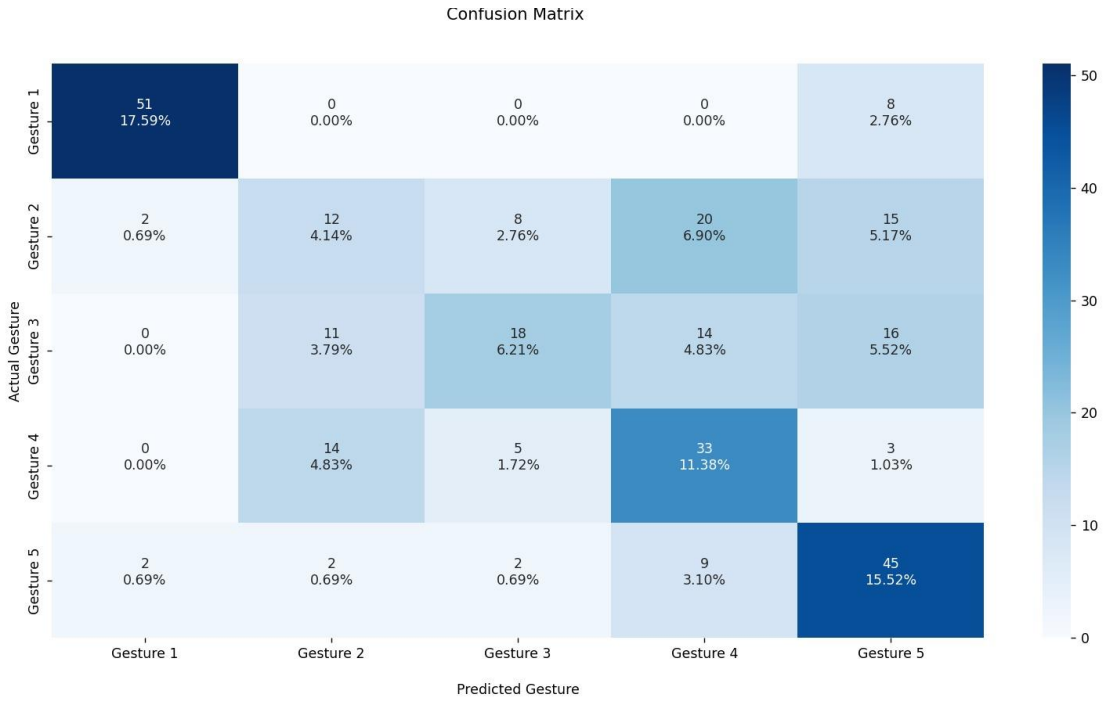

```
Model: Logistic Regression
[Train: 67%, Test: 33%]
TEST ACCURACY IS:    55.51724137931034 %
TRAINING ACCURACY IS:    56.12244897959183 %
RANDOMNESS EVALUATION IS:    22.413793103448278%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.89      0.93      0.91        59
           2       0.37      0.28      0.32        57
           3       0.49      0.39      0.43        59
           4       0.47      0.47      0.47        55
           5       0.49      0.68      0.57        60

    accuracy                           0.56       290
   macro avg       0.54      0.55      0.54       290
weighted avg       0.55      0.56      0.54       290
```

These are the results and confusion matrices for the SVC model with test size 20% with the implementation:



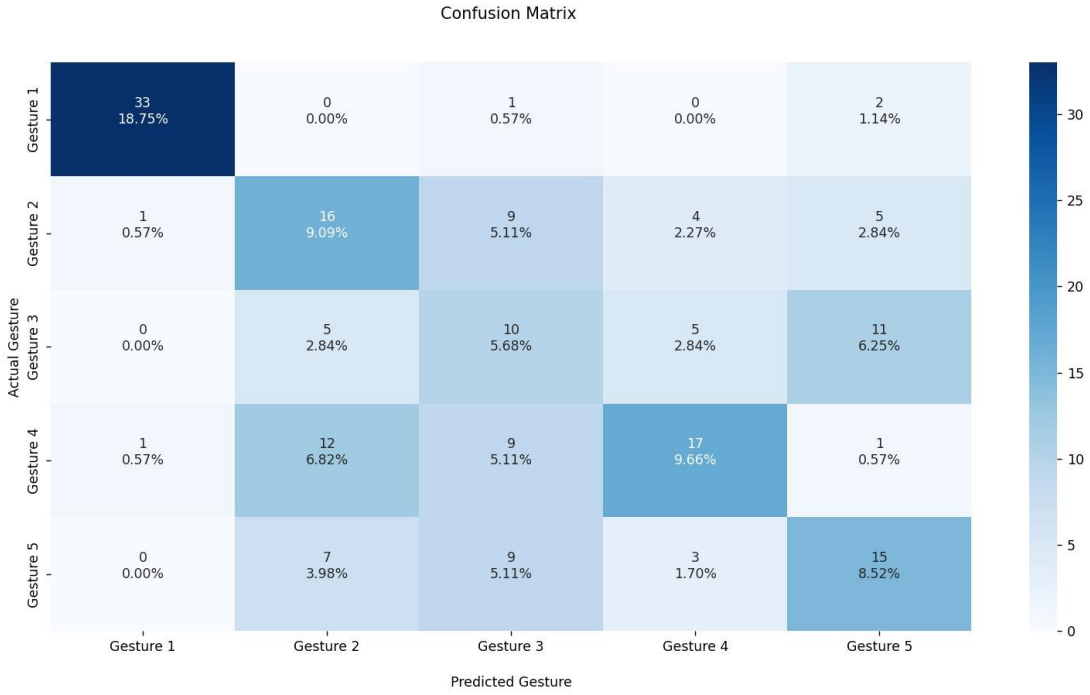

```
Model: Support Vector Classifier
[Train: 80%, Test: 20%]
TEST ACCURACY IS: 57.95454545454546 %
TRAINING ACCURACY IS:    62.10826210826211 %
RANDOMNESS EVALUATION IS:    19.886363636363637%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.91      0.89      0.90        36
           2       0.38      0.23      0.29        35
           3       0.50      0.42      0.46        31
           4       0.56      0.57      0.57        40
           5       0.49      0.76      0.60        34

    accuracy                           0.58       176
   macro avg       0.57      0.58      0.56       176
weighted avg       0.57      0.58      0.57       176
```

These are the results and confusion matrices for the SVC model with test size 33% with the implementation:



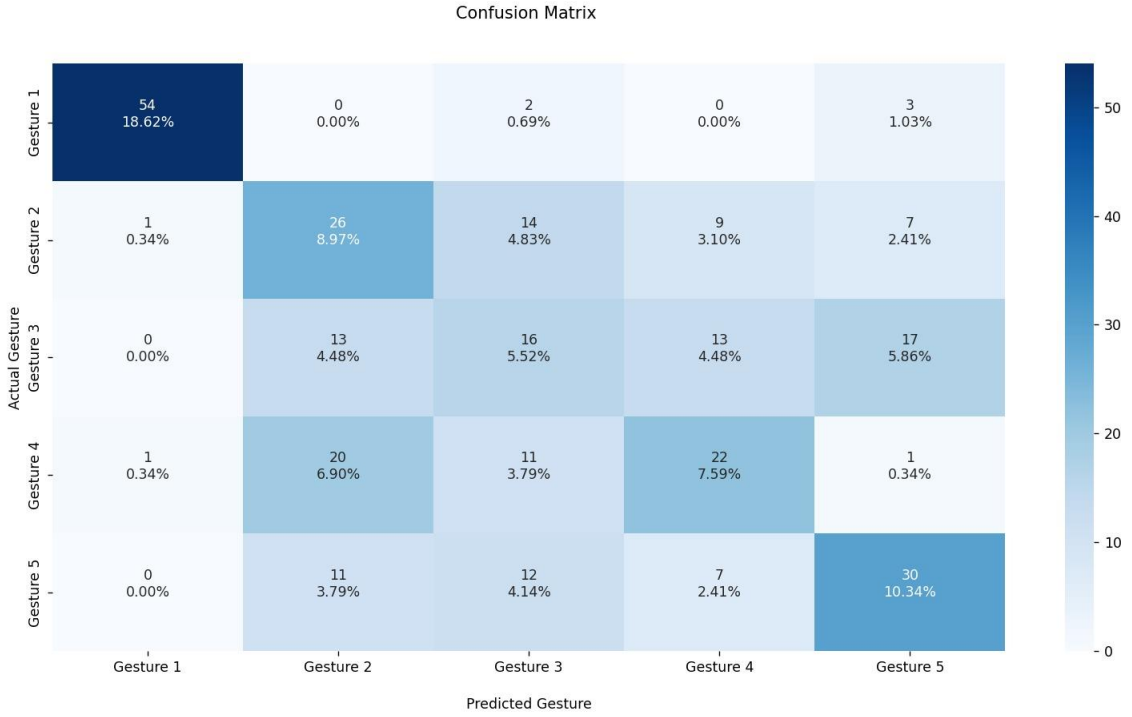

```
Model: Support Vector Classifier
[Train: 67%, Test: 33%]
TEST ACCURACY IS: 54.82758620689655 %
TRAINING ACCURACY IS:    61.564625850340136 %
RANDOMNESS EVALUATION IS:    19.655172413793103%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.93      0.86      0.89        59
           2       0.31      0.21      0.25        57
           3       0.55      0.31      0.39        59
           4       0.43      0.60      0.50        55
           5       0.52      0.75      0.61        60

    accuracy                           0.55       290
   macro avg       0.55      0.55      0.53       290
weighted avg       0.55      0.55      0.53       290
```

These are the results and confusion matrices for the KNN model with test size 20% with the implementation:




```
Model: K-Nearest Neighbours
[Train: 80%, Test: 20%]
TEST ACCURACY IS: 51.70454545454546 %
TRAINING ACCURACY IS:    64.38746438746439 %
RANDOMNESS EVALUATION IS:    22.15909090909091%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.94      0.92      0.93        36
           2       0.40      0.46      0.43        35
           3       0.26      0.32      0.29        31
           4       0.59      0.42      0.49        40
           5       0.44      0.44      0.44        34

    accuracy                           0.52       176
   macro avg       0.53      0.51      0.52       176
weighted avg       0.54      0.52      0.52       176
```

These are the results and confusion matrices for the KNN model with test size 33% with the implementation:




```
Model: K-Nearest Neighbours
[Train: 67%, Test: 33%]
TEST ACCURACY IS: 51.03448275862069 %
TRAINING ACCURACY IS:    62.755102040816325 %
RANDOMNESS EVALUATION IS:    15.517241379310345%
CLASSIFICATION REPORT:
              precision    recall  f1-score   support

           1       0.96      0.92      0.94        59
           2       0.37      0.46      0.41        57
           3       0.29      0.27      0.28        59
           4       0.43      0.40      0.42        55
           5       0.52      0.50      0.51        60

    accuracy                           0.51       290
   macro avg       0.52      0.51      0.51       290
weighted avg       0.52      0.51      0.51       290
```