TÉCNICO LISBOA

SignMeUp: A Wearable Sign Language Interpreter

Authors:

Guilherme Viegas guilherme.viegas@tecnico.ulisboa.pt

Abstract

In Portugal alone, around 120,000 people suffer from hearing loss and have a limited ability to communicate with the world surrounding them. To solve this problem, a wearable device is proposed, that can be easily weared on each hand of the deaf person. Using an MPU sensor and a *Xiao ESP32C3* microcontroller to capture the hand's motion, each device sends signals to a main machine that interprets each individual signal using a Decision Tree model, put together a coherent sentence through *OpenAI's gpt-3.5* model capabilities, and say it out loud. Overall, the proposed device is able detect when a new phrase is being gesticulated, segment between different signs, classify each of the signs into the respective word and produce a coherent speech understandable by the common listener.

1 Introduction

The ability to communicate defines us as human beings and as a society. It forms a foundation for decision making and relationship building. However, for millions of deaf and hard-of-hearing individuals, this is still a daily struggle. Hearing loss can have a profound impact on a person's quality of life, including their mental health. The most recent WHO estimate suggests that around 6.1 % of the world's population were living with disabling hearing loss in 2018, estimated to rise over 900 million by 2050. In Portugal it is estimated that about 120,000 people have some degree of hearing loss and about 30,000 are native speakers of Portuguese Sign Language (LGP). This community often resorts to impractical methods of communicating with society, such as the use of an interpreter that can become costly.

The aim of this project is to overcome such communication restrictions by creating a wearable device, which automatically translates LGP into spoken language.

When performing a gesture in sign language five parameters should be considered: configuration, position, movement and orientation of the hands, with the addition of a non-manual component of facial expressions and posture, which are out of this work's scope. These signals, obtained with an MPU, will be sent from a microcontroller to a central machine via *Wi-Fi*. Moreover, just as any other language, with a natural development, deeply ingrained with the culture where it originated, LGP has it's own grammar, including a different word order and verbal conjugation. To make it sound like coherent speech to a hearing subject, the phrases need to be reorganized, conjugated and punctuated. For this aim a *ChatGPT*-like Natural Language Processing mechanism is proposed. Two similar devices are used, one for each hand, sending data to a main machine that will interpret each individual signal, put together a coherent sentence, and say it out loud.

In Chapter 2 an overview of the current and most used approaches for the above challenge are presented. The following three chapters describe both the approach taken and the respective results. Chapter 3 describes how the data is acquired and processed from it's raw form. Chapter 4 dives into how such raw gesture data is classified into single words, with an overview of the machine learning approach taken and the results obtained. Chapter 5 demonstrates how these single and semantically incorrect words are converted into a fluid speech. Chapter 6 concludes this work also discussing possibilities for future work.

With its prototype nature, a parallel goal of the present project is to get acquainted, at an high-level, with such important topics of engineering that are data processing, classification models and micro-controller platforms.

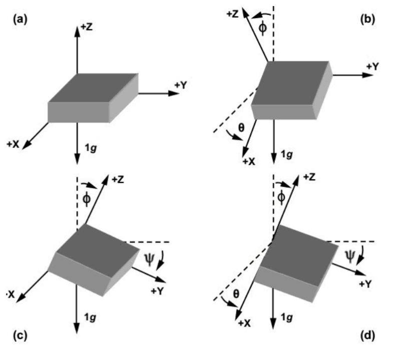
2 Related Work

The proposed work of developing a wearable device capable of translating hand gestures into spoken language can be divided into several challenges. Most of the work performed within this topic falls into a pipeline divided between Data Acquisition, Pre-Processing, Segment Detection and Classification. Each one of them can be tackled from different perspectives and using a multitude of approaches. Using sensors such as electromyography (EMG), piezoresistive effect, etc or using imaging devices and vision processing, both have been widely explored for sign language translation [13], [14]. Nevertheless, these techniques are limited by their structural complexity and unsuitability for long-term use. In what regards vision-based tools, the work developed by Elsayed et. al., [5] has shown a 99 % accuracy in labeling the American Sign Language letters of the alphabet using a Convolutional Neural Network (CNN) as a classification model. Although using solely pictures, this has then been extrapolated into video formats as successfully shown by Trigueiros et. al., [16] that used a *kinect* camera and a Support Vector Machine (SVM) model.

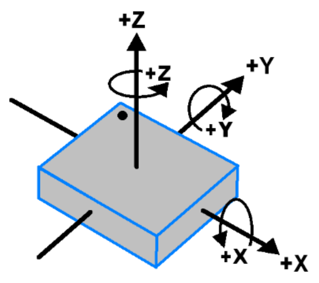
Nevertheless, this is not practical to the day-to-day of a deaf individual that wants to be heard. In Zhou et al [17] a wearable system was developed to achieve real-time and accurate translations of American Sign Language (ASL) gestures into audible speech. For this, not only yarn-based stretchable sensor arrays (YSSAs) were attached to a pair of gloves, but also adhesive sensors were used to capture facial expressions. Another approach emerged from Lee, K.H., et al. [11], using EMG sensors and an Artificial Neural Network (ANN) to recognize patterns in the EMG signals of specific gestures, enabling their classification. Other approaches have shown promising results when merging more than one type of sensor. This was proposed by Gu, Yutong et. al., [8] which used both Inertial Measurement Units (IMUs) for the sign detection and EMG sensors to capture facial expressions. First a CNN is applied to extract features from the input data. Later, a Long Short-term memory (LSTM) and Transformer models are explored to achieve end-to-end translation from input signals to text sentences. This project aims to explore the use of light and less complex methods of classification such as SVMs and Decision Trees, while trying to generate a more acceptable and fluid sentence using *OpenAI*'s capabilities.

3 Data Acquisition

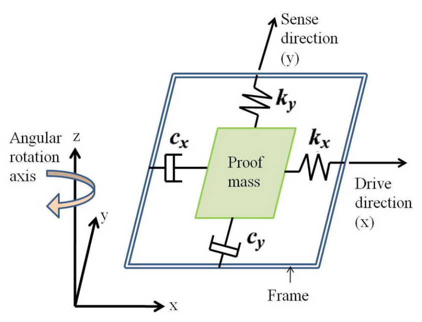
Any engineering project relies on data to be insightful. This means understanding what type of data will be used is the first step to solving the proposed problem. An MPU-6050 sensor [9], as shown in the right hand side of Figure 2b, per hand is used to convert physical properties into digital signals. This is a motion tracking device that combines a 3-axis Gyroscope, a 3-axis Accelerometer and a Digital Motion Processor all in a small package. The communication between MPU and microcontroller is achieved through the I2C protocol, which enables a synchronous and fast communication, at relatively short distances. Only two I/O pins, are required in I2C to exchange data. These are named SCL (Serial Clock) and SDA (Serial Data), with the first defining the rate at which communication is performed and the second one the dedicated line to exchange data. The built-in 3-axis Accelerometer is capable of detecting angle of tilt along the X, Y and Z axis, as shown in Figure 1a. This works through Micro Electro Mechanical (MEMs) technology, where an acceleration along any of the given axis causes a deflection on the movable mass inside the chip. This unbalance, created inside a capacitor, leads to an output voltage that is proportional to the acceleration. Using a similar MEMs technology, the gyroscope is used to detect angular velocity along the X, Y, Z axes as shown in below Figure 1b. It contains a set of proof mass, kept in a continuous oscillating movement. When an angular motion is applied, the Coriolis Effect [2] causes a change in capacitance between the masses depending on the axis of the angular movement, as in Figure 1c. This change in capacitance is sensed, amplified and filtered to produce a voltage proportional to the angular velocity.



(a) MPU-6050 Axis of accelerometer's inclination



(b) MPU-6050 Orientation and Polarity of Rotation

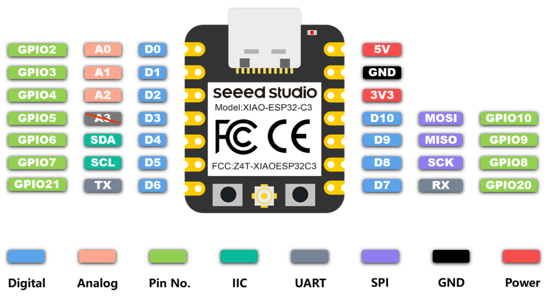


(c) Inner working of a gyroscope

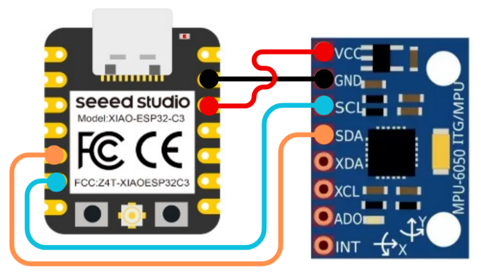
Figure 1: Overview of MPU-6050 and its inner workings

3.1 ESP32C3 Microcontroller

As the MPU's single capability is to retrieve the physical properties of acceleration and angular velocity into digital signals, a more powerful micro-controller is used to send the signals to a server (own computer) so that it can be segmented, classified and in the end, result in a fluid speech that anyone can understand. A *XIAO SeedUino ESP32C3* controller [6], shown in Figure 2, is used for this. The ESP32C3 controller is a development board with both WiFi and Bluetooth connectivity. It supports the three mainly used serial communication interfaces such as UART, I2C and SPI and comes included with an external antenna to increase the signal strength. Being such an high-performance, low power and cost-effective development board, it provides the perfect fit for achieving this project goals. The data is sent via I2C communication from the MPU sensor to the ESP32C3 board in it's raw form. The connection between MPU and ESP32C is as simple as attaching the 2 I2C cables, plus connecting the required power (*VCC* and *GND*) connections, as shown in Figure 2b.



(a) Xiao SeeedUino Pinout Diagram



(b) Connections: ESP32C and MPU6500

Figure 2: Xiao SeedUino ESP32C3 - Component Overview

3.2 WiFi - UDP Communication

As powerful it can be, the ESP32C3 still lacks the power of running complex program scripts to process the data. As a proof-of-concept, a personal computer is used running a *python* program to achieve so. To keep the wearable device as lightweight as possible, WiFi communication is used to receive the data from the ESP32C3 to the computer. For this, the User Datagram Protocol (UDP) [10] was used, enabling a fast communication between the two nodes. This is preferred over the Transmission Control Protocol (TCP) for time-sensitive applications as any lost packet of data is preferable over the latency that may exists in TCP. Being the scope of the project to receive and process data in real-time as the users makes it's gestures, in the end the simplicity of UDP outweighs the reliability of TCP.

To have a complete and working UDP connection between two nodes (i.e., ESP32C3 and Computer) both nodes have to have a shared port and know the other node's IP. A first scrip, as found in Appendix A, is run to retrieve the IP from the ESP32C3 that is connected to a local network. This IP is then prompted to the python code in the computer so that it can start the communication between the two. The ports, namely *4210* and *4211* for the left and right hands, respectively, are pre-defined in each of the nodes software. The computer, given the ESP32C3 IP and port, is able to send him a direct message with an *Hello World* message, which the ESP32C3 himself reads and retrieves the Computer's IP address. Once this exchange is complete, both nodes are ready to exchange messages from each other at any given time.

3.3 Outer Shell

For easy use and adaptation to the person's hands, a 3D printable box was designed, which can be held on by *velcro* and elastic bands. This box was designed with *TinkerCAD* and is a lightweight and cheap option with easy access to the sensors inside, through a slide-on lid.

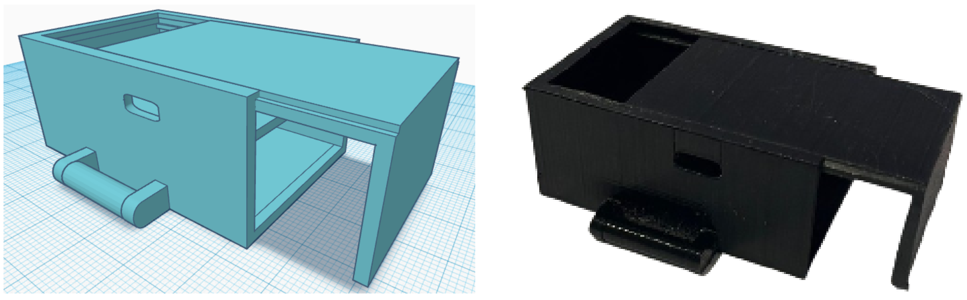

(a) CAD model of the 3D printable box. (b) 3D printed box.

Figure 3: CAD model and printed version of the container.

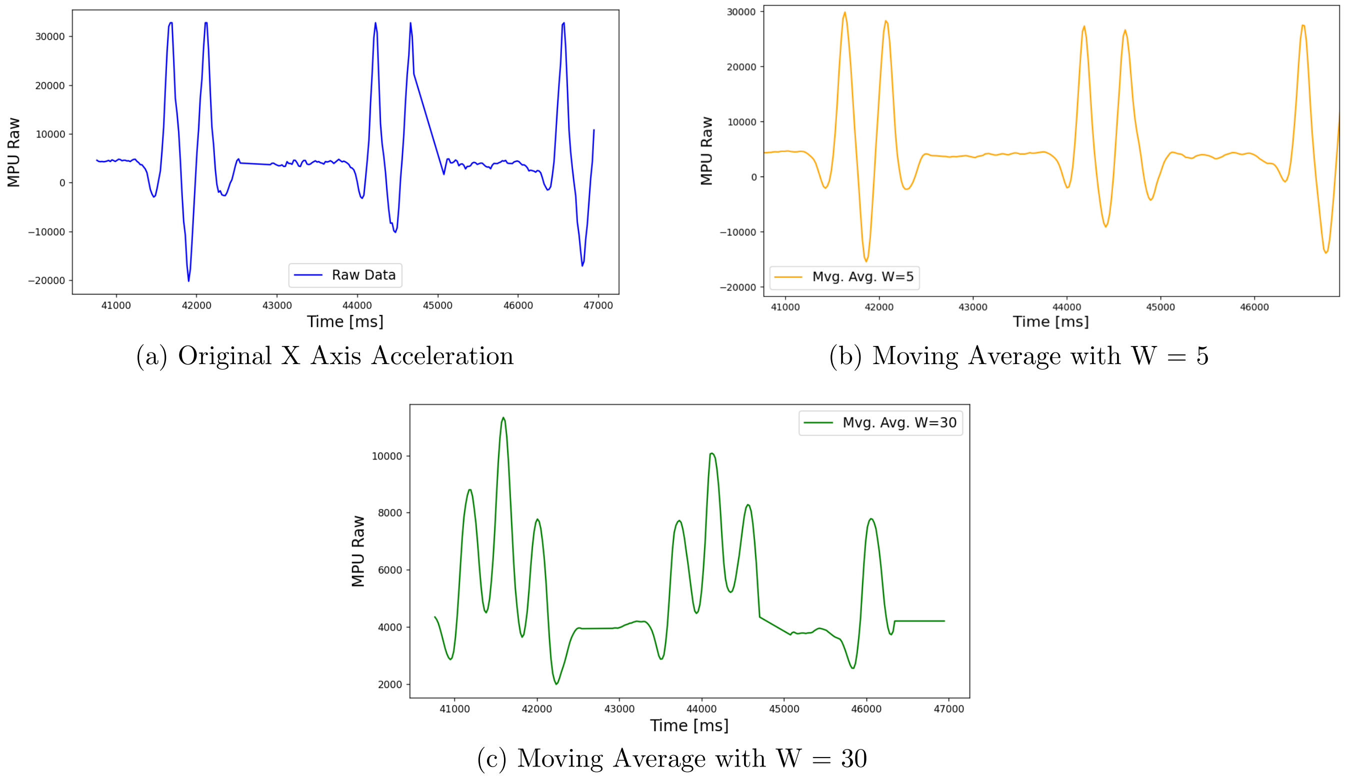
4 Data Processing and Segment Detection

During communication multiple gestures are performed to produce a meaningful phrase, but this comes from the ESP32C3 as only raw data resulting from the MPU sensor. This raw data has to be, first, cleaned and preprocessed so that any outliers or meaningless data samples are filtered out. Secondly, the data string is segmented to have sequences of data corresponding to different signs.

4.1 Data Cleaning

Working in the real world scenario, data always comes with noise or outliers. The characteristic noise of this type of signal is high in frequency and it is derived not only from the normal functioning of electronic components and connections, but also from imprecise movements and trembles of human nature.

High frequency noise can be greatly reduced through the application of a low-pass filter. In this case, a moving average was applied on the raw data, with a predefined window so that data can me smoothed out without loosing its key features. In Figure 4 a sequence of three *"Olá"* signs is performed, showing the impact the moving average algorithm can have on the original data. On Figure 4 (a) the original raw data is presented, whereas Figures 4 (b) and 4 (c) show the result of applying a Moving Average with a window size, $W = 5$ and $W = 30$, respectively. The bigger the window, the smoother the data, but this comes with the price of loosing what can be key features from the data. The right window size comes not only of multiple empirical tests with the used data, but also through evaluating the number of samples which will usually be at any given sign. Using a relatively small window size, $W = 5$, is able to smooth out the high frequency noise, while maintaining the key features of the signal. On the other hand, a sign usually consists of 75 to 125 samples, which means using a bigger window size, $W = 30$, ranges between 20 % to 40 % of its entirety, leading to notorious changes on the original signal.



(a) Original X Axis Acceleration

(b) Moving Average with W = 5

(c) Moving Average with W = 30

Figure 4: Differences on Data after applying a Moving Average

4.2 Segment Detection

As the classification step will only be performed over single words, the complete sequences of data have to be segmented into distinct hand gestures. Several approaches can be taken to tackle this challenge, differing on their complexity. Change Point Detection explores the statistical properties of the signal, from detecting shifts on the mean of the signal, to applying online Bayesian inference to detect such change points [1]. Other Segmentation algorithms can use a sliding window approach, dividing the series into fixed-size windows and applying change detection within each window. More recently, Machine Learning has been used, where segment detection is based on learnt patterns, namely the pattern of starting and/or stopping an hand gesture. Neural Networks

(CNNs and RNNs) are currently the state of the art approach to this challenge, showing great results albeit its need for real data to train the model.

To evaluate such change points, one useful metric used is the Root Mean Square (RMS) that, in practice, evaluates the *magnitude of change* in a signal. As the signals used here do not have zero mean, some adjustments are made, as the RMS would include both the mean and the varying part of the signal. This would require to know the entire signal before hand in order to compute the mean. As the goal of the present work is to classify gestures in a real-time manner, the approach used is to differentiate the original signal $x(t)$ so that $x^{'}(t)$ fluctuates around zero. Then, the RMS value of $x^{'}(t)$ can be used to determine whether the signal is active [12]. This is expressed through:

$$RMS(x) = \sqrt{\frac{1}{N-1}\sum_{i=0}^{N-2}(x(i+1)-x(i))^2} \tag{1}$$

The above RMS calculation is calculated over the entire signal, one per each channel, at each 0.1 s and using the last 5 samples. This choice is also taking in mind that the sampling frequency was defined in the ESP32C3 as 50 Hz, which is more than enough to provide a good representation of the signal. Summing all channels together, lead us to a transformed signal as depicted in Figure 5, for when three *"Olá"* signs were performed.

One assumption that is made, to provide a more robust way of identifying the change points in activity, is to define a minimum time (i.e., number of samples) per gesture. This was found important specifically for gestures with a small stop of the motion (e.g., "*menu*"). Considering a sampling frequency of 50 Hz and the empirically obtained value of $\approx$ 0.3 s for the minimum duration of an hand gesture, it is obtained the number of samples $N_a = 0.3 \times 50 = 15$. These values in conjunction with an analysis on the growth and decline points of the *Summed RMS* signal lead to a successful segmentation of the signs. The three *Olá* gestures are correctly marked with the blue areas shown in Figure 5. Another experimental test is performed, testing the segment detection algorithm with 3 different words {"*Olá*", "*Café*", "*Quero*"}, which were gestured 50 times, lead to a correct identification of $\approx$ 98 % of the active segments, as shown in Table 1.

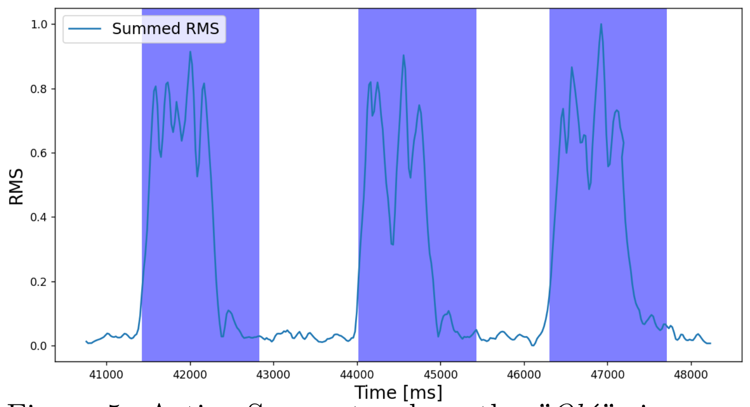


Figure 5: Active Segments where the *"Olá"* signs are performed.

Table 1: Accuracy results for the Summed RMS approach - 50 samples per word.

Word	# of Signs	Accuracy
"Olá"	53	94 %
"Café"	51	98 %
"Quero"	49	98 %
Average	51	98 %

5 Classification Problem

The next step on this project's pipeline is to successfully classify the segmented data into meaningful signs. As described in Section 2, multiple approaches could be taken here, from more simple ones of using predefined thresholds, to training deep neural networks with multiple layers. Such choice depends mostly on the type of data being fed into the model that will classify it and the required accuracy versus speed of the classification approach.

For this case, the data can be defined as a tabular data with a total of 12 columns, 6 columns per MPU and each per sensor channel (i.e., linear acceleration on X, Y and Z axis and angular velocity on X, Y and Z axis), and N number of lines, one for each sample taken. Given its sequential structure and the requirement for a fast classification, a Decision Tree model has shown to be a fairly acceptable approach to this challenge [3], [4]. As shown by Grinsztajn et al., 2022 [7], this is mainly because, while Neural Networks struggle to create the best-fit functions for non-smooth functions, Random Forests do much better with jagged and irregular patterns, identifying much accurate decision boundaries, proved by Figure 6.

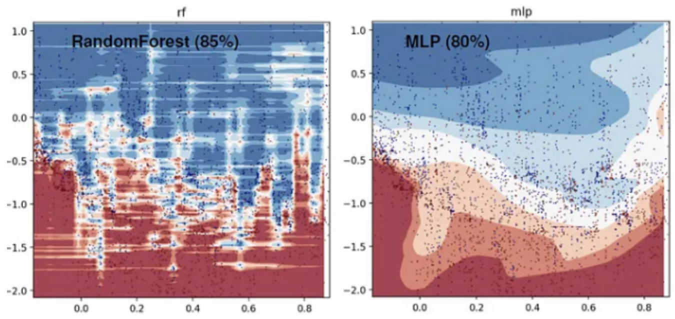


Figure 6: Decision boundaries of a RandomForest and MLP for the *electricity* dataset of *OpenML* [7]

Decision Trees follow a flowchart-like structure, with each node representing a decision done on the attributes, each branch representing the outcome of these decisions and leaf nodes being the final outcomes. At each step, the best feature is selected to split the data into different branches, according to a chosen metric. Two of the most common ones are:

- **Gini Impurity** which measures the probability of incorrectly classifying a randomly chosen element in the dataset, if randomly labeled as one of the classes in the dataset. It is calculated as $1 - \sum_{i=1}^{N}(p_i)^2$, with p_i the probability of an element being classified into one specific class.
- **Entropy** which measures the amount of uncertainty or impurity in the dataset. This can be calculated through $-\sum_{i=1}^{N}(p_i)\log_2(p_i)$, with p_i the same as in Gini Impurity.

To confirm and evaluate the proposed approach fifty samples (i.e., gestures) per word are collected for three different words, {"*Olá*", "*Café*", "*Quero*"}. These are collected and stored, in their *raw* form, as they are received from the ESP32C3 device. From there, the data is pre-processed and segmented as depicted in Section 4.2, and then separated into train and test datasets, allocating 80 % of the total data to training and the remaining 20 % to testing. For each of the samples, the data is then normalized, which makes each channel (i.e., feature) hold a similar importance for the classifier model. The *Min-Max* algorithm, according to the ratio $x_n = \frac{x_i - Min}{Max - Min}$, is a really fast-forward implementation that rescales data between 0 and 1, but comes with the risk of reducing variance and magnifying the effect of outliers. Another approach, more robust to expressive outliers, would be to substract the mean and divide by the variance, according to: $x_n = (x_i - \mu_x)/\sigma_x$, with μ_x and σ_x, resulting in a signal with null mean (μ_x) and unitary variance (σ_x^2). One important step to take in mind is that each sample is required to have the same size, meaning, the same number of MPU readings taken. Here, the maximum number of readings (i.e., rows) is computed across all **train** samples for the three words, and then, for each set, a first order interpolation is applied. Once this step is performed, 4 different models are trained, feeding them both the training set and a list with the respective labels. Comparing between a Support Vector Machine, Decision Trees with both *Gini* and *Entropy* criterion and a Multi-layer Perceptron, the Decision Tree using *Entropy* criterion yielded the best results, as presented in Table 2. Although a Decision Tree, with *Entropy* criterion, and Multi-Layer Perceptron models yielded similar results, the first has not only the advantage of a faster training and prediction, but also makes use of the sequential nature of the data and as is more explanatory as it's model can be visually represented.

Table 2: Experiment results for Different Classifier Models for labels {"*Olá*", "*Café*", "*Quero*"}.

Classifier Model	Accuracy	Precision	Recall	F1 Score	Log Loss
Support Vector Machine	0.75	[0.57, 1, 1]	[1, 1, 0.25]	[0.73, 1, 0.4]	2.58
Decision Tree (Gini)	0.96	[0.89, 1, 1]	[1, 0.88, 1]	[0.94, 0.93, 1]	1.50
Decision Tree (Entropy)	0.98	[0.95, 1, 1]	[1, 0.94, 1]	[0.97, 0.97, 1]	0.67
Multi Layer Perceptron	0.98	[1, 1, 0.95]	[0.94, 1, 1]	[0.97, 1, 0.97]	0.67

6 From Signs to Fluid Speech

The last step for a seamless and practical device is to generate a phrase that can be easily understood by the common person. As already introduced in the beginning, Sign Language lacks most of the semantic and grammar specifics other languages have. Although one can usually understand the context of a phrase that is composed by dispersed words, this still poses a barrier for the deaf person, limiting its authority and credibility.

The proposed approach to solve this issue is to use an already developed Natural Language Processing (NLP) model, to easily convert from dispersed words to more fluid speech. Here it is used *OpenAI's gpt-3.5-turbo* model, commonly known by the online chatbot *ChatGPT*, which uses it. *OpenAI* makes available an Application Programming Interface (API) that enables two software components (i.e., personal computer and *OpenAI* servers) to communicate with each other using a set of simple and straightforward definitions and protocols. The prompt and software snippet used can be found in Appendix A.

To test the above approach 3 different sentences were tested, feeding *OpenAI*'s API a list of words which can be seen in the Portuguese sign language dictionary and comparing the generated and target sentences. The Translation Edit Rate (TER) metric [15] is used to have a quantitative measurement of the success of the translation. This metric calculates the number of edits that are required to change the generated output into the target sentence. The tests performed are shown in Table 3, showing a relatively successful translation rate, only missing in subtle aspects like translating to a question for example. Although successful, this approach also has it's own drawbacks. Using an external system, through an API, to make the translation leads to higher response times than what could be achieved using a custom made model. Such custom made and lighter model is even more important when trying to perform real-time translation, meaning building the sentence on-the-go. As of now, the current approach is only able to perform the conversion into a fluid phrase once all the necessary signs are translated into words.

Table 3: Example of Sentence Generation Results

Input Set of Words	Generated Sentence	Target Sentence	TER
{"*Eu*", "*Comer*", "*Acabar*", "*Depois*", "*Café*", "*Tomar*"}	"Eu acabo de comer e depois vou tomar café"	"Vou tomar café depois de acabar de comer"	0.86
{"*Menu*", "*Posso*", "*Por favor*"}	"Menu, por favor"	"Posso pedir o menu, por favor?"	0.67
{"*Chocolate*", "*Gosto*", "*Muito*"}	"Eu gosto muito de chocolate"	"Eu gosto muito de chocolate"	0.0

With all the parts now tackled, the missing task is to merge it altogether. First, besides the words trained in Section 5, 3 more words are similarly trained, namely "*Eu*", "*Posso*", "*Gosto*", "*Chocolate*", "*Cafe*". A new Decision Tree model is then trained and saved. With the goal of saying the sentence "Eu gosto de chocolate", which in LGP is mainly gestured using the ordered sequence of words "*Eu*", "*Chocolate*" and "*Gosto*", the device was successfully able to capture, segment and classify the 3 words, also generating and producing an audio of the complete sentence.

7 Conclusion & Future Work

This work, with the intention of making deaf people heard, proposed a wearable device, attached to the person's wrist, that can translate Portuguese sign language gestures into fluid spoken phrases. The device, using MPU sensors that track the linear acceleration and angular velocity of the hand, was: 1) able to send data in real-time to one's computer, via WiFi; 2) pre-process and segment an entire phrase into distinct signs; 3) classify each of the hand gestures into Portuguese language words and 4) transform the set of words into a fluid sentence, presenting it in audio format. Nevertheless, the above approach still has room for improvement. Although most of the methods used to tackle each challenge are successful in the real-time aspect of it (i.e., data is transmitted in real-time or the segmentation performed using a window-like approach), the classification and translation into fluid speech still lack the speed required for such a real-time processing. For this to happen, more lighter and custom made models should be used instead. Instead of using *scikit-learn* library or *OpenAI*'s API, a model that classifies on-the-go, using Natural Language Processing techniques could be used to classify each word, start generating a sentence and output it in real-time. To improve the end-user experience, instead of a computer, a mobile application could be used, to process, classify and output the sentences. This would be a much lighter approach and, if successful, would for sure help any deaf person make herself heard in such a competitive world.

References

[1] Ryan Prescott Adams and David J. C. MacKay. Bayesian online changepoint detection, 2007.

[2] Vladislav Apostolyuk. *Coriolis Vibratory Gyroscopes: Theory and Design.* 01 2016.

[3] Mohd. Salman Hossain Bhuiyan, Nazmus Sakib Patwary, Protap Kumar Saha, and Md. Tanvir Hossain. Sensor-based human activity recognition: A comparative study of machine learning

techniques. In *2020 2nd International Conference on Advanced Information and Communication Technology (ICAICT)*, pages 286–290, 2020.

[4] Liming Chen, Jesse Hoey, Chris D. Nugent, Diane J. Cook, and Zhiwen Yu. Sensor-based activity recognition. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6):790–808, 2012.

[5] Nelly Elsayed. Vision-based american sign language classification approach via deep learning. *The International FLAIRS Conference Proceedings*, 35, May 2022.

[6] Espressif Systems. *ESP32-C3 Series Datasheet*, 2024.

[7] Léo Grinsztajn, Edouard Oyallon, and Gaël Varoquaux. Why do tree-based models still outperform deep learning on tabular data?, 2022.

[8] Yutong Gu, Chao Zheng, Masahiro Todoh, and Fusheng Zha. American sign language translation using wearable inertial and electromyography sensors for tracking hand movements and facial expressions. *Frontiers in neuroscience*, 16, 2022.

[9] InvenSense Inc. *MPU-6000 and MPU-6050 Product Specification Revision 3.4*, 2013.

[10] K. W. Kurose, J. F.; Ross. Computer networking: A top-down approach. *Boston, MA: Pearson Education*, 2010.

[11] Kyung Hyun Lee, Ji Young Min, and Sangwon Byun. Electromyogram-based classification of hand and finger gestures using artificial neural networks. *Sensors*, 22(1), 2022.

[12] Yuxuan Liu, Xijun Jiang, Xingge Yu, Huaidong Ye, Chao Ma, Wanyi Wang, and Youfan Hu. A wearable system for sign language recognition enabled by a convolutional neural network. *Nano Energy*, 116:108767, 2023.

[13] Adrián Núñez-Marcos, Olatz Perez de Viñaspre, and Gorka Labaka. A survey on sign language machine translation. *Expert Systems with Applications*, 213:118993, 2023.

[14] Maria Papatsimouli, Konstantinos Kollias, Lazaros Lazaridis, George S. Maraslidis, Heracles Michailidis, Panagiotis Sarigiannidis, and George Fragulis. Real time sign language translation systems: A review study. pages 1–4, 06 2022.

[15] Matthew Snover, Bonnie Dorr, Rich Schwartz, Linnea Micciulla, and John Makhoul. A study of translation edit rate with targeted human annotation. In *Proceedings of the 7th Conference of the Association for Machine Translation in the Americas: Technical Papers*, pages 223–231, Cambridge, Massachusetts, USA, August 8-12 2006. Association for Machine Translation in the Americas.

[16] Paulo Trigueiros, A. Fernando Ribeiro, and L. P. Reis. Vision-based portuguese sign language recognition system. *New Perspectives in Information Systems and Technologies*, April 2014.

[17] Zhihao Zhou, Kyle Chen, Xiaoshi Li, Songlin Zhang, Yufen Wu, Yihao Zhou, Keyu Meng, Chenchen Sun, Qiang He, Wenjing Fan, Endong Fan, Zhiwei Lin, Xulong Tan, Weili Deng, Jin Yng, and Jun Chen. Sign-to-speech translation using machine-learning-assisted stretchable sensor arrays. *Nature Electronics*, 3:571, 2020.

Appendices

A Software Snippets

The *Python* scripts below, as also the complete code developed, can be found in the Github repository here: SaraL-Oliveira/SignMeUp. A quick video showing a successful translation can be found in this *Google Drive* location.

Retrieve ESP32C3's IP

```
  WiFi.begin(ssid, password);
  // Wait for connection
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("Connected to wifi");
  Udp.begin(localUdpPort);
  Serial.printf("Now listening at IP \%s, UDP port \%d\\n",
   ↪  WiFi.localIP().toString().c_str(), localUdpPort);
\label{code:ip}
```

Prompting OpenAi

```
string_of_words =  "bom, dia, gosto, eu, morangos, maduros"

prompt = "Convert into a fluid phrase in portuguese the following:
          " + string_of_words

# Now we do our request to ChatGPT
try:
    completion = my_openai.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[{"role": "user", "content": prompt} ]
    )
    return completion.choices[0].message.content
except:
    return "Not able to return a fluid phrase!"
\label{code:openai}
```