UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER THESIS

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

Design and evaluation of electromyography signal processing techniques using resource-constrained devices

Pablo Sarabia Ortiz

JULY 2020

MASTER THESIS

Title: Design and evaluation of electromyography signal processing techniques using resource-constrained devices.

Author: Pablo Sarabia Ortiz

Advisor: Roberto Rodriguez Zurrunero

Academic advisor: Alvaro Araujo Pinto

Departament: Electronic Engineering Department

COMMITTEE:

Supervisor: D.

Reader: D.

Secretary: D.

Date: 2020

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

MASTER THESIS

MÁSTER UNIVERSITARIO EN INGENIERÍA DE SISTEMAS ELECTRÓNICOS

Design and evaluation of electromyography signal processing techniques using resource-constrained devices

Pablo Sarabia Ortiz

JULY 2020

Abstract

Surface electromyographic (sEMG) is an acquisition technique based on recording muscles potential over the skin. Processing those signals on a resource-constrained device is a key element in achieving wearable health devices. sEMG based devices have a wide range of applications: early diagnose and treatment of neurodegenerative diseases, tracking of daily activities, rehabilitation, and adaptive training. For most of these applications, it is required to identify which gestures the user is doing. This classification is difficult due to low signal to noise ratio, data complexity and size, and changes between sessions and subjects. These are a major challenge in a resources-constrained device.

The sEMG based devices consist of two parts: the signal acquisition and the processing of the data generated. This master thesis focuses on the data processing with two differentiated parts: An analysis of a public gesture dataset and the design of a classifier.

The quantitative analysis to the public available gesture dataset was made using Parallel Factor Analysis (PARAFAC) decomposition. It was found that reduction from 14 to 4 channels was possible without losing significant information. Better understanding of the sEMG signal was achieved, estimating that the most significant information is located under 300 Hz.

In this work, raw sEMG signals have been acquired from 4 passive electrodes and 8 and gestures have been classified using Support Vector Machine (SVM). The classifier achieved over 85% recognition accuracy similar to the state-of-the-art typical results. A factorial design was carried out to identify the relations between the coding of the SVM, the datatype used, the length of the sample and their influence in the memory footprint of the classifier and execution time. Memory was found to be the bottleneck for an embedded device implementation of the SVM algorithm. The SVM execution time was 130 ms and its memory footprint was 868 KB.

Keywords

Surface Electromyography, sEMG, Electromyography, EMG, Gesture classification, SVM, PARAFAC, factorial design.

Resumen

El procesado de las señales de electromiografía superficial (sEMG) en sistemas con recursos reducidos es clave para conseguir dispositivos de salud portátiles. Los dispositivos sEMG tienen un gran rango de aplicaciones desde su uso en la diagnosis y tratamiento de enfermedades neurodegenerativas, registro de las actividades diarias, rehabilitación hasta para entrenamientos personalizados. La mayoría de estas aplicaciones necesitan identificar los gestos que realiza el usuario. Esta tarea es difícil debido a las características de la señal como son: baja relación señal ruido, complejidad de los datos y las diferencias entre diferentes sujetos y sesiones.

Los dispositivos se componen de dos partes: la adquisición de la señal y el procesado de los datos adquiridos. Este Trabajo Fin de Máster se ha centrado en el procesado de la señal y consta de dos partes: primero se ha realizado un análisis de un set de datos público y segundo, un diseño de un clasificador de gestos realizados con la mano.

El análisis cuantitativo de una base de datos pública de gestos se ha llevado a cabo mediante una descomposición PARAFAC (Parallel Factor Analysis) con el objetivo de averiguar la influencia del número de canales, del cambio entre pacientes y de la frecuencia en la información subyacente. Se consiguió reducir el número de canales de 14 a 4 sin perdida significativa de información. Además, permitió estimar que las señales sEMG contienen la información más relevante en las frecuencias inferiores a 300 Hz.

Se han adquirido señales sin pretratar ni filtrar de 4 electrodos pasivos y se ha logrado clasificar 8 gestos diferentes usando SVM (Support Vector Machine). Se obtuvo una precisión en concordancia con la bibliografía revisada (más del 85 %). Se realizó un diseño factorial para analizar las relaciones entre la codificación del SVM, el tipo de dato utilizado, la longitud de la muestra y su influencia en la huella de memoria y el tiempo de ejecución del clasificador. Se concluyó que el cuello de botella para la ejecución del SVM en un sistema empotrado es su huella de memoria. El clasificador final SVM logró tener un tiempo de ejecución (130 ms) y una huella de memoria de 868 KB.

Palabras clave

Surface Electromyography, sEMG, Electromyography, EMG, Gesture classification, SVM, PARAFAC, Factorial design

Contents

Agradecer a mi tutor Rober
por la disponibilidad y todo lo que me ha enseñado.

A mi familia,
por el apoyo constante la confianza y los ánimos en los momentos difíciles.

A mi segunda familia en esta cuarentena Gabri y Vicky,
que han hecho que la cuarentena fuese más sencilla.

Por último, a mis compañeros del máster,
por el buen ambiente, y ser unas grandísimas personas.

1. INTRODUCTION

Healthcare has always been one of the aspirations of human being. In the past century substantial improvements were made in the field, but the limit of the generic and non-targeted medicine in western countries is almost achieved. For this reason, preventive medicine, and the use of technological advances to keep improving our wellness must be made. Putting the focus on one of our day's reality, the population is older and some neurological diseases (Parkinson) that could be terminal and degenerative open a new field of improving the daily life of the patients. First, developing fast methods to facilitate the diagnose of the disease to allow starting an early treatment. Second, creating customized solutions for each individual improving their daily life.

This search of new customized devices has led to the use of formerly considered not viable techniques to a rise. One of them is electromyography (EMG), which was previously only used as a diagnostical tool that was expensive and required complex instruments located in hospitals. EMG consists of measuring the electrical potential made by muscles in its contraction. EMG is used in a variety of fields such as: rehabilitation [1], early neurological disease diagnose [2] [3] and treatment, tracking of daily activities [4], Human Machine Interface (HMI) and prothesis control [5], among others. Surface EMG (sEMG) is a variant of EMG that uses noninvasive techniques (electrodes on skin) and provides a safer alternative compared to the use of needles or implants, which require surgical operation.

Although in later years there has been important advances in the field, they are not enough to make it feasible for general use. One of the challenges consists of making developing portable devices. These smaller devices need to run on batteries and should be comfortable enough to be worn all day long. The challenges are great and to the best of our knowledge there is no commercial solution yet that fulfills these requirements. There are several issues that should be addressed by doctors and engineers such as: noisy signals with low signal to noise ratio, hard to acquire signals because of the movement of electrodes. These characteristics of sEMG yields to the use of powerful systems that are not compatible with the low energy requirements.

The EMG devices can be divided in two separate subsystems, signal acquisition and data processing. Both are intrinsically related because the data acquisition (sampling rate, number of channels, ADC precision, etc.) has a huge impact in what can be done in the processing and sets the data bandwidth that the processing unit has to deal with. As an example of how the acquisition step has a vast impact in the processing part is the following: Increasing the number of channels to improve the signal to noise ratio also leads to bigger resources needed for processing all the data collected. In some cases it has been used 64 [6, 7]or even 128 channels [8]. To put the problem in figures 128 channels at 1kHz sampling rate with a 24-bit ADC can produce 384 KB of information in just one second, for comparison purposes the microcontroller used in this work has just 128 KB of RAM. So far only about memory has been talk, but the aim of this devices is to fulfill a function more than just logging the data in to storage. Many of the applications include the use of classifiers, to detect the movement that the user is doing. A classifier is a complex algorithm that requires of memory and processing power. Different approaches have been taken, such as is the use of application specific integrated circuits (ASIC), the problem with this approach is the cost and the lack of flexibility.

Another challenge to be addressed is dealing with the physiological differences between the subjects. Robust algorithms must me developed that require little or no calibration when applied to other person different from the original or the electrodes are replaced.

On the other hand, achieving wearable EMG devices with processing capabilities for detecting the gesture that the user is doing will also make possible to design adapted trainings for athletes that prevents injuries in trainings. It will also allow to improve rehabilitations by knowing the actual state of the muscles and keeping the optimal intensity. In the long term, the objective is to do closed loop stimulation [8], that consists in stimulating the patient while sensing the EMG. This technique could be used to mitigate the effects of neurodegenerative diseases like Parkinson [9].

1.1 Objectives

The objectives of this work are:

- Characterize the EMG signals and measure the relevance of frequency, the number of channels and the patient.
- Design and implement a classifier of gestures oriented to processing in resources constrained targets.
- Describe the influence of the design variables for the characteristics of the implemented classifier.

1.2 Project stages

The approach is to design simple and modular subsystems that can either be reused or improved without affecting the rest of the system.

Several attempts have been made and they are still being made by our colleges to achieve the goal of a portable EMG device. This master thesis will try to contribute to this area by designing and evaluating an EMG classifier oriented to resource constrained low-energy embedded platforms.

The project divides as follows.

- Analysis of publicly available gesture dataset, a Parallel Factor Analysis (PARAFAC) decomposition was made to identify where the information is located (frecuency) and identify the relevance of the number of channels and the variation between session and subject.
- Development of the software to acquire, send and store the data from a 4 channel sEMG acquisition system based on a MCU.
- Design of a classifier for the 8 gestures recorded.
- Design a factorial model to understand the impact of each design variable in the result (precision, memory footprint and processing time) and improve the memory footprint and computation time of the classifier.

1.3 Document organization

The document is composed of nine chapters:

- *State of art* describes the physical characteristics of EMG signals; the challenges and how other works have addressed the problem of processing these signals.
- *Tools* depicts the hardware and software used in this work.

- *Gesture analysis* introduces decomposition techniques that will be used to analyze the EMG data.
- *Experimental acquisition of dataset with μC* explains the design and decision taken to acquire EMG data.
- *Classifier* supports the election of a classifier and the design of it.
- *Design for embedded systems* contains all the explanations related on how the classifier was improved for its implementation in an embedded system.
- *Methodology* describes the procedures and gives detailed information of each of the previous design chapters implementation.
- *Results* contains the outcomes and discusses them.
- *Conclusion and future work* summarize the advances made in this thesis and future steps.

The document also contains a bibliography and 3 appendices: environmental and social impact, budget, and code.

2. STATE OF THE ART

In this chapter a review of the existing works has been carried out. There are a lot of works about the methods for acquiring the signals and how to classify them.

2.1 Electromyography (EMG) Characteristics

EMG signals are generated by physiological variations in the muscle membranes. The muscles contract when they are excited by the nervous system. This excitation is made by varying the ion amount in between the muscle cellule and the exterior, this exchange of ions leads to charge negatively the cell, shortly after the ions flow back in the opposite direction restoring the neutral charge. This process is called depolarisation and repolarisation [10].

EMG signals can be modelled as a small capacitor that charges and discharges and causes the potential swings that can be observed in the surface of the skin. The complete model is more complex because includes the interactions between other muscles, the skin, and the nervous system. The characteristics of the signal also depend on the type of contraction made by the muscle (isometric, concentric or eccentric) [2, 11].

The signal amplitude goes from microvolts to 20-30 mV and is proportional to the intensity of the contraction. These signals are in the frequency range of 20-500 Hz, Table 1. There is some controversy about this point because some authors state that there is information in the kHz range [12] however most of the projects target 500 Hz as the maximum frequency [13].It is important to note that the current of the signals is low in the range of picoamperes or femtoamperes. This leads to an important focus in the acquisition interface of the system that must have a large input impedance, and great gain

Table 1. EMG electrical characteristics.

Electrical characteristic	Typical range
Voltage	$\mu V - mV$
Frequency	20 - 500Hz
Current	fA - pA

2.2 Challenges in EMG signals

EMG signals have low voltage values so the problem of noise that mask the underlying information is present. The types of noise in EMG are [10, 13]: Inherent noise due to electronics equipment, ambient noise, motion artifact, inherent instability of signal, electrocardiographic (ECG) artifacts and crosstalk. In this work the approach has been to evaluate the overall precision of the classifier. In other works, there is an emphasis in improving one characteristic (e.g. filtering 50 Hz noise [14]) however this could lead to expend resources (design time, computational resources or latency) without evaluating the impact on the final application.

To understand the information underlaying in EMG recorded data the public available dataset [15] is discussed in the Gesture Analysis chapter.

The main problem, lack of robustness in the model when the subject is changed or data is taken in different days or the movement between sessions of the electrodes [12]. The solution given is to use a large number of channels (64) to overcome the variations between days and/or subjects [7]. A different approach is to aggregate the data between

different subjects and to choose the features that best classify the gestures over the time [16].

2.3 Implementation

There are different trends in how to acquire the signals: some of them consist of using medical grade equipment to record the signals [12], bulky equipment with great precision but not feasible for a wearable design. Others are using custom hardware designs, such as ASIC [6, 8, 17]. And a third approach uses microcontrollers with off the shelf ADC [3, 14, 18].

2.3.1 Feature extraction

There are also several approaches to the problem of classification the gestures. As mentioned before, EMG applications can generate a lot of data per second, feature extraction helps to mitigate this problem. Feature extraction is a technique that aims to reduce the dimensions of the input vector to the classifier. There are various feature extraction techniques with different complexities, it can consist of a frequency filter applied to the input, a zero crossing or any other conversion that can be applied to the sample [19].These techniques reduce the memory footprint and or improve the classifier performance. When feature extraction does not compress enough the data, then a dimensionality reduction is used. This consists of using a mathematical technique that allows to reduce the dimensions of the vector without losing of information.

The main feature extraction techniques are time domain (TD), frequency domain (FD) or time-frequency domain (TFD), a mixture of both. TD feature extraction is less computational expensive but has worse results that its FD counterpart. The use of classifiers in real-time processing would require the use of windowing for the use of FD feature extraction.

The selection of features is an important factor in the success of the application. TD are used to detect and classify a gesture whereas FD is more commonly used for understanding the state of the muscles, fatigue, degradation, or some type of muscle conditions. Not all features work well with all the classifiers, for example Principal Component Analysis (PCA) works very well with Linear Discriminant Analysis (LDA) [13].

2.3.2 Classification techniques

There are numerous classification techniques, of which three of them have been discussed. There have been chosen because there are most popular among the literature consulted [13, 20]: LDA, Support Vector Machine (SVM) and Artificial Neural Network (ANN) [21].

LDA is a linear classifier that allows multiclass classification, this means that one classifier can predict the class of the input vector. LDA has lower memory footprint that SVM and ANN [20]. However, LDA works only with normal distributed samples because it relays in the fact that the information is in the mean of the samples. This condition is not meet in sEMG data.

SVM is a linear and binary classifier. The key advantage of this classifier is that it does not require pretreatment or feature extraction. The memory footprint grows lineally with the size of the input vector. Because SVM can only be used as a binary classifier, if there is more than two classes a coding matrix is needed to split the multiclass problem in a number of binary classification problems [22]. Two coding are reviewed: “one vs one”

and “one vs all”. One vs all requires of a lower number of binary classifiers but has lower performance.

ANN consists of an input layer of neurons, two or more hidden layers and an output layer. It outperforms the others in terms of classifier precision, but it also has a huge computation time for predicting and has a great memory footprint.

2.3.3 Real-time processing

In the works reviewed there are two different groups: the papers that use embedded devices for processing the data and those using a computer, a smartphone or other device with high processing capabilities. In Table 2 it can be observed that real time classification when done in an embedded system is done with SVM: [5, 17, 23]. When more powerful platforms are available other classifiers are used [24] and [7].

Table 2. Summary of the real time classifying applications and its characteristics.

Response time (worst case)	Accuracy	Classifier	Hardware (where the classifier is runnning)		Sampling frequency	Reference
			CPU	RAM + Flash		
16 ms	99.9 %	ANN	Intel Edison, (dual Atom core x86)	1 + 4 GB	67 Hz	[24]
1.8 ms	98.3 %	SVM	ARM Cortex M4	n/a + 64 KB	500 Hz	[5]
41 ms	88%	SVM	ASIC (4 openRISC cores) [25]	160 KB + external flash	1000 Hz	[17]
5 ms	89.2 %	SVM	ARM Cortex M4	192 KB + 1MB	450 Hz	[23]
n/a	96.64 %	Hyper Dimensional Vectors	Desktop CPU	n/a	200 Hz	[7]

3. TOOLS

A detailed description of the hardware and software used is related in this chapter.

3.1 Software

To analyze the dataset two alternatives were evaluated: MATLAB or Python. This two were selected because both are high level easy to prototype with and with lots of libraries available in the field of data analysis and statistics. The data was provided in a MATLAB file, so for simplicity MATLAB was used.

For designing the classifier MATLAB was used because it has libraries for all the proposed classifications techniques and its simple to change parameters for the embedded design implementation. It also has a C/C++ coder that is useful for checking the approximate size of the trained algorithm.

The acquisition design was carried out using STM32CubeIDE provided by ST. The hardware platform was running a custom version of FreeRTOS called YetiOS [26].

The terminal serial chosen is RealTerm v2.0 because it has more options than its counterpart Putty. It has the feature of recording the input serial for a requested time. It was proven to be more stable than other options, not crashing when debugging the code in the embedded platform.

3.2 Hardware

The embedded system used in this thesis is Cortex M4 by ST (STM32L486RG) at 80 MHz [27] with 128kB of RAM and a four channel ADC by Analog devices (AD7124-4BBCPZ-ND) [28] with a differential amplifier configuration, shown in Figure 1. It also includes an external SD card and a serial interface, that can be used for logging data.

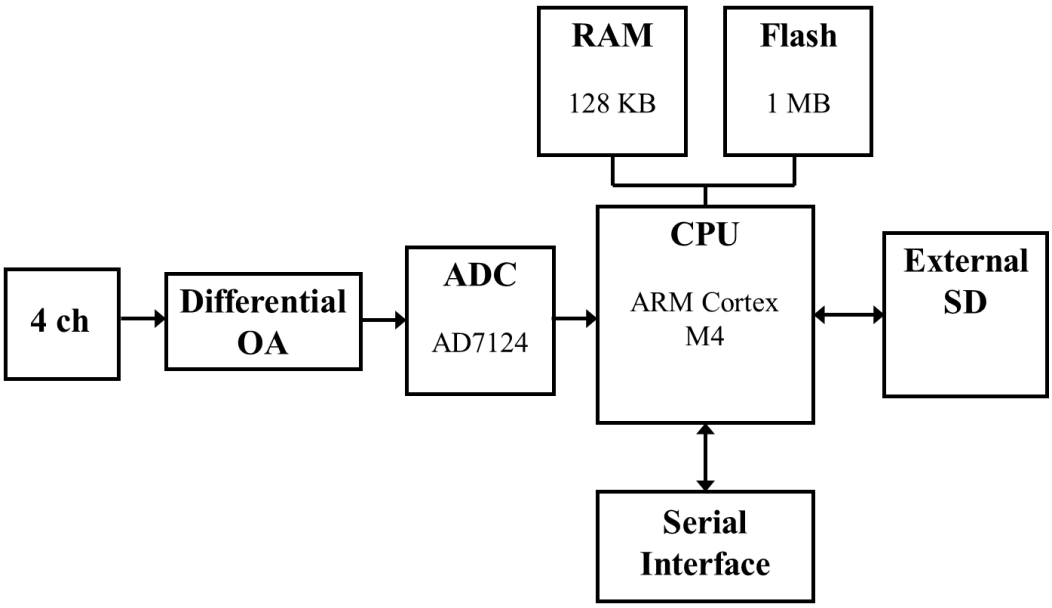


Figure 1. Block diagram of the hardware used.

The complete hardware setup is depicted in Figure 2. The electrodes and the differential amplifier are on the left side of the picture. The stack of boards consists of the ADC and the MCU. The USB connection is on the top side.

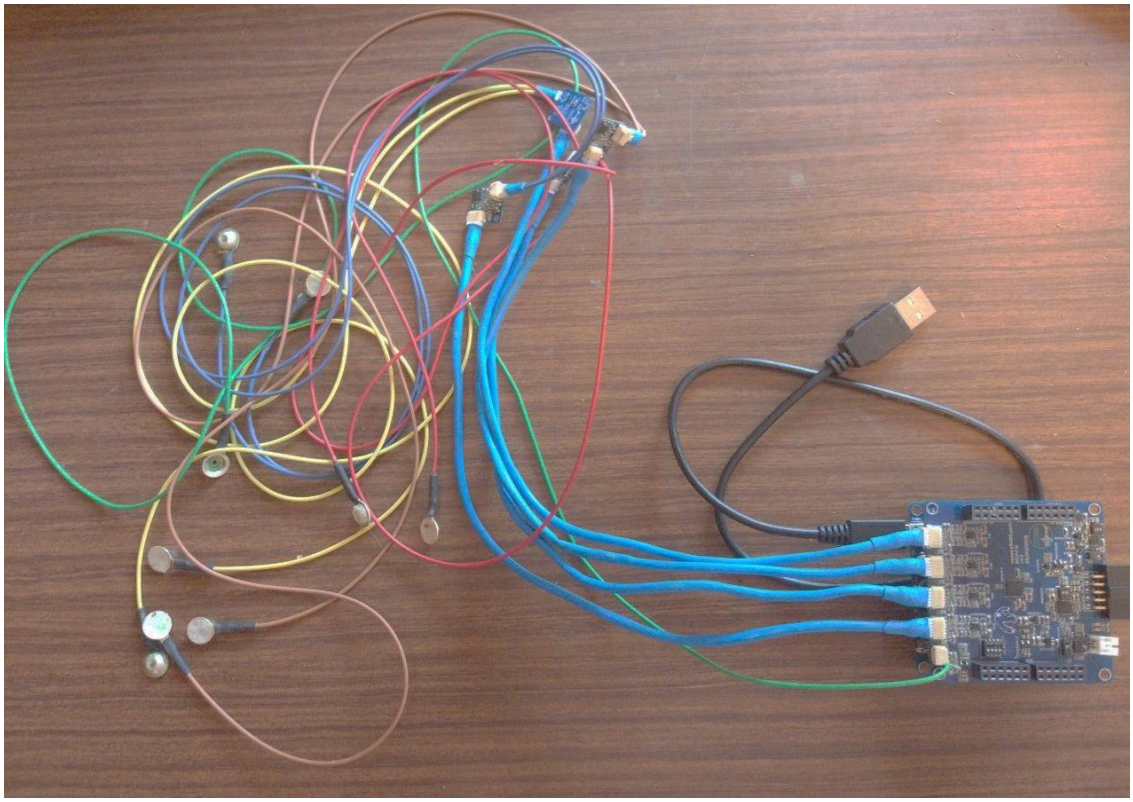


Figure 2. Hardware used in this work.

4. GESTURE ANALYSIS

The aim of this chapter is to understand how robust the information is, and the impact of changing the day or the subject. For this reason the NINAPRO [15] dataset 6 is very good suited for understanding how data changes between subjects and between sessions. The dataset contains data from different subjects in different days and in different times of the day. Each sample in the dataset is labeled with gesture, subject, day, and daytime.

4.1 Data structure: matrices and tensors

Data representation is central to data analysis. Vectors and matrices provide well-known structures to collect, store and represent data. Vectors and matrices consist of one and two modes, respectively, denoting them as one-way and two-way data objects. In the case of three-way data, PARAFAC decomposes a data tensor **X** into triads or trilinear factors [29, 30, 31, 32, 33] . The length of all samples must be equal, so a feature extraction is applied. A Discrete Fourier Transform (DFT) is applied to all the samples. In our case, measuring an EMG signal (each of the *K* arrays labeled as a gesture), for *J* frequencies yields a matrix, $\mathbf{X}_{\text{IxJ}}$, with *K* rows and J columns. One matrix is register for each of the *K* channels obtaining a three-way tensor $\mathbf{X}_{\text{IxJxK}} = (x_{ijk})$. This tensor has its trilinear PARAFAC model:

$$x_{ijk} = \sum_{f=1}^{F} a_{if}\, b_{jf} c_{kf} + e_{ijk}, \qquad \textbf{(Eq. 1)}$$

$$i = 1,2,\dots,I; j = 1,2,\dots,J; k = 1,2,\dots,K$$

Where F is the number of factors (sources of variability). $\boldsymbol{a_f}$, $\boldsymbol{b_f}$ and $\boldsymbol{c_f}$ are the loading vectors of the case, spectral and channel profiles, respectively, and $e_{ijk}$are the residuals of the model. The coordinates of the loading vectors are the columns of matrix **A**, **B** and **C** of size I × F, J × F and K × F, respectively.

In Figure 3 there is a representation of how the data array, containing all samples and labels, $\boldsymbol{X_{ijk}}$ is decomposed in several loading vectors $\boldsymbol{a_{if}}, \boldsymbol{b_{jf}}, \boldsymbol{c_{kf}}$ to explain the underlaying structure of the data. As mentioned before *f* is the factors.

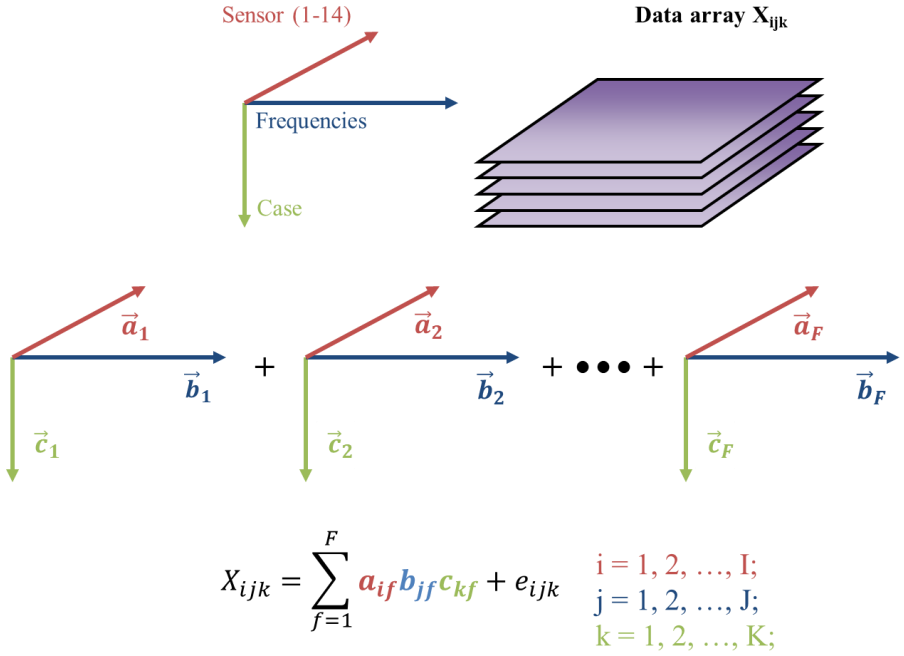


Figure 3. PARAFAC model representation

Data are trilinear if the experimental data tensor is compatible with the structure in (Eq. 1) . In this case, the estimation by least squares of all the coefficients that intervene in that equation is unique. The core consistency diagnostic (CORCONDIA) [34] measures the trilinearity degree of the experimental data tensor. If the data tensor is trilinear, then the maximum CORCONDIA value of 100 is found. Split-half procedure [33] asses the stability of the PARAFAC solution by splitting the data tensor in half and to perform a separate analysis on both parts. If there is a true underlying solution, it should show up in both analysis [33].

PARAFAC decomposition is a generalization of the Principal Component Analysis (PCA) [35, 36], that for a given matrix $\boldsymbol{X_{IxJ}} = (\boldsymbol{x_{ij}})$ is described as

$$x_{ij} = \sum_{f=1}^{F} a_{if}\, b_{fj} + e_{ij}, \qquad i = 1,2,\dots,I; j = 1,2,\dots,J \qquad \textbf{(Eq. 2)}$$

Where F is the number of principal components, the matrix **A** with IxF dimensions are the scores of I samples in the F components while the matrix **B** with FxJ dimensions contains as rows the loadings of each component.

A fundamental difficulty in the case of low-rank matrix factorization is its non-uniqueness. Given a matrix **X** with factorization $\mathbf{X} = \mathbf{AB}^{T}$, one can always find an orthogonal matrix **D** such that $\mathbf{X} = \mathbf{AB}^{T} = \mathbf{ADD}^{-1}\mathbf{B}^{T} = \mathbf{A}^{*}\,\mathbf{B}^{*T}$ with $\mathbf{A^*} = \mathbf{AD}$ and $\mathbf{B^*} = \mathbf{BD}$. The lack of (essential) uniqueness prohibits a meaningful interpretation of the factor vectors. Additional constraints are typically imposed on **A** and/or **B** to achieve (essential) uniqueness. Examples of constraints are orthogonality, triangularity and nonnegativity, leading to the singular value decomposition (SVD), LQ or QR decomposition and nonnegative matrix factorization (NMF), respectively.

While we have discussed that low-rank matrix factorization techniques require additional constraints to yield a unique solution, a low-rank PARAFAC decomposition is essentially unique under mild conditions by itself. Essential uniqueness allows the factor vectors to be determined up to scaling and permutation. It is thus valid to see factor vectors as the underlying components of the data. And consequently, PARAFAC was chosen over PCA because its essential uniqueness that shows the underlying components of the data.

5. EXPERIMENTAL ACQUISITION OF DATASET WITH μC

In this chapter the design for the acquisition of our own dataset is being discussed. The dataset has been obtained using a similar procedure as the studied dataset (NINAPRO) [15] used before. The purpose is to be able to compare both datasets, and for being able to test the classifier, the feature extraction and compare results. The forearm was chosen because of three reasons: i) There is a comprehensive dataset ii) It is an easy limb to attach electrodes and physiologically the muscles are superficial iii) It has a great variety of gestures.

The objective is to have an acquisition system that logs the data and transmit it to the computer for further analysis. It is meant to be fast, modular, and flexible enough for future reuse.

It was decided not to do any cast in the MCU to not condition the classifier to any datatype. Samples are sent and stored in the computer as int32_t as they came out of the ADC.

5.1 Buffer design

The samples need a temporal buffer in the microcontroller to be stored. A buffer is required because the acquisition of the signal is by means of an interrupt, so samples are stored for its later transmission over serial. A circular buffer was implemented in C. A circular buffer (o circular array) consists of an array of N size and two pointers, head, and tail. This buffer has the particularity that once the head pointer reaches the end of N element of the array the next iteration points again to the first element of the array. This allows the buffer to have fixed size during all execution. As the memory used is always the same it can be statically allocated, and heap fragmentation is avoided.

To handle the buffer a library (circbuff.c/circbuff.h) has been develop. The user is provided with a circular buffer datatype and eight functions to manage it. The circular buffer expects int32_t and four channels.

The library purpose is to encapsulate the underlying data structure so the user just uses the application program interface (API) and avoid problems, like writing out of index array that can lead to the failure of the program. The API is designed to be intuitive and easy to use.

The user is provided with functions for:

- Initialize the buffer, cBufInit().
- Erase the buffer and, cBufReset().
- Check the remaining space of the buffer, cBufSize().
- Know the size of the buffer, cBufCapacity().
- Append data to the buffer, cBufPut().
- Read the data from the buffer, cBufPop().
- Check if the buffer is empty or full, cBufisempty(), cBufisfull().
- Know which is the next channel that the buffer is expecting cBufexpectedchannel().

The operation of the library is the following.

1. The users allocates the fours arrays using malloc() or in our case pvPortMalloc(), from FreeRTOS.

```
Buffer0 = (int32_t*) pvPortMalloc(sizeof(uint32_t)*BUFFER_LENGTH);
Buffer1 = (int32_t*) pvPortMalloc(sizeof(uint32_t)*BUFFER_LENGTH);
Buffer2 = (int32_t*) pvPortMalloc(sizeof(uint32_t)*BUFFER_LENGTH);
Buffer3 = (int32_t*) pvPortMalloc(sizeof(uint32_t)*BUFFER_LENGTH);
```

2. The user initialize sthe circular buffer.

```
cBuf = cBufInit(Buffer0 ,Buffer1 ,Buffer2 ,Buffer3 , BUFFER_LENGTH);
```

3. Now the user can use any of the API functions, e.g. write data and read data

```
cBufPut(cBuf, sample, expectedChannel);
```

Where sample is an int32_t variable containing the value desired to store and expectedChannel is an uint8_t variable containing the channel of the sample

```
cBufPop(cBuf, outDataPtr);
```

Where outDataPtr is an int32_t pointer that has the address of where the user wants to store the sample.

5.2 Final design

The overall system, Figure 4, consists on storing the samples in a circular buffer in the microcontroller and then send it by serial to the computer. The data then is captured using RealTerm and processed using MATLAB function serialprocessdata(), available in the code appendix.

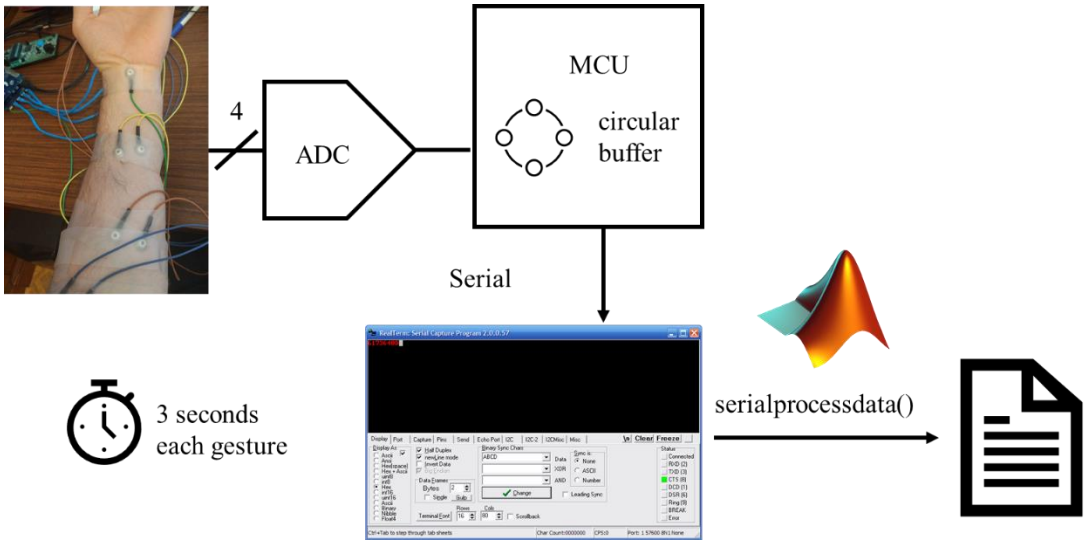


Figure 4. Detailed description of serial acquisition system.

6. CLASSIFIER

In this chapter the election and design of the classifier is discussed. Three classifiers are studied LDA, SVM and ANN. The choice of the classifier was determined according to its suitability for the data, its performance, and its feasibility for an embedded system.

It has been chosen to not use any feature extraction, because of two reasons. First, keep the algorithm as simple as possible so it can be a base for future work and explore the possibilities of the hardware and the acquisition system. Second, a much larger work is required to investigate the combination of each feature with the selected classifier. However, the workflow used, and methodology is flexible enough to include feature extraction in the future.

6.1 Selection of classifier

The estate of the art chapter reveals that the algorithm used for embedded systems is SVM, Table 2. The other alternatives LDA or ANN have high precision but require a larger memory, that is not usually available in an embedded system. In Table 3, a summary of the main characteristics of the classifier is presented.

Table 3. Comparison of the different classifiers.

	LDA	**ANN**	**SVM**
Precision	Very high	Very high	High
Execution time	Medium	High	Low
Memory footprint	Low	High	Low
Training time	Low	High	Medium
Complexity	Medium	Very high	Medium
Available in MATLAB	Yes	Yes	Yes
MATLAB C Coder	No	No	Yes

Although LDA and ANN are more precise SVM has been proven to be simple yet powerful enough to achieve precision rates above 80% [5] [17] [23]. LDA was discarded because feature extraction is required to work. Between ANN and SVM, SVM was chosen because is simpler and can be implemented directly by MATLAB C coder. Also, ANN requires a memory and processing resources that are not compatible with an embedded system.

6.2 Support Vector Machine

SVM is a binary (two classes) classifier, that uses an plane w and a bias b or to predict if a input sample belongs to one class or the other, Figure 5. It is based in maximizing the distance between the most similar samples between both classes. This approximation is only useful for simple problems where the input is a scalar, if the input is a vector $\boldsymbol{X}$ then a hyperplane and a kernel is required to identify the classes. In this thesis for simplicity a lineal kernel was used obtaining the (Eq. 3) for the SVM model.

$$y = \frac{X'}{s} \times \beta + b \qquad \textbf{(Eq. 3)}$$

Where *y* is the response, $\boldsymbol{X}$ is the input vector (with l length), $\boldsymbol{\beta}$ is the weights vector (with l length), s is a scale factor and b is a constant.

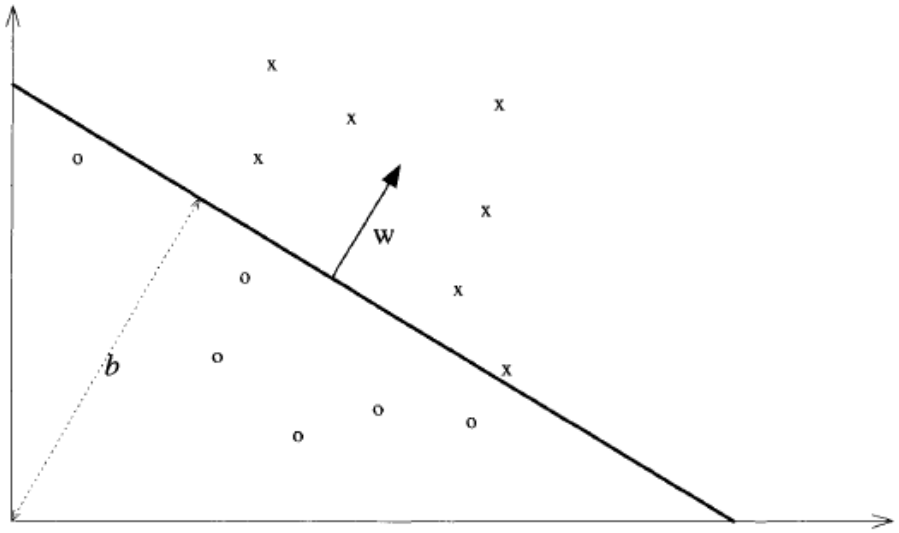


Figure 5. SVM for to classes (x and o), plane *w* dividing the two classes [37].

The (Eq. 3) can only be used to classify between two classes and in our problem, there are multiple classes. To classify between multiple classes a generalization of (Eq. 3) for several binary classifiers is done,

$$Y = \frac{X}{s} \times \beta + b \qquad \textbf{(Eq. 4)}$$

Now the **Y** is a vector with length n (number of binary) classifiers, $\boldsymbol{\beta}$ is a matrix of dimensions $l \times n$.

Given an input array **X** of samples it is multiplied by the matrix β/s obtaining a vector **Y** with the response for each classifier, in Figure 6 the basic algorithm to be implemented is shown in its matrix form.

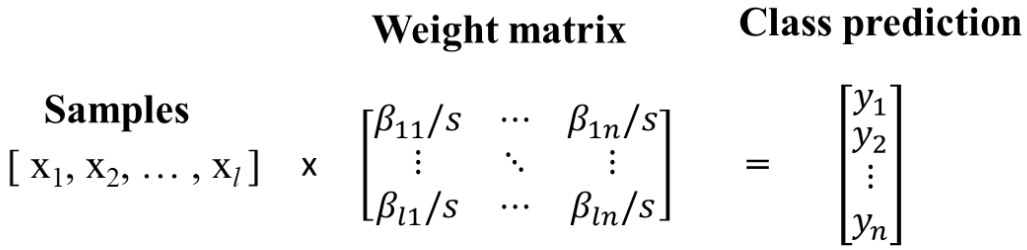


Figure 6. Matrix representation of the algorithm.

7. DESIGN FOR EMBEDDED SYSTEM

The objective in this design is to understand the relations between the design variables and the final size and execution time of the classifier. The approach is described in Figure 7. First data is acquired and converted to a usable MATLAB format then an iterative approach is taken to obtain a better model. This process is mean to be flexible and fast to test different classifiers and feature extraction.

A factorial design is used to understand the relations between the design variables (coding, sample length and datatype) and the impact in the response variables (memory footprint, processing time and precision of the algorithm). A factorial design consists of making all the possible combinations and adjusting a model for the response variables in function of the design variable based on the results of the combinations.

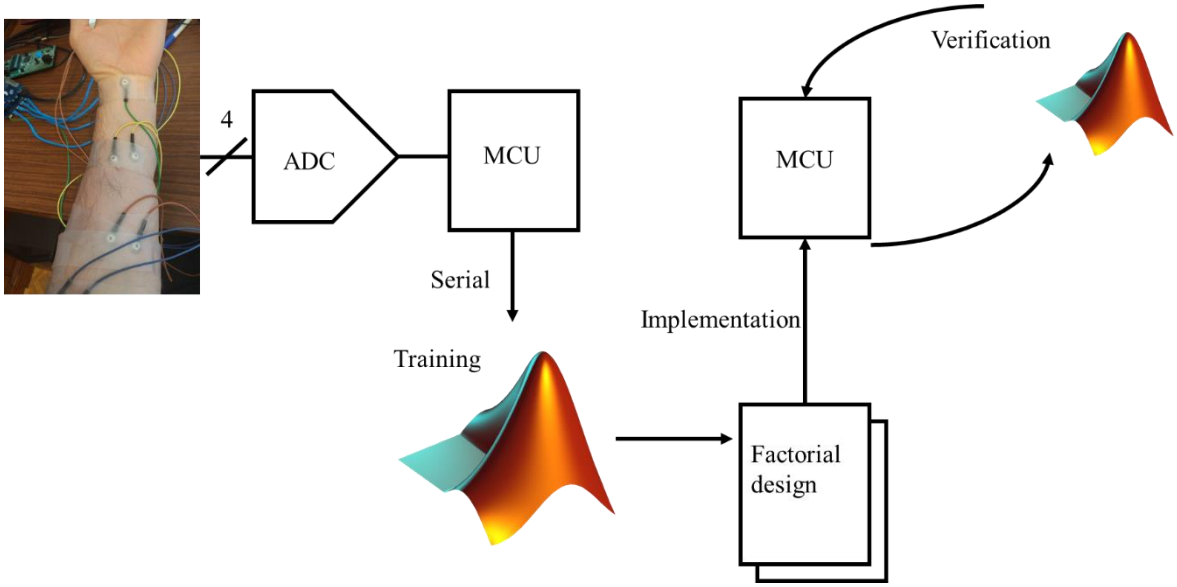


Figure 7. System approach to design a classifying algorithm for an embedded device.

7.1 Hardware architecture

To reduce memory footprint and processing time it is very important to know the hardware architecture of a low energy embedded system. The MCU used, ARM Cortex M4 [38], has floating point unit (FPU) It is also important to highlight that the Cortex M4 has a FPU of 32 bits (single float) and the function to add or multiply double float (float64) are implemented by software. According to tests to multiply or add a double takes a mean of 54-56 [39] cycles, this measurement is empirical and depends on the compiler and on the values of both operands. Note to mention that this architecture does not allow multiple instruction emission (parallelize integer operations with FPU coprocessor). Table 4 contains all the details of the architecture.

Table 4: Specifications of hardware and Cortex M4 instruction set.

RAM memory	128 KB
Flash memory	1 MB
Frequency	80 MHz
Load double float	3 cycles
Load single float	2 cycles
Multiply single float	1 cycle
Multiply double float	56 cycle
Add single float	1 cycle
Add double float	54 cycle
Store double float	3 cycles
Store single float	2 cycles

7.2 Design and response variables

The classifier is characterized with three response variables: Loss, cycles, and memory footprint.

- Loss, a function that estimates the error of the cross validated model. It ranges from 0 to 1 where lower is better.
- Cycles or execution time, number of cycles to predict a gesture, the lower is better, measured in cycles and seconds, respectively.
- Memory footprint, the space that is used by the classifier, the lower is better, measured in bytes.

With these three objective variables is easy to understand how the classifier is performing and the cost in memory and processing time that require.

The classifier has three design variables that were studied. Each of them has two levels. These three are: the variable type, the length of the sample and the coding of the SVM.

- The variable type can be either single float (32 bites) or double float (64 bites), this change is done in the cast of the ADC output where the sampled value once converted to volts can be stored in a single or a double.
- The length of the sample, in our case we are recording 3 seconds the variable can be set either to 3 seconds or to 1.5 seconds.
- The coding of the SVM, this means how the classifier is built. Two coding are used: “one vs one”, that means there is a binary classifier between each class and each other class (all possible combinations) and one vs all that means that there is just one binary classifier for each class against the rest of the classes.

Each of the objective function has been implemented to take into account the different changes in the design. This process has been done in MATLAB because it is easy to modify parameters in the classifier and to implement the functions to express the Cycles and the memory.

7.2.1 Loss function

The procedure for calculating the loss is shown in Figure 8:

1. The model is trained and optimized to minimize the error using fitcecoc() from the Statistics and Machine Learning Toolbox, a Bayesian optimizer is used to obtain the best *KernelScale* and *BoxConstraint* for the classifier.

```
fitcecoc(X,Y,'OptimizeHyperparameters',{'KernelScale','BoxConstraint'});
```

2. Cross-validate the model, which consists of training the model with nine tenth of the data then predict the one tenth left, this procedure is repeated ten times. With this method it is possible to evaluate the error of the model. The cross validation is done by the function crossval():

```
crossval(Model);
```

3. Estimate the error using the function kfoldLoss() which estimates the error in the crossvalidated model.

```
kfoldLoss(Model);
```

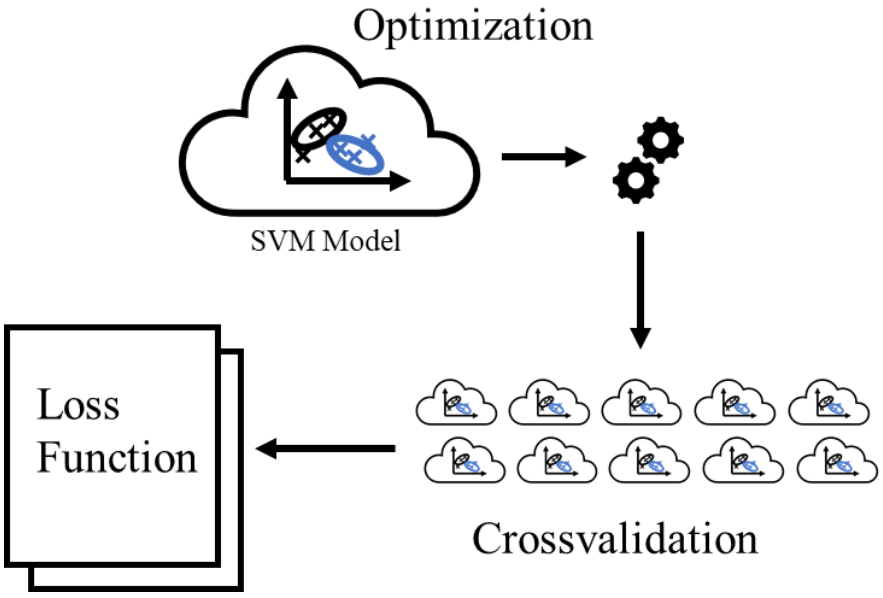


Figure 8. Procedure for evaluating the loss function. For further details on cross validation scheme see the text above.

7.2.2 Cycles function and computation time

The cycles function (code can be reviewed in the code appendix) estimates the number of the CPU cycles that are required to predict a gesture. To calculate the computation cost we take into account that the each binary classifier of the SVM has the following equation (Eq. 5) [37]:

$$y = \frac{X}{s} x\, \beta + b \qquad \textbf{(Eq. 5)}$$

Where y is the classifier result, X is the input vector, s is the is the scale, β is the vector with the weights of the predictors and b the bias of the model. Note that x is a dot multiplication between two vectors.

As mentioned in the previous subchapter the coding type determines the total number of binary classifiers. The formulas for the "one vs one" coding and one vs all coding are :

$$one\ vs\ one: classifiers = \frac{K \cdot (K-1)}{2}$$
$$one\ vs\ all: classifiers = K \quad \textbf{(Eq. 6)}$$

Where K is the number of classes in the whole classifier (8 in our case).

Then the total number of operations for each type are:

$$load = l \cdot (classifiers \cdot K + 1)$$
$$store = K \cdot classifiers$$
$$multiply = l \cdot K \cdot classifiers \quad \textbf{(Eq. 7)}$$
$$add = 2 \cdot l \cdot K$$

Where K is the number of classes (gestures) and l is the input sample vector.

The parameter l can be calculated as follows.

$$l = t_{acq} \cdot f_s \cdot \mathrm{n_{ch}} \quad \textbf{(Eq. 8)}$$

Where t_{acq} is the time for each gesture, f_s is the sampling frequency (500 Hz) and n_{ch} the number of channels (4 in this application).

Error! Reference source not found.It is possible to calculate the number of cycles that takes to do the required operations. Note that the control instructions are not consider because they are a small amount compared to the multiply, add and load functions and the estimation would have been much more complex.

For calculating the computation time, the resulting cycles are divided by the CPU frequency, this metric is used because it's more friendly to work with. For the shake of this approximation the frequency taken is 80 MHz.

7.2.3 Memory footprint function

To predict is required to save the value of $\boldsymbol{\beta}$ (Eq. 5) for each binary classifier. With this consideration and the (Eq. 6) is obtained.

$$mem(bytes) = \frac{size_{var}}{8} \cdot classifiers \cdot l \cdot K \quad \textbf{(Eq. 9)}$$

Where $size_{var}$ is the size of the type variable used (float32 or float64), classifier is the number of binary classifiers from (Eq. 6), l is the length of the input vector (Eq. 8) and K is the number of gestures (8 in our application).

Note that to take in account the full memory footprint the space required by the sampling buffer should be considered. The buffer was not included because it depends on the implementation and always be proportional to memory footprint of the classifier, so the relative improvements in memory footprint are valid.

The function memoryfootprint() written in MATLAB is available in the code appendix.

8. METHODOLOGY

In this chapter is presented the different methodologies followed in the processing of the dataset and the acquisition of data.

8.1 Analysis of gestures

The workflow, Figure 9, used to analyze the gestures and the underlying structure of data is carried out using MATLAB. It takes three steps to obtain a PARAFAC model to work with.

First the data was picked from the NINAPRO database [15] and organized in a more comfortable format, using the dataExtraction script, see code appendix. This script asks the user to choose the subject, day, date and daytime, then performs a DFT of the samples. Once the frequency spectrum is obtained(DFT), parafac() [40] is applied to obtain the PARAFAC model.

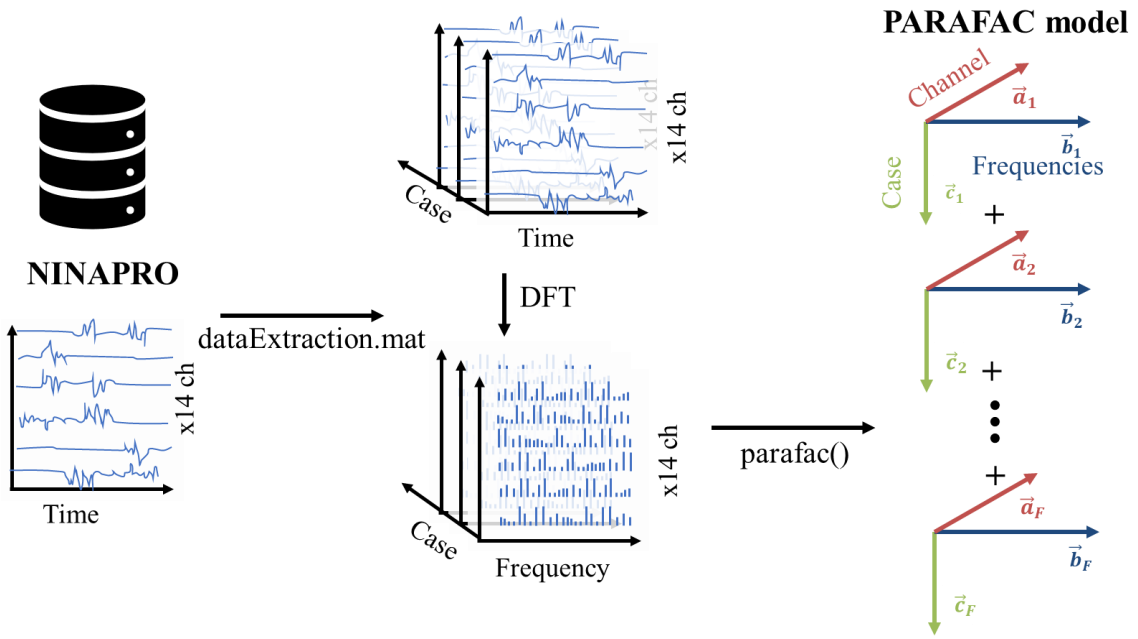


Figure 9. Workflow for obtaining the data from NINAPRO and processing it to obtain its PARAFAC model.

Because the number of original variables is larger than three day, subject and daytime was encoded in a single variable called case. The codification is shown in Table 5. Four gestures were selected, two subjects, two days and two different times of the day. Each of the selected gesture has 6 replicas.

Table 5. Codification of the cases for PARAFAC.

Case	Subject	Day	Time of the day	Gesture
1-6	1	1	2	1
7-12	10	1	2	2
13-18	10	5	2	3
19-24	1	5	1	3
25-30	10	1	1	1
31-36	1	5	2	4
47-42	1	1	1	2
43-48	10	5	1	4

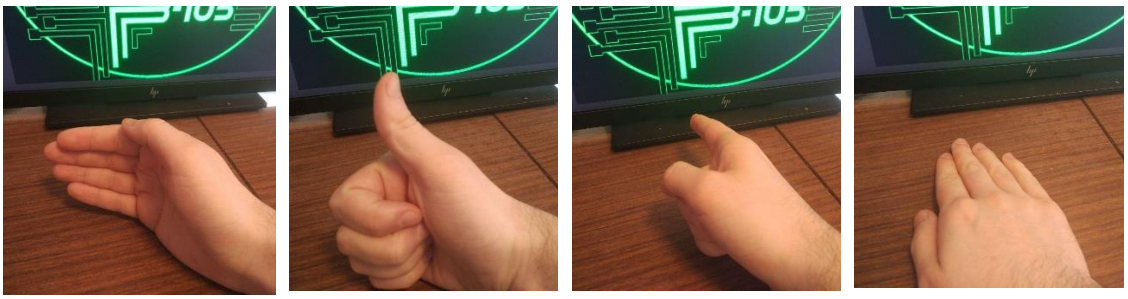
8.2 Acquisition with embedded system

The acquisition was made by the system described in Figure 1 with four pair of electrodes connected to 4 channel differential amplifier with unity gain at 500 Hz sample rate.

8.2.1 Gestures analyzed

The dataset recorded is composed of 8 different gestures, recorded during 3 seconds by the 4 channels. The samples are labeled with the number of the subject the time of the day and the day of testing that they were recorded. The recorded data file has the following attributes and the data was stored in a .mat file (MATLAB 2019b version).

The gestures done by the subject are shown in Figure 10. These gestures are a selection of the twenty recorded in work [16], which are useful for prothesis control because they involve fine operations with different types of grasp and flexing/contracting individual fingers.



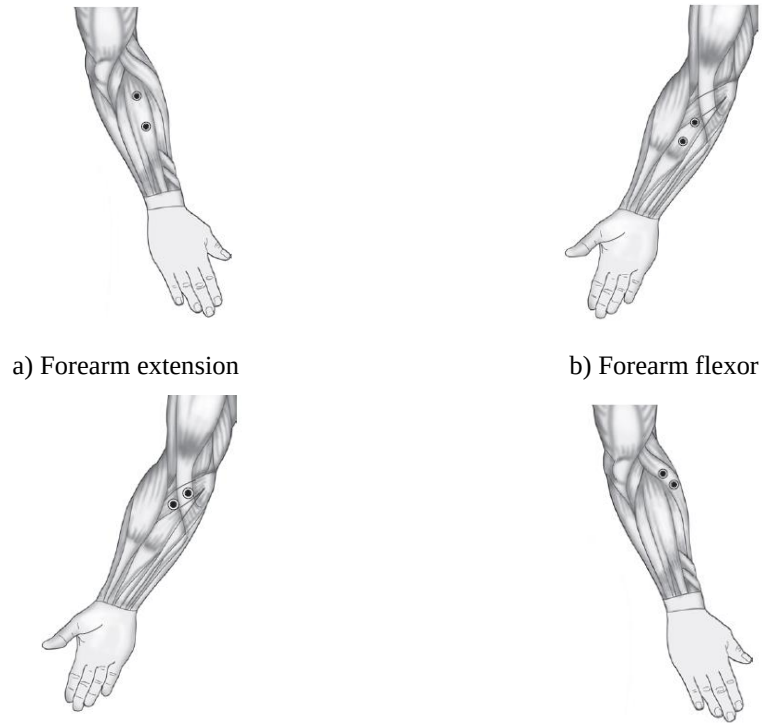
a) Rest position | b) Thumb extension | c) Index extension | d)Wrist extension

d) Wrist contraction | e) Circular grasp | f) Clamp grasp | g) Precision tripod grasp

Figure 10. Different gestures recorded, with different type of grasp and hand positions.

8.2.2 Placement of electrodes and acquisition routine

The electrodes are placed where the biggest muscular groups are located [41]. In Figure 11 , the placement is described. The electrodes pair *a* and *b* are related with the movement of the wrist. The placement *c* is related with the finger extension (index and thumb). The grasp movements are complex, and several muscles are involved. The reference electrode is located on the wrist because there are no muscles underneath.



a) Forearm extension b) Forearm flexor

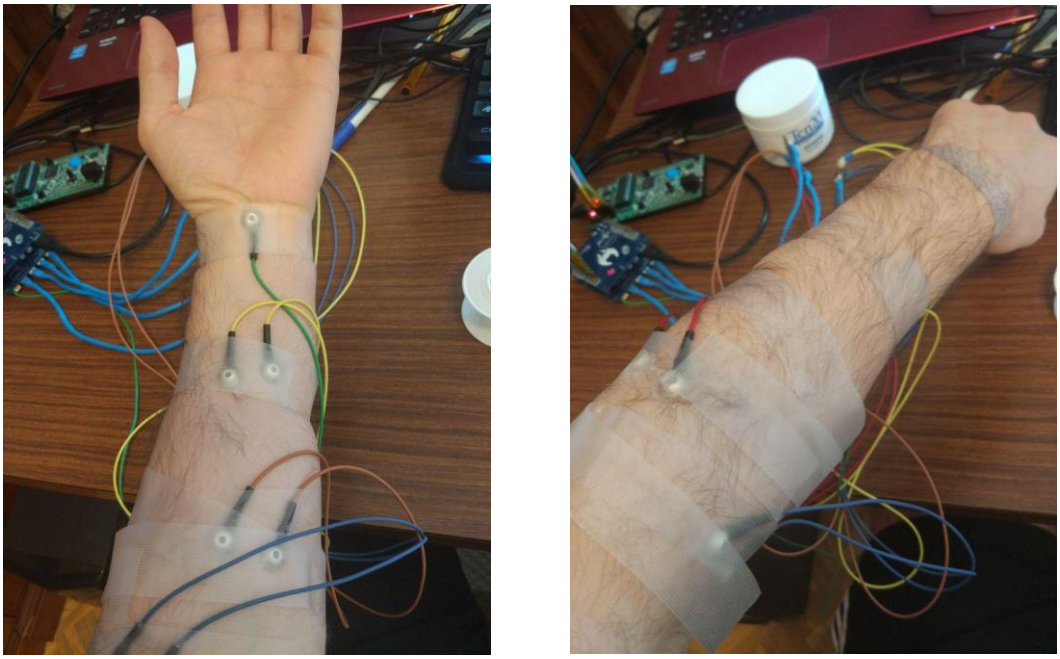
c) Pronator teres series d) Extensor carpi radialis

Figure 11. Placement of electrodes [41].

Following Figure 11, the electrodes are attached to the subject. It is very important to mark the skin of the subject with a pen. This allows to set the electrodes in the same position in the next session. The steps for placing the electrodes are:

1. Clean with alcohol the skin of the subject.
2. Apply conductive paste to the electrodes and to the skin.
3. Place the electrodes firmly against the skin of the subject.
4. Fix the electrodes with adhesive tape.

The placement on the subject is shown in Figure 12.



a) Internal forearm b) External forearm

Figure 12. Placement of the electrodes in the subject.

The subject first is shown the gestures, with a photo and asked to do it. The subject is told to do the gestures with the minimum strength possible. The supervisor checks that the gestures are performed correctly and then the recording process starts. The acquisition device is connected by a serial over USB to the computer. At the computer, the samples are recorded by RealTerm and saved in a file. This procedure is repeated ten times for each gesture and session.

8.3 Factorial design

To understand the influence of the design variables a two-level three-factor full factorial design was used. A factorial design was chosen because it models the response variables using a linear equation (Eq. 10) by making all the possible combinations. The list of the three variables: coding, sample length and data type, are named A, B C respectively and the encoding and values for each level is shown in the following table.

Table 6. Variables and levels for the two-level and three factor full factorial design.

Variable	Symbol	Real values of coded level	
		-1	1
Coding	A	“One vs one” (ovo)	“One vs all” (ova)
Sample length	B	3 s	1.5 s
Data type	C	Float_64	Float_32

A first order polynomial was fit to the experimental results. This model represents the effects of the design variables (A, B, C) and their interactions on the response variables (loss, memory footprint and processing time). The model is represented can be expressed by the following equation.

$$Y = b_0 + b_1A + b_2B + b_3C + b_{12}AB + b_{13}AC + b_{23}BC + b_{123}ABC \quad \textbf{(Eq. 10)}$$

Where Y is the response variable; b_0 is the model constant; b_0,b_1,b_2 are the linear coefficients; $b_{12}, b_{13}, b_{23}, b_{123}$ are the cross product coefficients, that represent the interaction between the design variables. The model has been built to have a unique solution, due to the codification of the variables. It can be resolved by any algebraical method.

9. RESULTS

In this chapter the results of the gesture analysis, classification and factorial design are presented and discussed.

9.1 Gesture analysis

The gestures from the dataset were selected and coded as mentioned in Table 4. A PARAFAC model of the 14 channels and one of 4 selected channels were done. Two components were used because when going from two components to three the CONCORDIA value dropped which means that model is overfitted and it does not reflect accurately the data. The three factors of the model are: frequency, channel and case.

With two factors and 14 channels the CONCORDIA value is 100% which means that the hypothesis that data is trilinear is fulfilled. In Figure 13, it can be observed that the first component (blue) presents differences with the second one up to 250 Hz. This means that the information that is more significant and unique is in that range (0-250 Hz). There is no relevant weight in the 50Hz frequency which means that the power line noise has little impact.

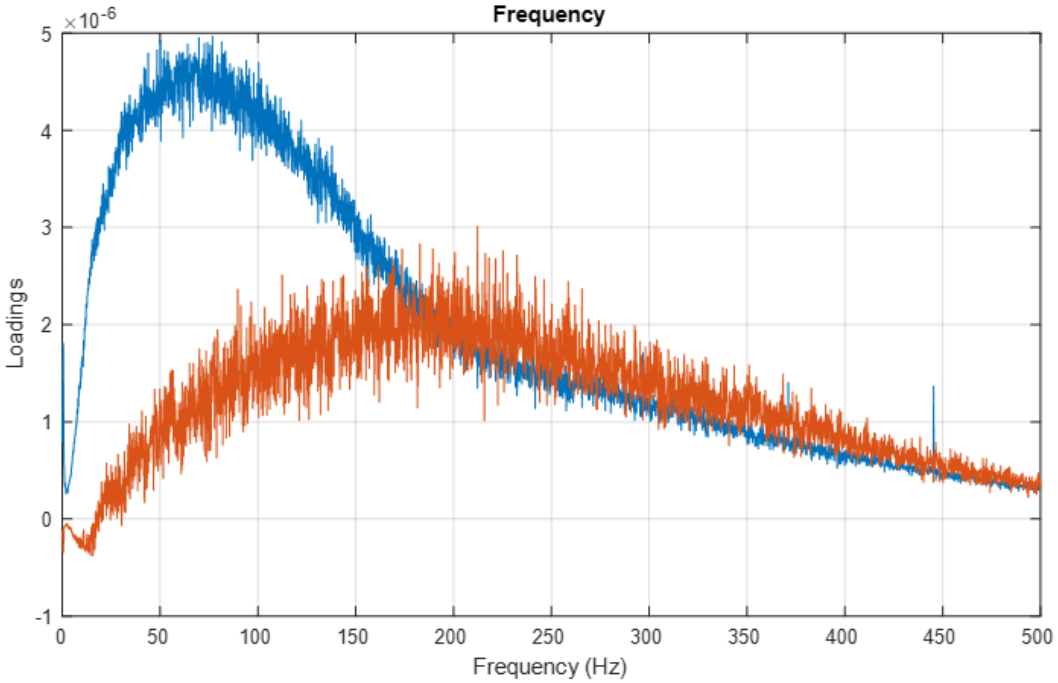


Figure 13. Frequency factor loadings (proportional to the voltage recorded in the sample) for 14 channels PARAFAC model. Blue is the first component and red is the second one.

The second factor is the channel, Figure 14. From the analysis of the result it is observable that there is a channel that has almost no weight in the model (channel 8). And that lots of channels (3, 4, 5, 6, 12) have the same loading, this suggests that deleting some of those does not imply a lost in the information underlying. Lastly there are some channels (1, 2, 9 or 10) that have notoriously different values, this is a sign on what channels to choose for obtaining better results.

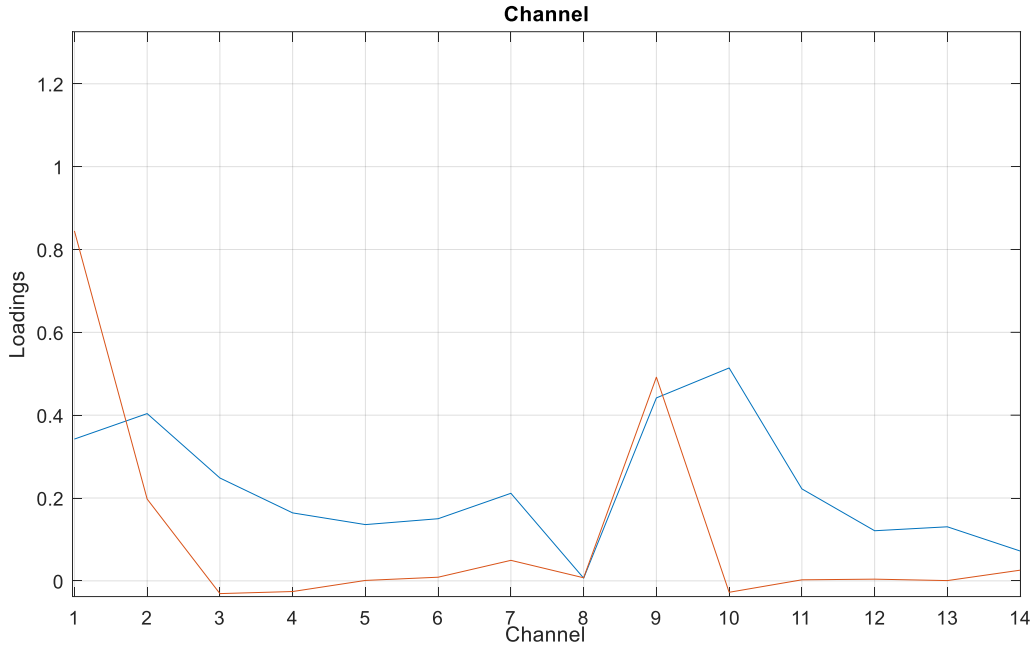


Figure 14. Channel factor loadings (proportional to the voltage recorded in the sample) for 14 channels PARAFAC model. Blue is first component and red is the second one.

The last factor is the case, and each gesture has six samples, according to Table 5. In Figure 15 the gestures are labeled on top. There is no clear distinction between gestures, 2 can be confused with 3. Furthermore, changing the subject (first gesture 1 or 4 to second gesture 1 or 4) changes completely the loadings. This confirms that the data is highly dependent on the subject.

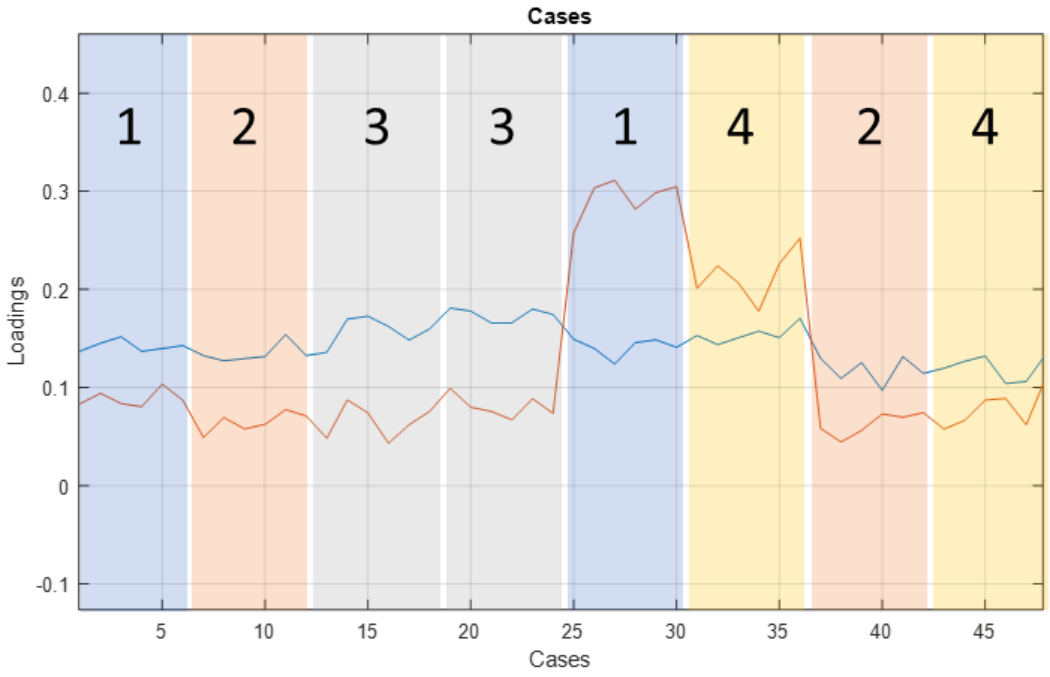


Figure 15. Cases factor loadings for 14 channels (1, 2, 9, 10) PARAFAC model. Blue line is first component, red line second one. The gestures are labeled on top.

The following PARAFAC model was done selecting 4 channels (1, 2, 9, 10) out of the 14, this channels were selected between the most relevant in Figure 14. The frequency

factor for the 4 channel PARAFAC, Figure 16, is almost identical to the one obtained with 14 channels, which means that the relevant frequencies are the same.

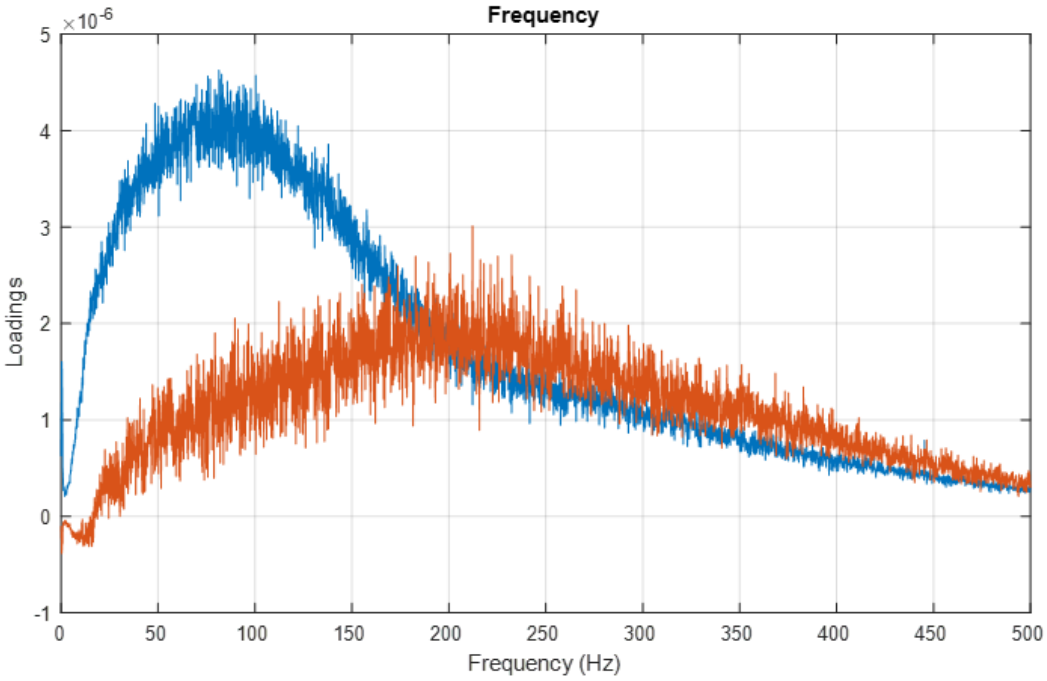


Figure 16. Frequency factor loadings (proportional to the voltage recorded in the sample) for 4 channels (1, 2, 9, 10) PARAFAC model. Blue is the first component and red is the second one

The channel factor, Figure 17, now only has 4 channels and it is important to note that the first component weight of all the 4 channels is very similar and they differ in the second component. This could be a sign that the differences in the information that carries each channel is located in the second component. The peak of the second component frequency is around 250 Hz and could be a sign of important information in that frequency range.

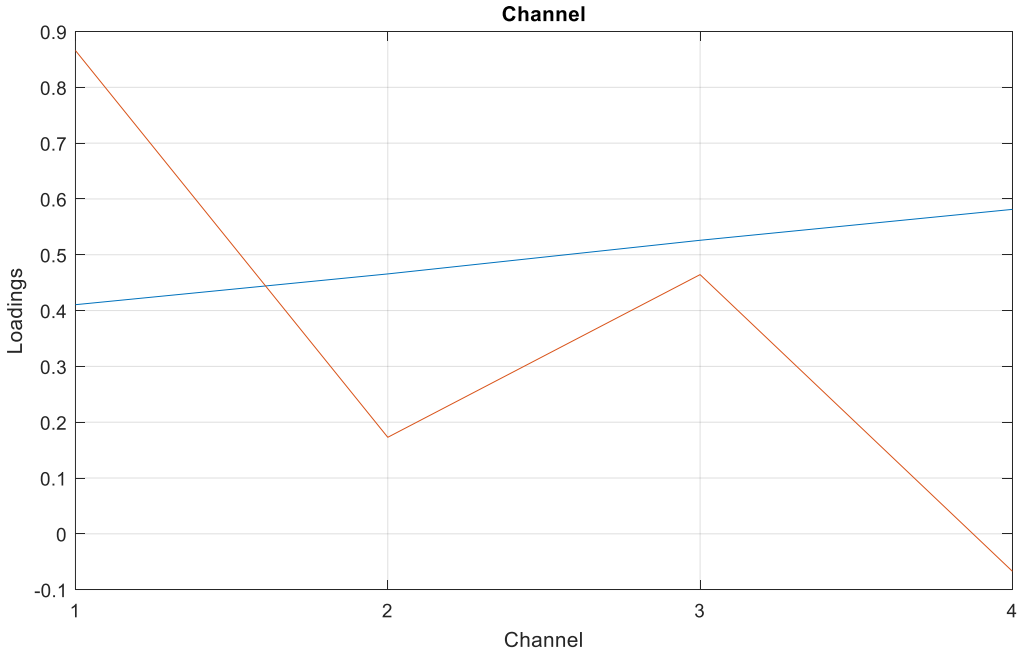


Figure 17. Channel factor loadings (proportional to the voltage recorded in the sample) for 4 channels (1, 2, 9, 10) PARAFAC model. Blue is first component and red is the second one.

Last factor, the cases, Figure 18, shows a great improvement from its 14 channel counterpart, Figure 15. Now gestures 3 and 2 have a unique pattern that allows it easy identification. However, gestures 4 and 1 are still mixed and hard to set apart.

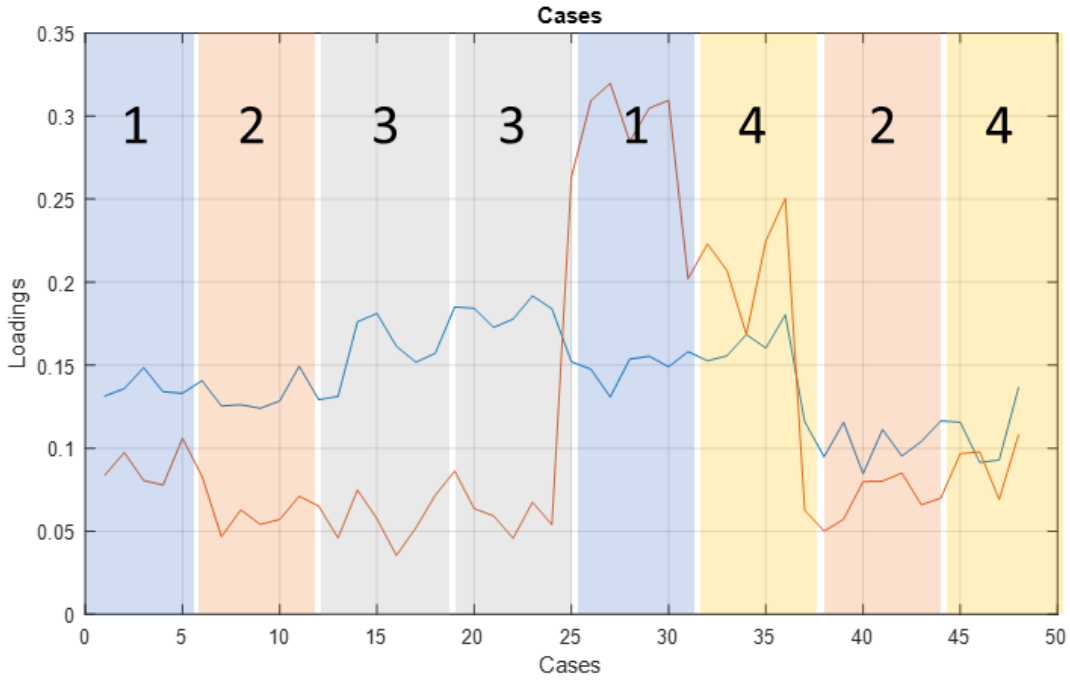


Figure 18. Cases factor loadings (proportional to the voltage recorded in the sample) for 4 channels (1, 2, 9, 10) PARAFAC model. Blue line is first component, red line second one. The gestures are labeled on top.

The last PARAFAC model was done using another 4 different channels (1, 3, 5, 7), these were choosen because channel 3,5 and 7 have similar loadings according to Figure 14 .The first factor, Figure 19, differ greatly from the pattern observed in the two previous model. This is a sign that information has been lost and the mentioned channels (3, 5 and 7) provide similar data.

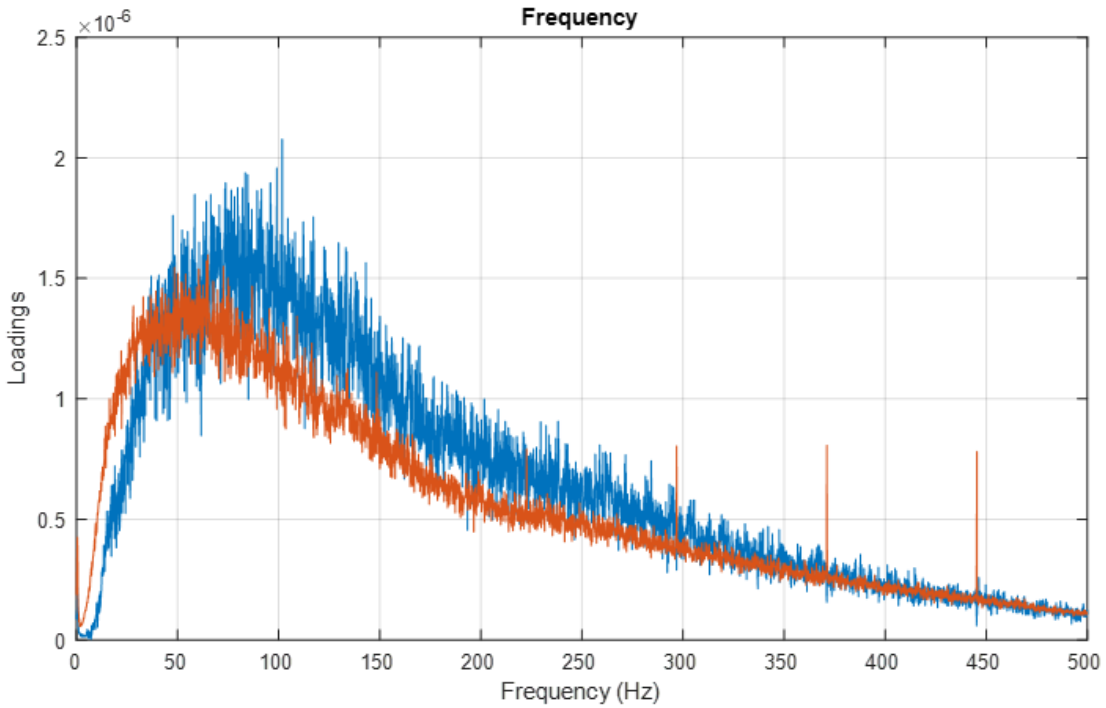


Figure 19. Frequency factor loadings (proportional to the voltage recorded in the sample) for 4 channels (1, 3, 5, 7) PARAFAC model. Blue is the first component and red is the second one

The channel factor, Figure 20, shows a drop in the first factor being close to zero in 3 of the four channels. A significant lost of information can be deduced.

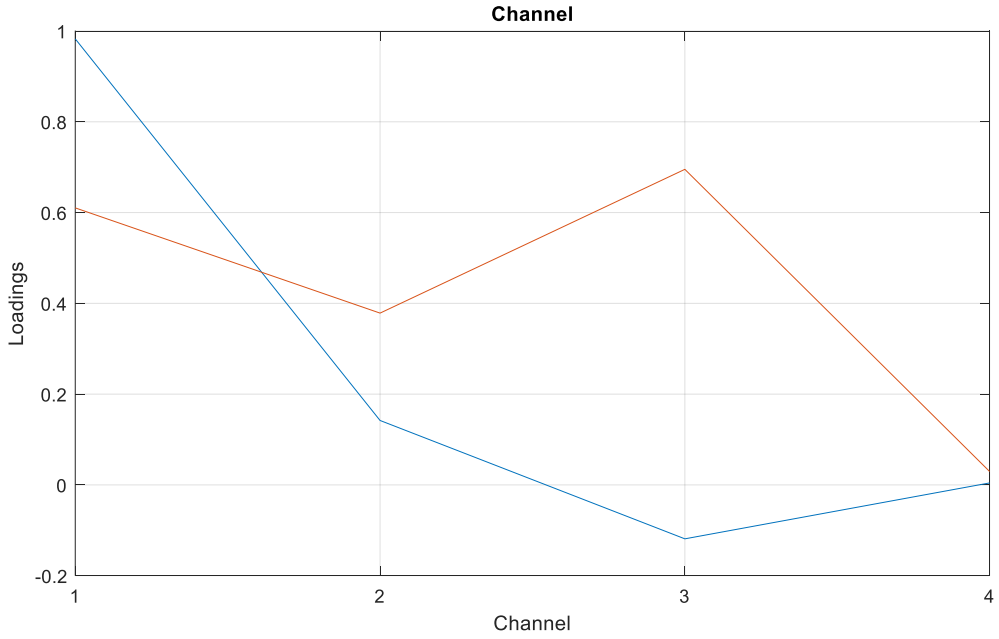


Figure 20. Channel factor loadings (proportional to the voltage recorded in the sample) for 4 channels (1, 3, 5, 7) PARAFAC model. Blue is first component and red is the second one.

Last factor of the PARAFAC model, Figure 21, confirms that now is impossible to tell apart any of the gestures.

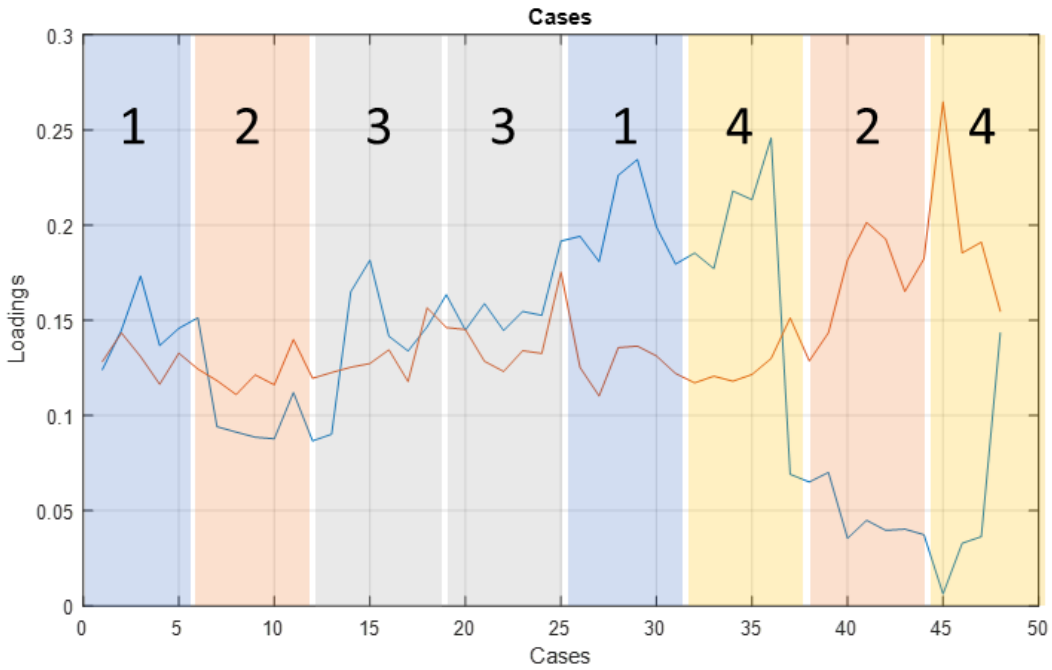


Figure 21. Cases factor loadings (proportional to the voltage recorded in the sample) for 4 channels (1, 3, 5, 7) PARAFAC model. Blue line is first component, red line second one. The gestures are labeled on top.

9.2 Classification

The trained SVM model with the acquired samples yield good results in cross validation achieving a 99% of accuracy, with the confusion matrix shown in Figure 22.a). Only one sample out of one hundred was misclassified. The memory footprint was 11 MB and the processing time 3 seconds, Table 7. These results are far from meeting the requirements of an embedded system memory available or the real-time classification that were aimed for. While performing the classification of the registered samples it was found that channel 2 is not working, however it was decided to continue using it because it did not affect the precision and made the following estimations of memory and execution time more accurate to the original design of 4 channels.

9.3 Factorial design

A total of 8 different models (all possible combinations of the three design variables) were made and evaluated. The results are exposed in the following table.

Table 7. Observed values for the response variables.

Run	Conditions			Loss	Processing time (s)	Memory footprint (KB)
	A	B	C			
	Coding	Sample Length (s)	Data type			
1	ovo	3	float_64	0.0125	3.0302	11612
2	ovo	3	float_32	0.0125	0.0908	5806
3	ovo	1.5	float_64	0.025	1.5151	5806
4	ovo	1.5	float_32	0.025	0.0454	2903
5	ova	3	float_64	0.1125	0.8659	3317
6	ova	3	float_32	0.1125	0.0260	1658
7	ova	1.5	float_64	0.1625	0.4329	1658
8	ova	1.5	float_32	0.15	0.0130	829

With these values we can solve the equation system to obtain the coefficients values for each response variable (in the three of them the less the better).

$$loss = 0.0766 + 0.0578A + 0.0141B - 0.0016C + 0.0078AB - 0.0016AC - 0.0016BC - 0.0016ABC$$

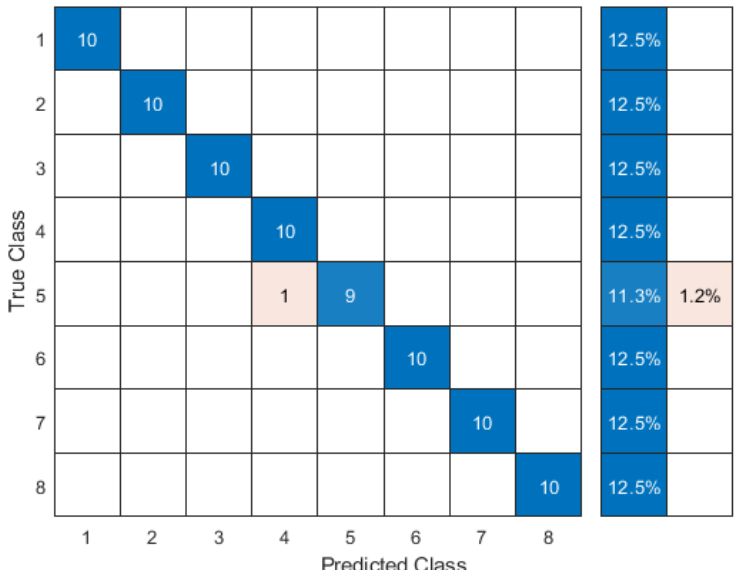
$$time(s) = 0.7525 - 0.4180A - 0.2508B - 0.7086C + 0.1393AB + 0.3937AC + 0.2362BC - 0.1312ABC \quad \textbf{(Eq. 11)}$$

$$mem(KB) = 4199 - 2333A - 1400B - 1400C + 778AB + 778AC + 467BC - 259ABC$$

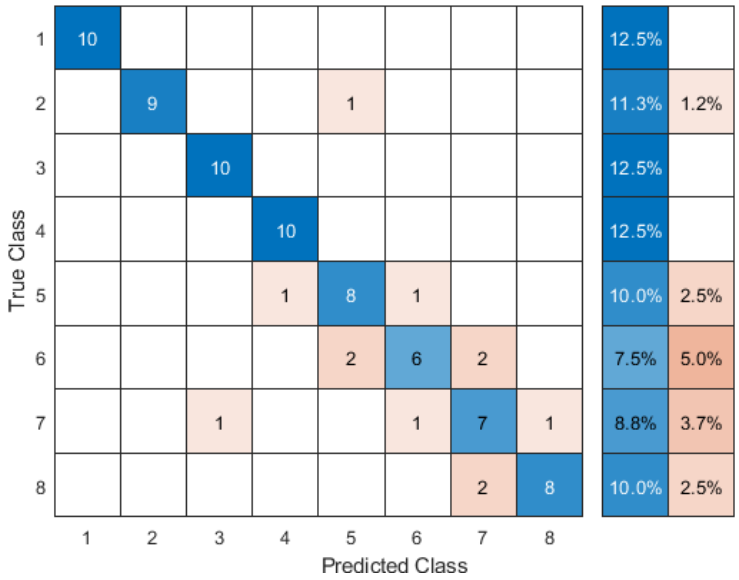
For better understanding of how the classifier is performing the confusion matrix for the run 1 (base case) , run 8(case with the greater reduction in memory size and computation time) and run 6 (same case as 8 but with 3 instead of 1.5 seconds sample length.) are provided in the Figure 22.

The confusion matrix, Figure 22, represents the predicted class of a sample versus its true class in the cross validated model. The best is that all samples match its true class. It can be observed that lowering the memory footprint and processing time (Run 8) impact negatively in how good the classes are predicted.

a)



b)



c)

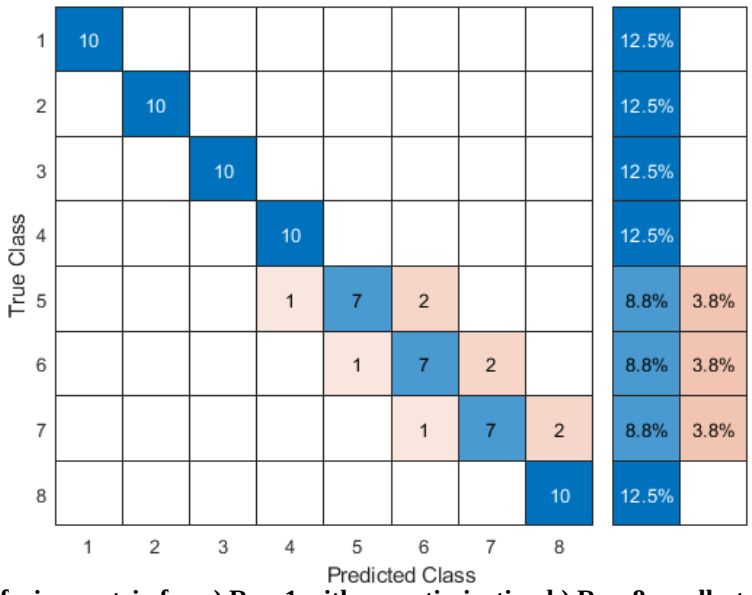


Figure 22. Confusion matrix for a) Run 1 with no optimization b) Run 8 smallest processing time and memory footprint c) Run 6 with intermediate optimization.

From the (Eq. 11) it can be observed that the change in the data type (C) has negligible impact on the loss, the coding (A) used has greater impact that the sample length (B). The interactions between the variables are trivial because they are at least one order less.

The time response variable is greatly impacted by the data type (C), so using float64 is highly discouraged. This observation make sense with the fact that the FPU of the Cortex M4 is float32. Fixing the variable data type to float32 (coded as -1) proves also show that the interactions between AC and BC are significant. This means that, changing the coding to one vs all or changing the acquisition time from 3 to 1.5 seconds just slightly improves the processing time.

The memory footprint is more impacted by the change in the coding than from the change in the sample length or the variable type change.

To conclude, changing from float_64 to float_32 has only advantages, less time and memory footprint and no drawback, same loss. The coding increased the loss but also significantly reduce the memory footprint. The sample length has lower impact in both improving time and memory footprint but also affects less to the loss of the classifier.

The fastest and lowest memory footprint is the one achieved by using float32_t, 1.5 s of sampling time and one vs all codification. The results are a memory footprint of 868 KB with an execution time of 130 ms. The results show that the limiting variable is the memory because it is almost all the flash memory that the MCU used provides. The precision of this classifier is 85%.

10. CONCLUSION AND FUTURE WORK

The PARAFAC decomposition showed that EMG signals are complex. The gesture information is highly dependent on the subject and varies within days. According to the analysis carried out the interest frequencies are located under 300 Hz. It was also proved that is possible to reduce the number of channels from 14 to 4 without significant information loss.

An acquisition system was developed using a low energy MCU ARM Cortex M4, the data was successfully logged and sent by serial. The acquisition system is flexible and modular. It was designed for its reutilization in future works.

Eight gestures were recorded, labeled, and classified by an SVM classifier. The design SVM classifier was trained with the data acquired obtaining a 99% precision, 1 misclassification out of 100 samples. It was demonstrated that a SVM classifier with no feature extraction is capable of making a good prediction.

A reduction of 1/14 of memory footprint and 233x speedup in the execution time for the classifier was achieved. The lowest memory footprint and execution time achieved was 868 KB and 130 ms respectively. This classifier has an 85% precision. The factorial design carried out allowed to conclude that the use of float64 as datatype yields no benefit. It also proved that there is a tradeoff between the precision and the memory footprint and processing time, and coding of the classifier is the most relevant in memory footprint and execution time reduction. Memory footprint was identified as the bottleneck for an implementation

This works opens the possibility for a real-time memory and energy constrained device capable of identifying the gestures made by the user using EMG signals.

10.1 Future work

From the results of this work the following next steps should be taken.

- Reduce the active channels by developing a method to select the best ones. This could lead to an acquisition platform with several channels but only a few of them being used simultaneously, multiplexing them, and choosing the best ones. This approach could probably reduce the memory and processing required without a drawback in the precision of the classifier.
- Develop and implement an optimized C version of the classifier that improves its performance.
- Record a complete dataset of gestures, with different subjects and days to measure the impact on day and subject variation in the classifier. Allowing the use of PARAFAC to analyze the data acquired and enhance the classifier by knowing better where the information is.
- Reduce the sampling frequency of the ADC to reduce the size of the classifier.
- Explore other classifiers or feature extraction techniques aiming to decrease the memory footprint and execution time of the classifier.

11. REFERENCES

[1] Q. Meng, J. Zhang y X. Yang, «Virtual Rehabilitation Training System Based on Surface EMG Extraction and Analysis,» *Journal of Medical Systems,* vol. 43, nº 48, 2019.

[2] F. Sadikoglu, C. Kavalcioglu y B. Dagman, «Electromyogram (EMG) signal detection, classification of EMG signals and diagnosis of neuropathy muscle disease,» de *9th International Conference on Theory and Application of Soft Computing, Computing with*, Budapest, 2017.

[3] S. Beniczky, I. Conradsen, O. Henning, M. Fabricius y P. Wolf, «Automated real-time detection of tonic-clonic seizures using a wearable EMG device,» *Neurology,* vol. 90, nº 5, 2018.

[4] G. Biagetti, P. Crippa, L. Falaschetti, S. Luzzi y C. Turchetti, «Recognition of Daily Human Activities Using Accelerometer and sEMG Signals,» de *Intelligent Decision Technologies 2019. Smart Innovation, Systems and Technologies*, vol. 143, Springer, Singapore, 2019, pp. 37-47.

[5] S. Benatti, B. Milosevic, E. Farella, E. Gruppioni y L. Benini, «A Prosthetic Hand Body Area Controller Based on Efficient Pattern Recognition Control Strategies,» *Sensors,* vol. 17, nº 869, 2017.

[6] B. C. Johnson, S. Gambini, I. Izyumin, A. Moin, A. Zhou, G. Alexandrov, S. R. Santacruz, J. M. Rabaeyh, J. M. Carmena y R. Muller, «An implantable 700uW 64-channel neuromodulation IC for simultaneous recording and stimulation with rapid artifact recovery.,» de *Symposium on VLSI Circuits* , 2017.

[7] A. Moin, A. Zhou, A. Rahimi, S. Benatti, A. Menon, S. Tamakloe, J. Ting, N. Yamamoto, Y. Khan, F. Burghardt, L. Benini, A. C. Arias y J. M. Rabaey, «An EMG Gesture Recognition System with Flexible High-Density Sensors and Brain-Inspired High-Dimensional Classifier,,» de *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, Florence, 2018.

[8] A. Zhou, S. R. Santacruz, b. C. Johnson, G. Alexandrov , A. Moin, F. L. Burghardt, J. M. Rabaey, J. M. Carmena y R. Muller, «A wireless and artefact-free 128-channel neuromodulation device for closed-loop stimulation and recording in non-human primates,» *Nat Biomed Eng,* vol. 3, pp. 15-26, 2019.

[9] A. Baraka , H. Shaban, M. Abou El-Nasr y O. Attallah, «Wearable Accelerometer and sEMG-Based Upper Limb BSN for Tele-Rehabilitation,» *Applied Sciences,* vol. 9, nº 2795, 2019.

[10] N. Nazmi, M. Azizi Abdul Rahman, S.-I. Yamamoto, S. Anom Ahmad, H. Zamzuri y S. Amri Mazlan, «A Review of Classification Techniques of Signals during Isotonic and Isometric Contractions,» *Sensors,* vol. 16, nº 1304, 2016.

[11] R. Merletti y P. Parker, «Basic Physiology and Biophysics of EMG Signal Generation,» de *Electromyography: Physiology, Engineering, and Non-Invasive Applications*, IEEE, 2004, pp. 1-25.

[12] A. Phinyomark and E. Scheme, "A feature extraction issue for myoelectric control based on wearable EMG sensors," in *2018 IEEE Sensors Applications Symposium (SAS)*, Seoul, 2018.

[13] R. H. Chowdhury, M. B. I. Reaz, M. Alauddin Bin Mohd Ali, A. A. A. Bakar, K. Chellappan y T. G. Chang, «Surface Electromyography Signal Processing and Classification Techniques,» *Sensors,* vol. 13, pp. 12431-12466, 2013.

[14] M. Tomasini, S. Benatti, B. Milosevic , E. Farella y L. Benini, «Power Line Interference Removal for High-Quality Continuous Biosignal Monitoring With Low-Power Wearable Devices,» *IEEE Sensors Journal,* vol. 16, nº 10, pp. 3887-3895, 2016.

[15] M. Atzori, A. Gijsberts, S. Heynen, A.-G. Mittaz Hager, O. Deriaz, P. van der Smagt, C. Castellini, B. Caputo y H. Müller, «Building the NINAPRO Database: A Resource for the Biorobotics Community,» de *2012 IEEE International Conference on Biomedical Robotics and Biomechatronics*, Rome, 2012.

[16] N. Rabin, M. Kahlon, S. Mayalev y A. Ratnovsky, «Classification of human hand movements based on EMG signals using nonlinear dimensionality reduction and data fusion techniques,» *Expert Systems With Applications,* vol. 149, nº 113281, 2020.

[17] S. Benatti, G. Rovere, J. Bosser, F. Montagna, E. Farella, F. Glasser, P. Schönle, T. Burger, S. Fateh, Q. Huang y L. Benini, «A sub-10mW real-time implementation for EMG hand gesture recognition based on a multi-core biomedical SoC,» de *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*, Vieste, 2017.

[18] S. De la Peña, A. Polo y C. Robles-Algarín, «Implementation of a Portable Electromyographic Prototype for the Detection of Muscle Fatigue,» *Electronics,* vol. 8, nº 619, 2019.

[19] A. Phinyomark, F. Quaine, S. Charbonnier, C. Serviere, F. Tarpin-Bernard y Y. Laurillau, «EMG feature evaluation for improving myoelectric pattern recognition robustness,» *Expert systems with applications,* vol. 40, nº 12, pp. 4832-4840, 2013.

[20] A. Dellacasa Bellingegni, E. Gruppioni, G. Colazzo, . A. Davalli, R. Sacchetti, E. Guglielmelli y L. Zollo, «NLR, MLP, SVM, and LDA: a comparative analysis on EMG data from people with trans-radial amputation.,» *Journal of NeuroEngineering and Rehabilitation ,* vol. 14, nº 82, 2017.

[21] R. O. Duda, P. E. Hart y D. G. Stork, Pattern classification, 2nd ed., Wiley, 2001.

[22] S. Escalera, O. Pujol y P. Radeva, «On the Decoding Process in Ternary Error-Correcting Output Codes,» *IEEE Transactions on Pattern Analysis and Machine Intelligence ,* vol. 32, nº 1, pp. 120-134, 2010.

[23] S. Bennati, F. Casamassima, B. Milosevic, E. Farella, P. Schöndle, S. Fateh, T. Burguer, Q. Huang y L. Benini, «A Versatile Embedded Platform for EMG Acquisition and Gesture Recognition,» *IEEE TRANSACTIONS ON BIOMEDICAL CIRCUITS AND SYSTEMS,* vol. 9, nº 5, pp. 620-630, 2015.

[24] G. Gini, L. Mazzon, S. Pontiggia y P. Belluco, «A Classifier of Shoulder Movements for a Wearable EMG-Based Device,» *Journal of Medical Robotics Research,* vol. 2, nº 2, 2017.

[25] P. Schönle, G. Rovere, F. Glaser, N. Brun, X. Han, T. Burguer, S. Fateg, Q. Wang, L. Benini y Q. Huang, «A Multi-Sensor and Parallel Processing SoC for Wearable and Implantable Telemetry Systems,» de *43rd IEEE European Solid State Circuits Conference*, Leuven, 2017.

[26] R. Rodriguez-Zurrunero, F. Tirado-Andrés y A. Araujo, «YetiOS: an Adaptive Operating System for Wireless Sensor Networks,» de *Thirteenth IEEE Workshop on Practical Issues in Building Sensor Network Applications 2018*, 2018.

[27] STMicroelectronics, STM32L486xx Ultra-low-power Arm® Cortex®-M4 32-bit MCU+FPU, 100DMIPS, 1MB Flash, 128KB SRAM, USB OTG FS, LCD, ext. SMPS, AES, STMicroelectronics, 2018.

[28] Analog Devices, AD7124-4: 4-Channel, Low Noise, Low Power, 24-Bit, Sigma-Delta ADC with PGA and Reference Data Sheet (Rev. D), Analog Devices, 2018.

[29] R. B. Cattell, «Parallel proportional profiles and other principles for determining the choice of factors by rotation,» *Psychometrika*, vol. 9, p. 267–283, 1944.

[30] R. A. Harshman, «Foundations of the PARAFAC procedure: model and conditions for an 'explanatory' multi-mode factor analysis, UCLA Working Papers in phonetics,» *UCLA Working Papers in Phonetics,* vol. 16, pp. 1-84, 1970.

[31] R. A. Harshman, PARAFAC: methods of three-way factor analysis and multidimensional scaling according to the principle of proportional profiles, UCLA, 1976.

[32] H. A. L. Kiers, Three-way methods for the analysis of qualitative and quantitative two-way data., Groningen, 1989.

[33] P. M. Kroonenberg, Applied Multiway Data Analysis, John Wiley & Sons, 2008.

[34] R. Bro y H. A. L. Kiers, «A new efficient method for determining the number of components in PARAFAC models,» *Journal of Chemometrics,* vol. 17, pp. 274-286, 2003.

[35] K. Pearson, «On lines and planes of closest fit to systems of points in space,» *Philosophical Magazine*, vol. 2, pp. 559-572, 1901.

[36] I. Jolliffe, Principal Component Analysis, 2nd ed., Springer, 2002.

[37] N. Cristianini y J. Shawe-Taylor, «Chapter 6: Support vector machine,» de *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*, Cambridge, Cambridge University Press, 2013, p. 93.

[38] ARM Limited, Arm® Cortex®-M4 Processor: Revision: r0p1 Technical Reference Manual, 2020.

[39] R. Segger, «Segger Blog: Floating-point face-off, part 2: Comparing performance,» Segger, 7 November 2019. [En línea]. Available: https://blog.segger.com/floating-point-face-off-part-2-comparing-performance/. [Último acceso: June 2020].

[40] R. Bro, «The N-way Toolbox,» 2020. [En línea]. Available: https://es.mathworks.com/matlabcentral/fileexchange/1088-the-n-way-toolbox. [Último acceso: 20 February 2020].

[41] E. Criswell, Cram's Introduction to Surface Electromyography, Jones & Barlett, 2011.

APPENDIX A: ETHICAL, ECONOMICAL, SOCIAL AND ENVIRONMENTAL IMPACT

A.1 Introduction

The work is developed in the R&D of the health sector. As mentioned in the first chapter the purpose is to make portable health devices that uses sEMG measures to improve the early detection of diseases and improve the daily life of those affected.

Now there is no such a device in the market and one of the challenges presents is to make a classifier in a resource-limited platform. This will enable to work on batteries during all day.

A.2 Description of relevant aspects related to the project

This thesis has no ethical treat, however the applications that may appear could. An important issue with wearables that log health related data is the use of it. The future users and us as developers should be concern about it.

In the economic aspect the healthcare system in Spain is a substantial economic activity (5.9 % of GDP in 2018[1]). Medical equipment, and this sEMG subfield, is a product of high technological innovation. This implies that lot of resources must be invested before a commercial prototype and revenue is obtained. This work is an early stage for any possible viable solution.

In the social aspect, the daily life of thousands of neurological sufferers could be helped if the ultimate goal of a portable sEMG device is achieved. It would not only benefit the patients but all the family and caretakers around them. It is crucial in our society with a growing elder population to develop affordable solutions for healthcare, to make our welfare system sustainable.

The environmental impact of the project is unclear because it has not yet been developed. But it is a new device so probably more electronic waste will be generated. The approach that should be taken is to make long lasting system with interchangeable batteries and electrodes, so they have a long-life cycle.

A.3 Detailed analysis of private health data

The private data of users was recently protected by the General Data Protection Regulation (GDPR), however abusive terms of use and contracts still exits. Data has become a new business that has little regulation against monopoly, most of recent startups they are funded in base of which data will be to acquire. The reason is simple nowadays the patterns of user behavior can be predicted by using machine learning, but these algorithms requires enormous quantities of data to work. And the only way of obtaining data is recording it so the first company that starts to save and classify it will be ahead of the competitors. Even further data acquisition cannot be accelerated if a new company wants to compete against the leading one it always will be behind in terms of data by the difference of time between both companies started recording data. Privacy should be protected but also data should be made public so there is no unfair competition in the market. In our project the aim is to make public available the results and the datasets generates to contribute to this cause.

[1] "Estadística de Gasto Sanitario Público 2018" published in 2020 (https://www.mscbs.gob.es/estadEstudios/estadisticas/docs/EGSP2008/egspPrincipalesResultados.pdf)

A.4 Conclusions

This project in the current state has little impact in any of the above-mentioned aspects. But it will set the basis for the health devices of the future that may help thousands of people to improve their life quality. It will also rise concerns about the environmental impact, but with a modular design with replaceable parts the problem can be solved. In the ethics aspect the problem is already in our society with all the information that we as users generate.

APPENDIX B: BUDGET

Human Resources	Cost per hour (€)	Hours	Subtotal
Salary junior electronic designer	16.2	1040	16848
Salary senior engieneer	24	170	4080
		Total Human cost	**16848**

Hardware Resources	Unitary cost (€)	Month used	Amortization in months	Subtotal
Acquisition platform	120	8	8	120
Personal Computer	1800	8	32	450
			Total Hardware	**570**

Software Resources	Unitary cost (€)	Month used	Amortization in months	Subtotal
Matlab license	800	6	12	400
Statistics and Machine Learning toolbox	400	6	12	200
Microsoft 365 (Word, Powerpoint, Excel)	150	6	12	75
			Total Software cost	**675**

	Total direct cost	**18093**
General Costs (15%)		2713.95
	Total execution	**20806.95**
Profit margin (10%)		2080.695
	Total execution with profit	**22887.65**
VAT (21%)		4806.405
	Total	**27694.05 €**

APPENDIX C: CODE

C.1 dataExtraction.m

```
%%dataExtraction.m
% Extracts the data from the Ninapro DB6 and generates the data cube
% this data cube contains the time array and the converted ones The
user
% is asked about which gesture, subect and day wants. The original raw
% files have to been in the providen folder by the ninapro dataset.

%Accelerometer data is left out

%%Automatic reading
%  This script must be run in the DB6 folder
% subfolders 'DB6_s1',   ,'DB6_s10' (each of them conatins data from
one
% subject)
% e.g. for subject 1 DB6_s1/S1_D1_T1.mat ... S1_D5_T2.mat

% output
% In the DB6_s1 folder a ".mat" file will be created with the
following
% matrices M1 ... M6 (time domain) and Mfft1 ... Mfft6
% The file names C1_3_2_E4_O2_PRE1.matcodifies:
        % subjet 1,
        % day 3,
        % time 2 (afternoon)
        % estimulus 4 (see table)
        % object 2 (see table)
        % Pretratamiento 1 (zero padding)
% The script clears the workspace and come back to 'DB6'

%Parameters of fft_vector function
Fs = 2000;      %Frecuency sample of the data
max_f = 500;    %Maximum freq to save
tol = 1;        %Tolerance to compare the max_f with the closest f
%The user will be asked if he/she wants to pad with zeroes
    %(not recommended unless it takes to long)

fast_time = input('Improve speed by zero padding? 0 (off) 1 (on)    ')
;

s = input('Subject (1 o 10)   ');

d = input('Day  (1, 2,..., 5)   ');

t = input('Time  (1 0 2)   ');

cd (['DB6_s', num2str(s)]); %

fl = ['S',num2str(s),'_D',num2str(d),'_T',num2str(t),'.mat'];
load (fl, 'emg', 'restimulus' , 'reobject', 'daytesting', 'subj',
'time');

cd ..;
% control

if d ~= daytesting; error ('error en el dia '); end %hacer algo para
parar el script
if s ~= subj; error ('error en el sujeto ');  end%hacer algo
if t ~= time; error ('error en el tiempo '); end%hacer algo
```

```
e=input ('Estímulo 1,2,...., 7    ');
o=input ('Objeto 1, 2    ');


tabla=[1 2 4
    3 12 23
    4 14 10
    6 25 18
    9 19 20
    10 8 15
    11 4 9];

es=tabla(e,1); %restimulus == es
ob=tabla(e,o+1); %reobject == ob


% generetas the logical vector
ind_es = logical(restimulus == es);
ind_ob = logical(reobject == ob);

a=ind_es.*ind_ob;

% cont have booth the first and last sample with the required
codification
l=length(a);
k=1;

if a(1) == 1; cont(1)=1; k = k+1; end
for i = 2:l
    if a(i-1)-a(i) == -1;
        cont(k)=i; k=k+1;
    end
    if a(i-1)-a(i) == 1;
        cont(k) = i-1;k = k+1;
    end
end
if a(l)==1 ; cont(k)=l;end

% Generates the six matrices (6 replicas)


for i=1:length(cont)/2
    A = emg(cont(2*i-1):cont(2*i),[1:8 11:16]);
    for ch = 1: size(A,2)
    A_fft(:,ch) = fft_vector(A(:,ch), Fs, max_f, tol, fast_time);
    end
    eval(['M',num2str(i),'=A;']);
    eval(['Mfft',num2str(i),'=A_fft;']);
    clear A A_fft
end


flou=['C',num2str(s),'_',num2str(d),'_',num2str(t),'_E',num2str(e),'_O
',num2str(o),'_PRE',num2str(fast_time),'.mat'];
save (flou, 'M1', 'M2', 'M3', 'M4', 'M5'
,'M6','Mfft1','Mfft2','Mfft3','Mfft4','Mfft5','Mfft6');
clear all
```

C.2 fft_vector()

```
function vector = fft_vector(vector_in, Fs, f_max, tol, fast)
%fft_vector Return the magnitude fft of a given vector and cuts up up
%to a certain frecuency.
%Fs frecuency sample rate, cut_f desired maximum frecuency
%tol is the error admited when comparing f_max
%fast is the parameter to extend the length of the power to next 2
power 1
%to enable
if f_max > (Fs/2)
    error('The f_max cannot be greater than half the sampling
frequency,',...
        ' god save Nyquist')
end



L_vector = length(vector_in);           %Length of the vector

if fast == 1
    L_adjusted = 2^nextpow2(L_vector);
elseif fast == 0
    L_adjusted = L_vector;
else
    error('The fast argument can only take value of 0 or 1')
end

f = Fs*(0:(L_adjusted/2))/L_adjusted;   %Frrequency value for each
index
index_cut = find(abs(f - f_max)<tol );  %The index where the frecuency
                                        %reaches the desired value
if isempty(index_cut)
    warning('No coincidences for the given cut frecuency and tol,',
...
        'consider rising the tol argument')
end

if length(index_cut) > 1
    warning('More than one frequency found for the given tolerance')
    %It works ok but it should be noticed just in case
end
temp_fft = fft(vector_in,L_adjusted);
vector = abs(temp_fft(1:index_cut)/L_adjusted);

end
```

C.3 serialprocessdata()

```
function [CH_matrix] = serialprocessdata(filepath, starting_byte_1,...
    starting_byte2, Vref, double, precision)
%SERIALPROCESSDATA Converts the given binary datafile into a matrix of
four
%double array.
%   Specify the two starting bytes previous to start transmission.
Expected
%   int32 data.
%   Double is a flag to enable the conversion to volts in double
notation
%   precision will set to double floating esle simple floating
file = fopen(filepath);
a = fread(file,'uint8');
a = uint8(a);
i_length=length(a);

i=1;
ch_i = 1;


 while i<=(i_length-10)
    if(a(i) == starting_byte_1)
        i = i+1;
        if(a(i) == starting_byte2)
           i = i+1;
           ch1(ch_i) = typecast(a(i:i+3), 'int32');
           i = i+4;
           ch2(ch_i) = typecast(a(i:i+3), 'int32');
           i = i+4;
           ch3(ch_i) = typecast(a(i:i+3), 'int32');
           i = i+4;
           ch4(ch_i) = typecast(a(i:i+3), 'int32');
           i = i+4;
           ch_i = ch_i+1;
        end
    else
        i = i +1;
    end
 end
 fclose(file);
 if double == 1
     ch1_volt = transpose(sample2volt(Vref, ch1, precision));
     ch2_volt = transpose(sample2volt(Vref, ch2, precision));
     ch3_volt = transpose(sample2volt(Vref, ch3, precision));
     ch4_volt = transpose(sample2volt(Vref, ch4, precision));
 else
     ch1_volt = transpose(ch1);
     ch2_volt = transpose(ch2);
     ch3_volt = transpose(ch3);
     ch4_volt = transpose(ch4);
 end
CH_matrix = [ch1_volt ch2_volt ch3_volt ch4_volt];

end
```

C.4 circbuff.h

```
/*
 *
 *  Created on: 20 May 2020
 *  Author: Pablo Sarabia Ortiz                <psarabia@b105.upm.es>
 *  Modified:
 *
 */
/**
 * @file circbuff.h
 */
#ifndef CIRCBUFF_H_
#define CIRCBUFF_H_

#include "yetiOs.h"

typedef struct cBuf_t cBuf_t;

typedef cBuf_t* cBuf_handle_t;

cBuf_handle_t cBufInit(int32_t* buffch0, int32_t* buffch1, int32_t*
buffch2,
int32_t* buffch3, size_t size);

void cBufReset(cBuf_handle_t cBuf);

size_t cBufSize(cBuf_handle_t cBuf);

size_t cBufCapacity(cBuf_handle_t cBuf);

int8_t cBufPut(cBuf_handle_t cBuf, int32_t data, uint8_t channel);

int8_t cBufPop(cBuf_handle_t cBuf, int32_t * dataArray);

uint8_t cBufisempty(cBuf_handle_t cBuf);

uint8_t cBufisfull(cBuf_handle_t cBuf);

uint8_t cBufexpectedchannel(cBuf_handle_t cBuf);
#endif /* CIRCBUFF_H_ */
```

C.5 circbuff.c

```
/*
 * circbuff.c
 *
 *  Created on: 15 May. 2020
 *  Author: Pablo Sarabia Ortiz          <psarabia@b105.upm.es>

 */
/**
 * @file circbuff.c
 * @brief Library for handling the circular buffer of 4 channels
 */

#include <stdlib.h>
#include "yetiOs.h"
#include "circbuff.h"


struct cBuf_t {
      int32_t * buffer0;
      int32_t * buffer1;
      int32_t * buffer2;
      int32_t * buffer3;
      uint8_t head;
      uint8_t tail;
      size_t max;
      uint8_t full;
      uint8_t empty;
      uint8_t expected_channel;     //This will make the samples
synchronized even if the adc skips one
};

/** cBufInit(uint32_t* buffch0, uint32_t* buffch1,
 *                 uint32_t* buffch2, uint32_t* buffch3,
 *                 size_t size)
 *    Initialize the Circular Buffer
 *    */
cBuf_handle_t cBufInit(int32_t* buffch0, int32_t* buffch1,
                   int32_t* buffch2, int32_t* buffch3,
                   size_t size){

  cBuf_handle_t cBuf = pvPortMalloc(sizeof(cBuf_t));

  cBuf->buffer0 = buffch0;
  cBuf->buffer1 = buffch1;
  cBuf->buffer2 = buffch2;
  cBuf->buffer3 = buffch3;
  cBuf->max = size;
  cBufReset(cBuf);

  return cBuf;
}

void cBufReset(cBuf_handle_t cBuf){

  cBuf->head = 0;
  cBuf->tail = 0;
  cBuf->full = 0;
  cBuf->empty = 1;
  cBuf->expected_channel = 0;
}
/** void cBufIncHead(cBuf_handle_t cBuf)
```

```
 *    Increase in one the value of the head and the tail if the
 *    buffer is full (overwrite).
 *    It also sets the full flag if head equal to tail.
 *
 **/
static void cBufIncHead(cBuf_handle_t cBuf){

  if (cBuf->full == 1){
      cBuf->tail = (cBuf->tail + 1) % cBuf->max;
  }
  cBuf->head = (cBuf->head + 1) % cBuf->max;
  cBuf->full = (cBuf->head == cBuf->tail);
  cBuf->empty = 0;
}

/** void cBufIncTail(cBuf_handle_t cBuf)
 *    Increase the tail index by one should be invoked
 *    after a check if the buffer is not empty
 */
static void cBufIncTail(cBuf_handle_t cBuf){

  cBuf->full = 0;
  cBuf->tail =(cBuf->tail + 1) % cBuf->max;
  cBuf->empty = (cBuf->head == cBuf->tail);
}


/** size_t cBufSize(cBuf_handle_t cBuf)
 *    Retrieves the remaining space in the buffer
 */
size_t cBufSize(cBuf_handle_t cBuf){

  size_t size = cBuf->max;

  if(cBuf->full){
      if(cBuf->head >= cBuf->tail){
        size = (cBuf->head - cBuf->tail);
      }
      else{
        size = (cBuf->max + cBuf->head - cBuf->tail);
      }
  }
  else{
      size = 0;
  }

  return size;
}

/** size_t cBufCapacity(cBuf_handle_t cBuf)
 *    Returns the size of the cBuf
 * */
size_t cBufCapacity(cBuf_handle_t cBuf){

  size_t size = cBuf->max;
  return size;
}

/** uint8_t cBufPut(cBuf_handle_t cBuf, uint32_t data)
 *    Appends the value to next location overwrite if full
 *    returns 0 if done -1 if the channel is out of boundaries
 * */
int8_t cBufPut(cBuf_handle_t cBuf, int32_t data, uint8_t channel){
```

```
  int8_t ret = -1;
    switch(channel){
      case 0:
      cBuf->buffer0[cBuf->head] = data;
      cBuf->expected_channel = 1;
      ret = 0;
      break;
      case 1:
      cBuf->buffer1[cBuf->head] = data;
      cBuf->expected_channel = 2;
      ret = 0;
      break;
      case 2:
      cBuf->buffer2[cBuf->head] = data;
      cBuf->expected_channel = 3;
      ret = 0;
      break;
      case 3:
      cBuf->buffer3[cBuf->head] = data;
      cBuf->expected_channel = 0;
      cBufIncHead(cBuf);
      ret = 0;
      break;
      default:
      break;
    }


  return ret;
}

/** uint8_t cBufPop(cBuf_handle_t cBuf, uint32_t * dataArray)
 *    Reads the data from the tail position of 4 channels returns 0 if
done
 *    -1 if the buffer is empty.
 *    IMPORTANT:¡¡ dataArray must be a pointer to a buffer for 4
uint32_t !!!
 * */
int8_t cBufPop(cBuf_handle_t cBuf, int32_t * dataArray){

  int8_t r = -1;
  if(!cBufisempty(cBuf)){
      dataArray[0] = cBuf->buffer0[cBuf->tail];
      dataArray[1] = cBuf->buffer1[cBuf->tail];
      dataArray[2] = cBuf->buffer2[cBuf->tail];
      dataArray[3] = cBuf->buffer3[cBuf->tail];
      cBufIncTail(cBuf);

      r = 0;
  }
  return r;
}

uint8_t cBufisempty(cBuf_handle_t cBuf){
  return cBuf->empty;
}

uint8_t cBufisfull(cBuf_handle_t cBuf){
  return cBuf->full;
}

uint8_t cBufexpectedchannel(cBuf_handle_t cBuf){
  return cBuf->expected_channel;}
```

C.6 cycles()

```
function [cycles,time] = cycles(SVM_opt, vector_size, precision,
n_gesture, freq)
%CYCLES Givemn the optimization settings: SVM_opt 0 for one vs all 1
for
%one vs one vector_size size of the array sample, precision 0 for
double 1
%for single float, n_gesture  and the freq of the processor
%   SVM uses a the following formula for each gesture in each binary
%   classifier:f(x)=(x/s)Beta+b. where Beta are the coef of the svm x
the
%   samples, s a constant and b another constant (bias and kernel
scale respectively)
%   the computational cost is to load Beta=(vector_size*n_gestures)
and to do
%   vector_size multiplications and 2*vector_size additions, the
precision
%   will determine if they are double or single operations.
%   the number of load will be (n_gesture*n_classifers + 1); (loading
data and coeffs)
    cyc_load_32 = 2;
    cyc_load_64 = 3;
    cyc_add_32    = 1;
    cyc_mult_32 = 1;
    cyc_mult_64 = 56; %%aproximate empirical value because its done by
sw
    cyc_store_32 = 2;
    cyc_store_64 = 3;
    cyc_add_64 = 54;
    if SVM_opt == 1
        classifiers  = n_gesture;
    else
        classifiers = n_gesture *(n_gesture-1)/2;
    end
    n_load = vector_size*(n_gesture*classifiers + 1);
    n_store = n_gesture*classifiers;
    n_mult = vector_size*n_gesture*classifiers;
    n_add = 2*vector_size*n_gesture*classifiers;
    if precision == 32
        cycles = n_load * cyc_load_32 + n_store * cyc_store_32 + ...
                n_mult * cyc_mult_32 + n_add * cyc_add_32 ;
    else
        cycles = n_load * cyc_load_64 + n_store * cyc_store_64 + ...
                n_mult * cyc_mult_64 + n_add * cyc_add_64 ;
    end
    time = cycles/freq;
end
```

C.7 memoryfootprint()

```
function [mem_bytes] = memoryfootprint(SVM_opt, vector_size,
precision, n_gestures)
%MEMORYFOOTPRINT Function that calculates the memory footprint of a
single
%implementation according to the SVM_opt (o for one vs one 1 for one
vs
%all) vector size(number of samples) and precision (64 for double
float),
%32 for single float
%   one vs all has n_gestures binary classifiers
%   one vs one has n_gestures*(n_gestures-1)/2
%   each binarty classifier is composed by an array beta sized
%   (n_gesturesxvector_size)
%

if SVM_opt == 1
    classifiers  = n_gestures;
else
    classifiers = n_gestures *(n_gestures-1)/2;
end
mem_bytes = (precision/8) * classifiers * vector_size * n_gestures;
end
```