**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ**
**ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ**
**ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ**

# Διπλωματική Εργασία

του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών της Πολυτεχνικής Σχολής του Πανεπιστημίου Πατρών στα πλαίσια του προγράμματος μεταπτυχιακών σπουδών
«Βιοϊατρική Μηχανική»

**ΣΤΡΑΤΑΚΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ**
του Ιωάννη
Αριθμός Μητρώου: 1003687

Θέμα

## «Myoelectric control of Robotic arm»

Επιβλέπων
Ευάγγελος Δερματάς

**Αριθμός Διπλωματικής Εργασίας:**

Πάτρα, Μάιος 2019

UNIVERSITY OF PATRAS
DEPARTAMENT OF ELECTRICAL AND COMPUTER ENGINEERING

MSc Thesis of

**Stratakis Konstantinos**
ID:1003687
For the Postgraduate Education Program
“Biomedical Engineering”

# «Myoelectric control of Robotic arm»

Supervisor
Evaggelos Dermatas

Patras, May 2019

# ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με θέμα

**«Myoelectric control of Robotic arm»**

Του φοιτητή του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

**ΣΤΡΑΤΑΚΗΣ ΚΩΝΣΤΑΝΤΙΝΟΣ**
του Ιωάννη
Αριθμός Μητρώου: 1003687

Παρουσιάστηκε δημόσια και εξετάστηκε στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις
……../……../……

Ο Επιβλέπων

Καθηγητής Δερματάς Ευάγγελος

Ο Διευθυντής του Τομέα :
Καθηγητής?

**Αριθμός Διπλωματικής Εργασίας:**

## Θέμα: «Myoelectric control of Robotic arm»

Φοιτητής: Στρατάκης Κωνσταντίνος

Επιβλέπων: Ευάγγελος Δερματάς

Summary

The subject of this Master Thesis is the use of recorded myoelectric signals for the control of a robotic arm. The robotic arm with 4 degrees of freedom was assembled with lego building blocks and the operation of its motors was done with the lego NXT microprocessor. The NXT communicated with an STM32 microprocessor via Bluetooth. In the latter microprocessor the myoelectric signal was recorded with an analog to digital converter (ADC). After the necessary signal processing the data recorded were used to calculate a four dimensional vector, who's each element is the energy of each channel over a window of time, was used to train a Support Vector Machine for classification. Based on the classification of the newly received data corresponding commands to these classes are sent to the NXT.

I owe gratitude to my supervising professor Mr. Dermatas for his guidance, to my parents for their support.

# Contents

# 1.Introduction

Despite the fact that robots came about approximately 50 years ago, the way humans control them is still an important issue. In particular, since robots are being used more frequently in everyday life tasks (e.g., service robots, robots for clinical applications), the human–robot interface plays a role of outmost significance. Electromyographic (EMG) signals provide an extremely useful non-invasive measure of ongoing muscle activity. They could thus be potentially used for controlling devices such as robotic prosthetics that can restore some or all of the lost motor functions of amputees and disabled individuals. Most commercially available prosthetic devices have limited control (e.g., one degree-of-freedom in the case of a prosthetic gripper), nowhere near the original levels of flexibility of the organ they are intended to replace. Amputees and partially paralyzed individuals typically have intact muscles that they can exercise varying degrees of control over. Thus, a lot of ongoing research is focus on decoding the signals that reach these intact muscles to operate robotic devices, prosthetic and not.

This diploma thesis aimed to investigate the efficiency of real-time controlling a robotic arm with EMG signal captured while a user is holding a gesture. Each gesture performed, is presumably producing a distinct signal that describes it and can be recognized by it. Thus, a pattern recognition algorithm, Support Vector Machines, is trained to recognize these gestures and then send commands to the motors of the robot. The methods and the procedures followed in this thesis are inspired work done at the University of Washington by Pradeep Shenoy, Kai J. Miller, Beau Crawford, and Rajesh P. N. Rao. In their research Support Vector Machines were trained to recognize 8 gestures using EMG signal detected by electrodes on 7 muscles. These muscles were:

| | Muscle | Function |
|---|---|---|
| 1 | Brachioradialis | forearm flexion |
| 2 | Extensor carpi ulnaris | extension and adduction of hand at the wrist |
| 3 | Pronator teres | pronation and elbow flexion |
| 4 | Extensor communis digitorum | finger extension at metacarpo-phalangeal joints, wrist extension at forearm |
| 5 | Flexor carpi radialis | hand flexion and abduction at wrist |
| 6 | Anconeus | antagonistic activity during forearm pronation |
| 7 | Pronator quadratus | initiates pronation |

**Table 1.1**

**Figure 1.1: On the photographs above the placement of the electrodes is shown**

The 8 gestures that were classified are:

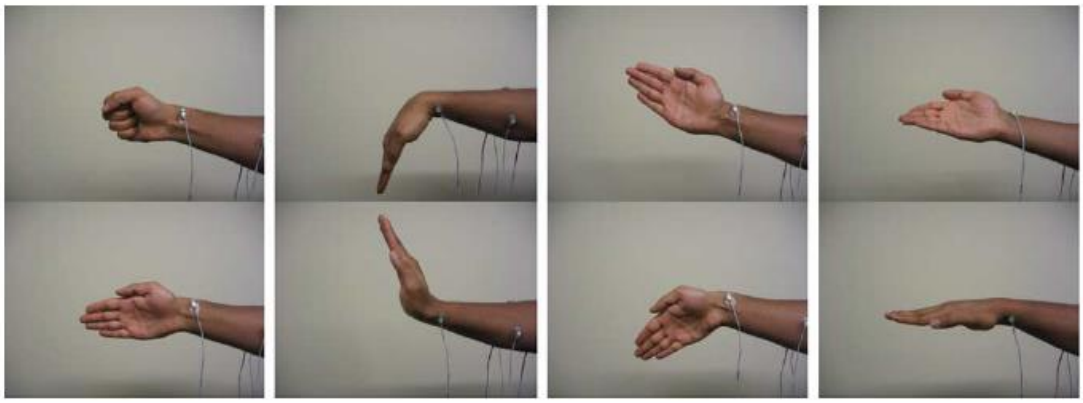


**Figure 1.2**

In their experiments 92-98% classification accuracy was achieved cross-subject in offline evaluation. In their work they stated that the full eight gesture set can be recognized with fewer than 8 electrodes. It is also notable that all gestures produced a motor movement to a 4 degrees of freedom robotic arm for a task-oriented experiment. In this case the user is consequently performing gestures. The situation, where the user wants to be at ease and then start operating the robot or stop operating the robot for a while between gestures was not addressed. This is a major issue when the goal is control of a prosthetic device. Furthermore, in their work there was no mention on the analog circuits used for capturing the EMG signal. Thus, it is presumed they used integrated circuits found in retail. The price of such a circuit for recording the signal of 1 muscle is around 30$.

In this thesis the accuracy that can be achieved using only 4 muscles was investigated. The muscles used are:

1.Extensor carpi ulnaris
2.Pronator quadratus
3.Extensor communis digitorum
4.Flexor carpi radialis.

Furthermore, an additional gesture was used for classification. This gesture is done by the user when he relaxes his hand. While, this gesture is detected the robot is idle. This was done to address the case of the user wanting to stop moving the robot while performing a task. Also, a custom build analog circuit was implemented for the amplification and filtering of the EMG signal. This aimed to investigate the suitability of cheap electronics for signal processing for a delicate signal like EMG. The cost of this complete circuit for the 4 channels required for the 4 muscles was around 8$.

## 2.Overview of Procedure

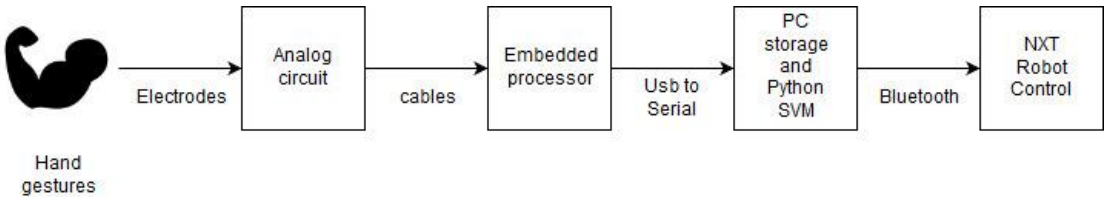



**Figure 2.1:Procedure Chain**

The user holds a static gesture with his hand and the system recognizes it using a trained Support Vector Machine giving a corresponding command to the motors of the robot arm. The procedure of the control is divided in four hardware-separated segments. The physical circuit, the code written in C++ executed on the embedded processor, the code written in Python executed on a PC and the commands sent to the NXT controlling the motors. However, before the execution of the Python code there is a step in which the data need to be collected in order the SVM to be trained.

The role of the analog circuit is to amplify and filter the artifacts from the EMG signal captured by 4 electrodes placed on the 4 muscles mentioned in the introduction. The software of the microprocessor samples the analog signal creating a digital one. This digital signal is filtered again with digital filters to remove the artifacts the analog circuit failed to completely remove. Then the Root mean square (Rms) of each channel is calculated over a fixed window of time. The Rms values taking the form of a 4-D vector are sent to a PC and are stored. Thus, the data-set for the training of the Support Vector Machines algorithm is created. When the SVM is trained the gestures performed by the user are recognized in real time and appropriate commands are sent to the robot.

## 3.Myoelectric Signal

In this chapter an explanation is given of how the voltage detected by the electrodes is generated. The body has nerves that connect the brain to the rest of the organs and muscles, just like telephone wires connect homes all around the world. When you want your hand to move, your brain sends signals through your nerves to your hand telling the muscles to contract. But the nerves don't just say "hand, move." Instead the nerves send lots of electrical impulses (called action potentials) to different muscles in the hand, allowing the movement of the hand with extreme precision.
Neurons are a special type of cell with the sole purpose of transferring information around the body. Neurons are similar to other cells in that they have a cell body with a nucleus and organelles. However, they have a few extra features which allow them to be fantastic at transferring action potentials. Dendrites receive signals from neighboring neurons. Axon transmit signals over a distance (like telephone wires). Axon terminal, transmit signals to other neuron dendrites or tissues (like a radio transmitter). Myelin sheath speeds up signal transmission along the axon.

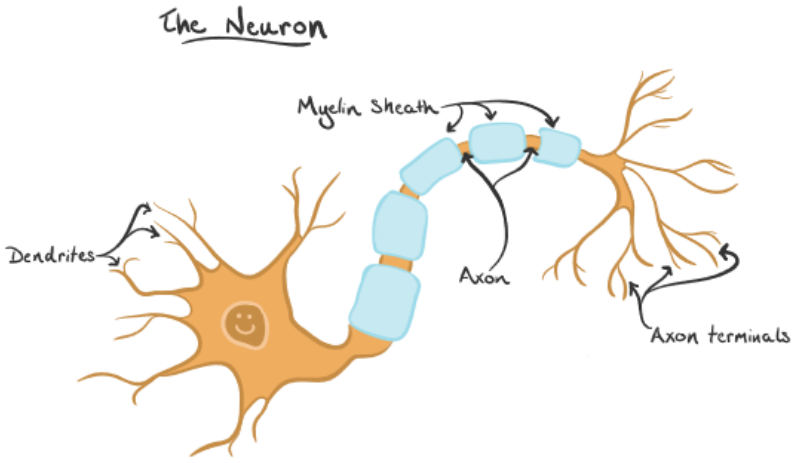



**Figure 3.1: The Neuron**

Concentration gradients are key behind how action potentials work. In terms of action potentials, a concentration gradient is the difference in ion concentrations between the inside of the neuron and the outside of the neuron (called extracellular fluid). If there is a higher concentration of positively charged ions outside the cell compared to the inside of the cell, there would be a large concentration gradient. The same would also be true if there were more of one type of charged ion inside the cell than outside. The charge of the ion does not matter, both positively and negatively charged ions move in the direction that would balance or even out the gradient.

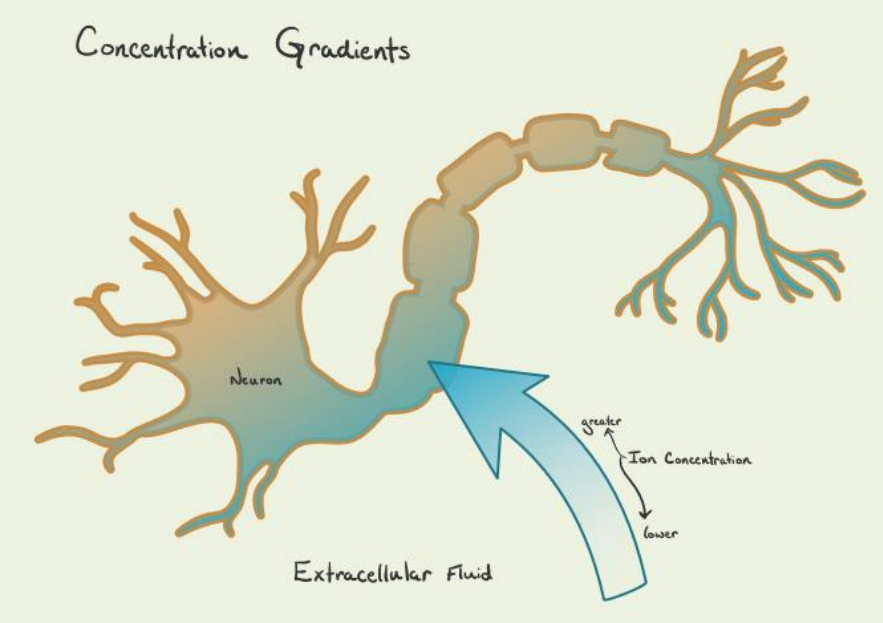



**Figure 3.2: Concentration Gradients**

Neurons have a negative concentration gradient most of the time, meaning there are more positively charged ions outside than inside the cell. This regular state of a negative concentration gradient is called resting membrane potential. During the resting membrane potential there are more sodium ions (Na+) outside than inside the neuron and more potassium ions (K+) inside than outside the neuron. The concentration of ions isn't static though. Ions are flowing in and out of the neuron constantly as the ions try to equalize their concentrations. The cell however maintains a fairly consistent negative concentration gradient (between -40 to -90 millivolts). The neuron cell membrane is super permeable to potassium ions, and so lots of potassium leaks out of the neuron through potassium leakage channels (holes in the cell wall). The neuron cell membrane is partially permeable to sodium ions, so sodium atoms slowly leak into the neuron through sodium leakage channels. The cell wants to maintain a negative resting membrane potential, so it has a pump that pumps potassium back into the cell and pumps sodium out of the cell at the same time.

Action potentials (those electrical impulses that send signals around your body) are nothing more than a temporary shift (from negative to positive) in the neuron's membrane potential caused by ions suddenly flowing in and out of the neuron. During the resting state (before an action potential occurs) all of the gated sodium and potassium channels are closed. These gated channels are different from the leakage channels, and only open once an action potential has been triggered. We say these channels are "voltage-gated" because they are open and closed depends on the voltage difference across the cell membrane. Voltage-gated sodium channels have two gates (gate m and gate h), while the potassium channel only has one (gate n).

- Gate m (the activation gate) is normally closed, and opens when the cell starts to get more positive.
- Gate h (the deactivation gate) is normally open, and swings shut when the cells gets too positive.
- Gate n is normally closed, but slowly opens when the cell is depolarized (very positive).

Voltage-gated sodium channels exist in one of three states:

1. Deactivated (closed) - at rest, channels are deactivated. The m gate is closed, and does not let sodium ions through.
2. Activated (open) - when a current passes through and changes the voltage difference across a membrane, the channel will activate and the m gate will open.
3. Inactivated (closed) - as the neuron depolarizes, the h gate swings shut and blocks sodium ions from entering the cell.

Voltage-gated potassium channels are either open or closed.

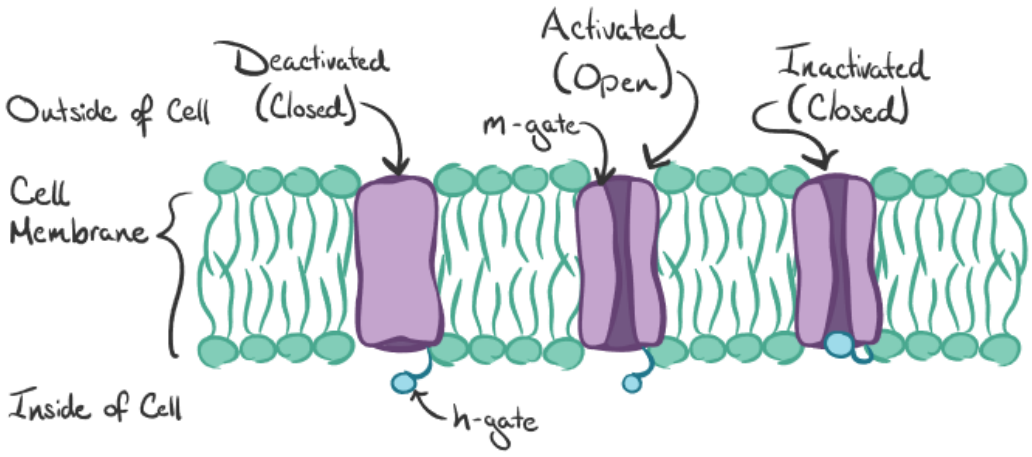



**Figure 3.3: Ion Channels**

There are three main events that take place during an action potential. Firstly, a triggering event occurs that depolarizes the cell body. This signal comes from other cells connecting to the neuron, and it causes positively charged ions to flow into the cell body. Positive ions still flow into the cell to depolarize it, but these ions pass through channels that open when a specific chemical, known as a neurotransmitter, binds to the channel and tells it to open. Neurotransmitters are released by cells near the dendrites, often as the end result of their own action potential! These incoming ions bring the membrane potential closer to 0, which is known as depolarization. An object is polar if there is some difference between more negative and more positive areas. As positive ions flow into the negative cell, that difference, and thus the cell's polarity, decrease. If the cell body gets positive enough that it can trigger the voltage-gated sodium channels found in the axon, then the action potential will be sent. Sequentially, depolarization makes the cell less polar (membrane potential gets smaller as ions quickly begin to equalize the concentration gradients). Voltage-gated sodium channels at the part of the axon closest to the cell body activate, thanks to the recently depolarized cell body. This lets positively charged sodium ions flow into the negatively charged axon, and depolarize the surrounding axon. We can think of the channels opening like dominoes falling down - once one channel opens and lets positive ions in, it sets the stage for the channels down the axon to do the same thing. Though this stage is known as depolarization, the neuron actually swings past equilibrium and becomes positively charged as the action potential passes through. Repolarization, the event after depolarization, brings the cell back to resting potential. The inactivation gates of the

sodium channels close, stopping the inward rush of positive ions. At the same time, the potassium channels open. There is much more potassium inside the cell than out, so when these channels open, more potassium exits than comes in. This means the cell loses positively charged ions, and returns back toward its resting state. Lastly, Hyperpolarization, makes the cell more negative than its typical resting membrane potential. As the action potential passes through, potassium channels stay open a little bit longer, and continue to let positive ions exit the neuron. This means that the cell temporarily hyperpolarizes, or gets even more negative than its resting state. As the potassium channels close, the sodium-potassium pump works to reestablish the resting state.

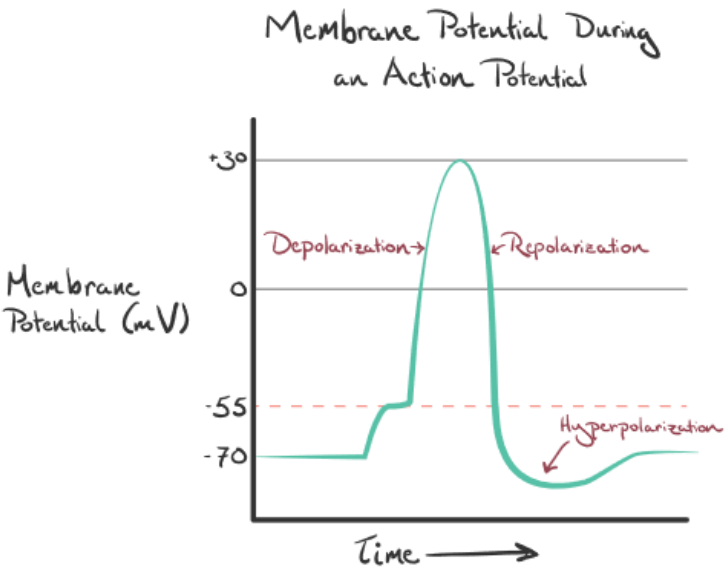



**Figure 3.4: Action potentials**

Action potentials work on an all-or-none basis. This means that an action potential is either triggered, or it isn't – like flipping a switch. A neuron will always send the same size action potential. When the brain gets really excited, it fires off a lot of signals. How quickly these signals fire tells us how strong the original stimulus is, the stronger the signal, the higher the frequency of action potentials. There is a maximum frequency at which a single neuron can send action potentials, and this is determined by its refractory periods.

- Absolute refractory period: during this time it is absolutely impossible to send another action potential. The inactivation (h) gates of the sodium channels lock shut for a time, and make it so no sodium will pass through. No sodium means no depolarization, which means no action potential. Absolute refractory periods help direct the action potential down the axon, because only channels further downstream can open and let in depolarizing ions.

- Relative refractory period: during this time, it is really hard to send an action potential. This is the period after the absolute refractory period, when the h gates are open again. However, the cell is still hyperpolarized after sending an action potential. It would take even more positive ions than usual to reach the appropriate depolarization potential than usual. This means that the initial triggering event would have to be bigger than normal in order to send more action potentials along. Relative refractory periods can help us figure how intense a stimulus is - cells in your retina will send signals faster in bright light than in dim light, because the trigger is stronger.

Refractory periods also give the neuron some time to replenish the packets of neurotransmitter found at the axon terminal, so that it can keep passing the message along. While it is still possible to

completely exhaust the neuron's supply of neurotransmitter by continuous firing, the refractory periods help the cell last a little longer.

A muscle cell, known technically as a myocyte, is a specialized animal cell which can shorten its length using a series of motor proteins specially arranged within the cell. While several associated proteins help, actin and myosin form thick and thin filaments which slide past each other to contract small units of a muscle cell. These units are called sarcomeres, and many of them run end-to-end within a larger fiber called a myofibril. A single muscle cell contains many nuclei, which are pressed against the cell membrane. A muscle cell is a long cell compared to other forms of cells, and many muscle cells connect together to form the long fibers found in muscle tissue.
A muscle cell is a compact bundle of many myofibrils. Each myofibril is made of many sarcomeres bundled together and attached end-to-end. A specialized form of the endoplasmic reticulum, known as the sarcoplasmic reticulum, extends in and around these myofibril bundles. The sarcoplasmic reticulum (SR for short) concentrates a chemical needed for the muscle cells to contract, and is activated by signals from nerve cells. The signals travel through the transverse tubules (T tubules in the picture below) after being received from a nerve and activates the SR. Mitochondria are densely packed throughout muscle cells, to provide a constant flow of ATP (the chemical from that transfers energy within the cell). The entire cell is covered in a specialized cell membrane known as the sarcolemma. The sarcolemma has special opening which allow nerve impulses to be passed into transverse tubules.

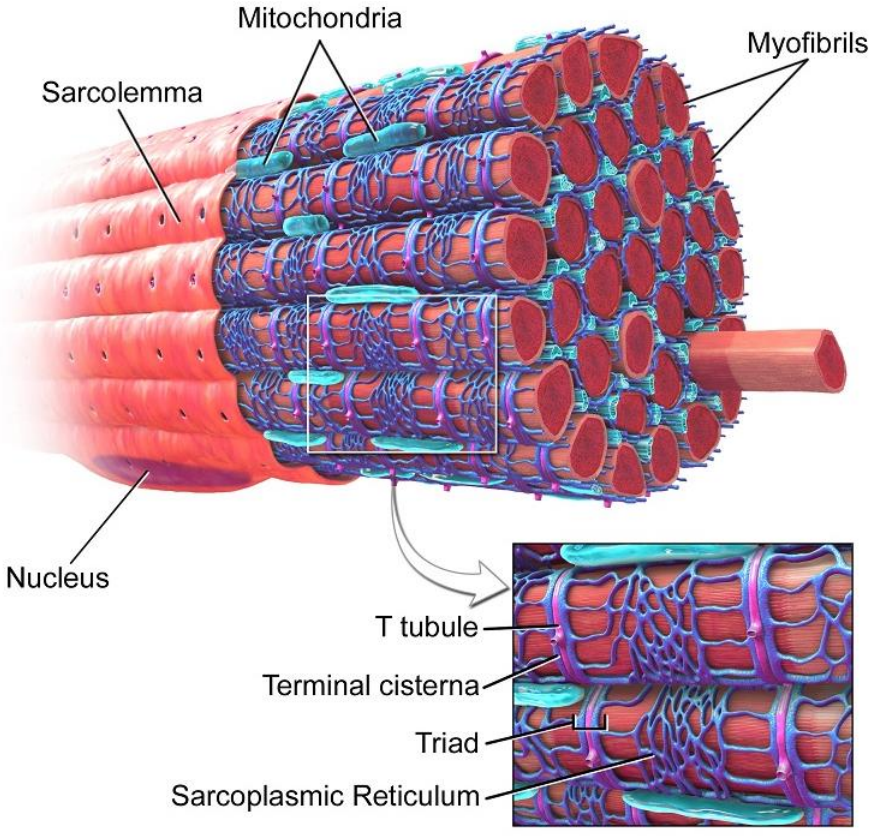



**Figure 3.5: Muscle Fibers**

Each sarcomere is made primarily from thick and thin filaments. Thick filaments are made from repeating units of a protein known as myosin. Myosin has small heads on it which can bind to an actin filament. Repeating units of the protein actin make up the thin filament. Actin is supported by a number of accessory proteins which give the strands stability and allow the muscle to be controlled by nerve impulses. The actin filaments are supported on each end by specialized proteins. The CapZ protein holds actin to the Z plate, while tropomodulin connects to the end of each actin

filament. Nebulin connects CapZ to tropomodulin, providing a structural framework to hold the actin filaments rigid. Another large protein, titin, connects the Z plates together and prevents the sarcomere from being overstretched when it is not contracting. These proteins can be seen in the image below. Actin is covered by two additional proteins, troponin and tropomyosin. Troponin is the small yellow ball in the image below, while tropomyosin is the thread-like protein which follows the actin filament. The myosin proteins can also be seen. The heads extend upward from a thick fiber made of many myosin tails wound together.

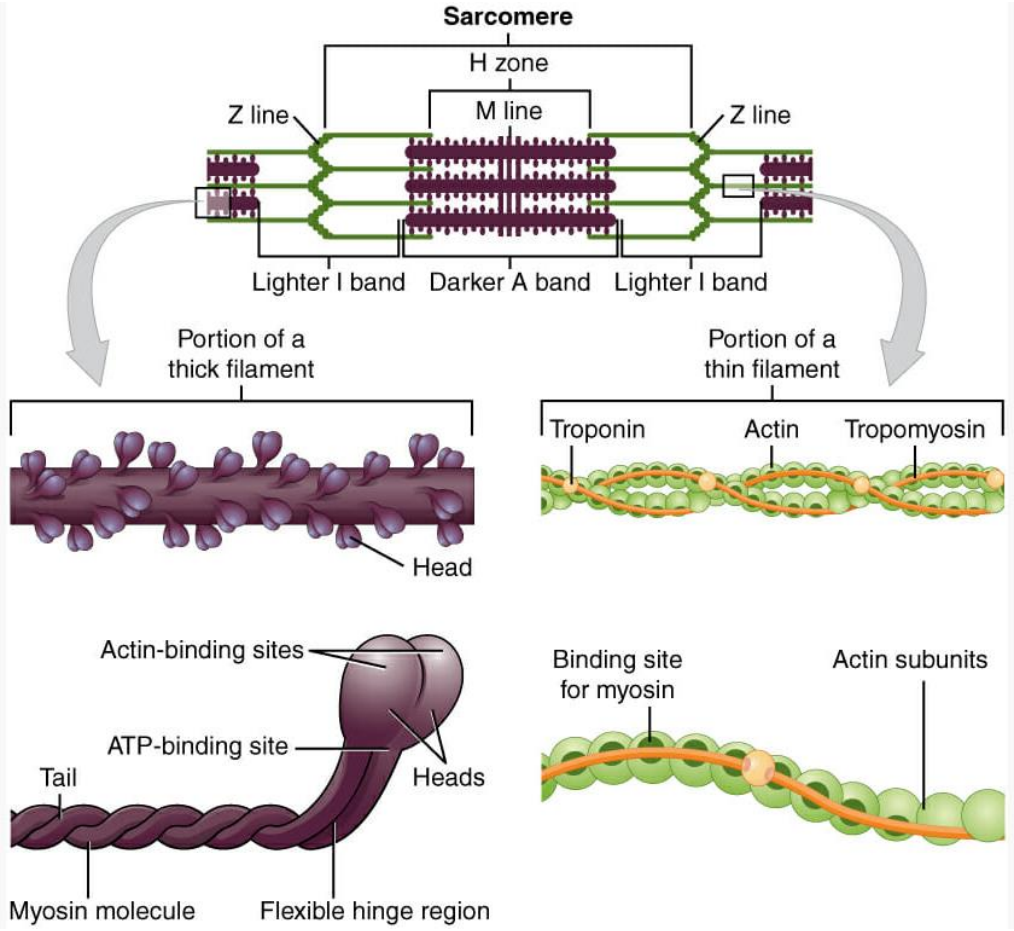



**Figure 3.6: How the muscle contracts.**

To activate a muscle, the brain sends an impulse down a nerve. The nerve impulse travels down the nerve cells to the neuromuscular junction, where a nerve cell meets a muscle cell. The impulse is transferred to the nerve cell and travels down specialized canals in the sarcolemma to reach the transverse tubules. The energy in the transverse tubules causes the SR to release of the $Ca^{2+}$ it has built up, flooding the cytoplasm with calcium. The $Ca^{2+}$ has a special effect on the proteins associated with actin. Troponin, when not in the presence of $Ca^{2+}$, will bind to tropomyosin and cause it to cover the myosin-binding sites on the actin filament. This means that without $Ca^{2+}$ the muscle cell will be relaxed. When $Ca^{2+}$ is introduced into the cytosol, troponin will release tropomyosin and tropomyosin will slide out of the way. This allows the myosin heads to attach to the actin filament. Once this happens, myosin can use the energy gained from ATP to crawl along the actin filament. When many sarcomeres are doing this at the same time, the entire muscle contract. While only a small percentage of the heads are attached at any one time, the many heads and continual use of ATP ensures a smooth contraction. The myosin crawls until it reaches the Z

plate, and full contraction has been obtained. The SR is continually removing $Ca^{2+}$ from the cytoplasm, and once the concentration falls below a certain level troponin rebinds to tropomyosin, and the muscle releases. While the above model is a generalized version of what happens in *skeletal muscle*, similar processes control the contractions of both *cardiac* and *smooth muscle*. By analyzing the whole process of the muscle activation, it is obvious that flow of ions in and out the extracellular matrix is producing the voltages that are detected by the electrodes.

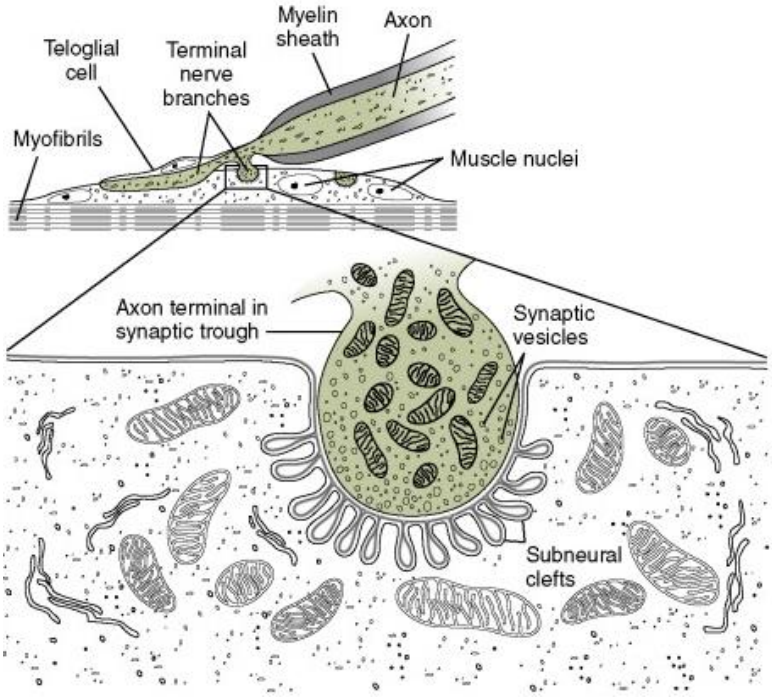



**Figure 3.7: Neuromuscular junction**

# 4.Signal Processing

The background knowledge needed by the reader to understand the designing of filters, analog and digital, that are used for detecting and processing the EMG signal is presented in this chapter. In classical signal processing, the input signal to a system is the summation of information and noise. In this case the signal processing focuses on removing the noise so that the information signal is revealed. Basically, after the processing the signal is analyzed in its two components (information and noise), of which only information is retained. Every signal can be analyzed in the summation of two or more different signals with different ways. There is an infinite number of combinations of signals that, when summed result in the same signa. When there is no additional information these different combinations are totally equivalent. Thus, the problem of analyzing the signal becomes insoluble. In order to end up in a single solution analysis of the signal it becomes necessary to determine additional characteristics of the information signal and the noise signal. Additionally, to the description of a signal in time domain, there is also thee description in the frequency domain. The frequencies a signal contains $X_\alpha(j\Omega)$ of a signal $x_\alpha(t)$ is computed with the help of Fourier Transform.

$$X_\alpha(j\Omega) = \int_{-\infty}^{\infty} x_\alpha(t) e^{-j\Omega t} dt \tag{4.1}$$

Where $\Omega=2\pi f$ is the angular frequency. The function computed shows which frequencies the signal contains and what is the energy distribution among them. In order to discern the information signal from the noise a basic hypothesis is needs to be established and taken for granted. This hypothesis is that the these to signals can be discerned by the frequencies they contain. Thus, if the frequencies that compose these two signals are known then if the frequencies of the noise are removed, then only the only the information signal remains. The frequencies in every kind of signal appear in bandwidths. For example, in the following figure the signal $x_\alpha(t)$ with Fourier transform $X_\alpha(j\Omega)$, is summation of information signal $s_\alpha(t)$, with Fourier Transform $S_\alpha(j\Omega)$ (cyan) and noise $S_\alpha(j\Omega)$ with Fourier Transform $W_\alpha(j\Omega)$ (grey). It can bee observed that the information signal contains frequencies in the bandwidth $[0,\Omega_0]$, while noise in $[\Omega_0,\infty)$. In a general case, every kind of signal can be comprised of more than one bandwidth.

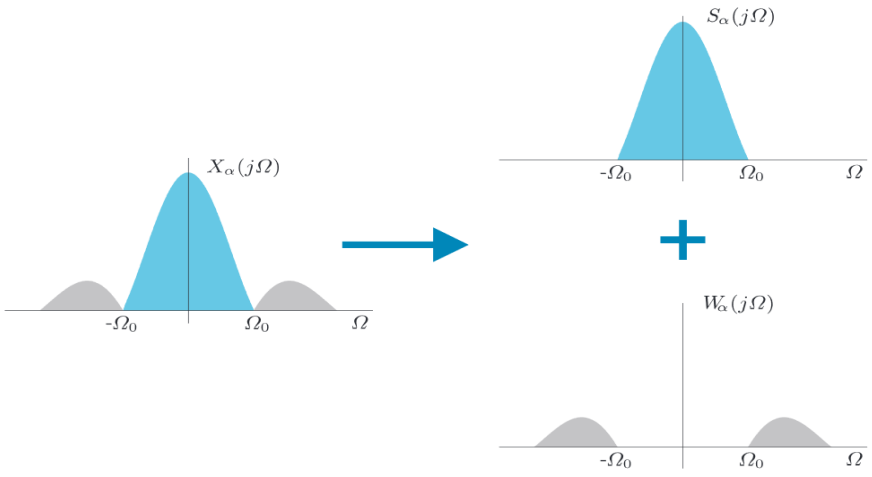



**Figure 4.1: Signal bandwidth**

With the help of bandwidths, the separation of signal $X_\alpha(j\Omega)$ in information signal $S_\alpha(j\Omega)$ and noise signal $W_\alpha(j\Omega)$ is immediate and unique. This analysis is possible with knowledge about only the bandwidths $[0, \Omega_0]$ and $[\Omega_0, \infty)$ (and $x_\alpha(t)$), without the bandwidths $S_\alpha(j\Omega)$ and $W_\alpha(j\Omega)$ to be known, since they are the ones needed to be found.

Of major importance is the understanding of the differences and the relations between analog and digital signals, since in most cases the digital signals result from sampling an analog signal. The greatest difference between a signal of continuous time and a signal of discrete time is that the frequencies of the analog signals take values in range [0, ∞), while the frequencies of the digital in range [0, 1/2]. In order the reason this happens to be understood an example will be given. Let the periodic step function of the following figure be. It is obvious that the frequencies in this signal re due to the changes between positive and negative values.


**Figure 4.2: Analog and digital periodic signals**

In continuous time is always possible to create changes between positive and negative values, no matter how small the period is. Thus, since the period T satisfies the constraint $0<T\leq\infty$, it is deduced that the analog frequency f=1T is between the limits $0\leq f<\infty$. In discrete time the period N, can only be of integer values. Consequently, the smaller period that allows changes of sign is N=2. Thus, in discrete time the period is between the limits $2\leq N\leq\infty$, as a result the corresponding frequency $\lambda=1N$ to be constrained by $0\leq\lambda\leq1/2$. If instead the normal frequencies the angular are used, then the analog frequencies will be symbolized as $\Omega=2\pi f$ and the discrete frequencies will be symbolized as $\omega=2\pi\lambda$. The corresponding limits of the angular frequencies are $0\leq\Omega<\infty$ and $0\leq\omega\leq\pi$. However, the Fourier transform dictates negative frequencies, so the limits are shaped as $-\infty<f,\Omega<\infty$ for continuous signals, while $-1/2<\lambda\leq1/2$ and $-\pi<\omega\leq\pi$ for discrete.

In case of an analog signal $x_\alpha(t)$, that is sampled with regular sampling at times $t_n = nTs$, where n an integer, then a digital signal $x_n$ is generated that satisfies the relation

$$x_n = x_\alpha(nTs). \quad (4.2)$$

Ts is calls sampling period and the inverse of that number fs=1/Ts is sampling frequency. In analog signals frequency measurement units is Hz (cycles/sec). In discrete time signals the frequencies are normalized, the do not have a measurement unit, since they do not refer to real time but to samples. However, when the discrete time signal comes from sampling of analog signal, then it is possible to measure the frequencies in Hz. In this case, the relation between a normalized frequency λ in discrete time and the corresponding f in analog is the following.

$$f = \lambda * fs = \lambda * 1/Ts. \quad (4.3)$$

It is observed that all frequencies f of the sampled signal are in range of (−fs/2, fs/2], since the normalized frequencies are in range (−1/2, 1/2]. In other words, the frequencies found in a sampled signal cannot be greater than the half of sampling frequency.

The above statement leads the need of discussion of the aliasing effect. In case of a signal $x_\alpha(t) = \cos2\pi f_0 t$ with analog frequency $f_0$ after the sampling it is interpreted as $x_n = \cos2\pi f_0 Ts_n$ of digital frequency $\lambda_0 = f_0Ts = f_0fs$. It is obvious that the relation between analog and digital frequency appears to be 1-1. Tha means that there is a change but not a loss in frequency information. With a simple example it will be indicated that this statement is false. In the following figure a sinusoidal signal ( grey) is shown with frequency f=0.8. It is sampled with period $T_s$=1. According to what is

stated so far, the digital signal should be of frequency $\lambda_0 = f_0 Ts = 0.8$. It is also obvious that from the points, that are the samples, another sinusoidal signal with frequency $f'_0 = 0.2$ coincides (cyan). Consequently, the digital sinusoidal can also have the frequency $\lambda'_0 = f'_0 T_s = 0.2$. The right frequency for the digital signal is $\lambda'_0$, since the frequencies in the digital domain cannot exceed 0.5. As a result two different analog sinusoidal signals is possible to have the same sampling. Subsequently digital sinusoidal signal is not capable of representing uniquely the analog. However it is important to define a deterministic relation between analog and discrete frequency.

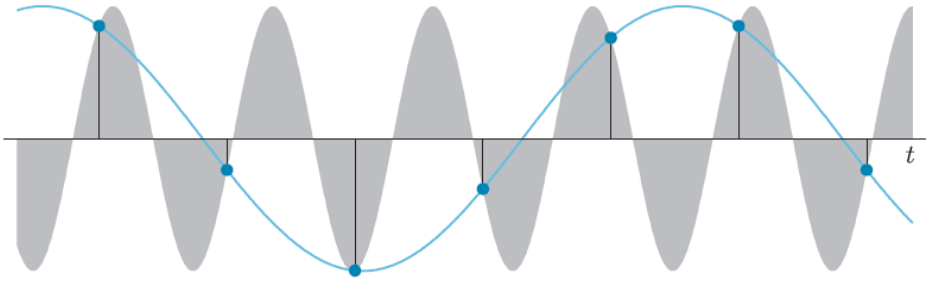



**Figure 4.3 Relation of analog and sampled signal**

Establishing $\lambda_0 = f_0/f_s$ and $\lambda'_0 = f'_0/f_s$ then due to the periodicity of the sinus, the following relation is true

$$x_n = \cos 2\pi(f_0/fs)n = \cos 2\pi(f'_0/fs)n \quad (4.4)$$

where, $f'_0 = f_0 - [f_0/f_s]f_s$, (or in digital frequencies $\lambda'_0 = \lambda_0 - [\lambda_0]$ and the symbol "$[\cdot]$", means the integer part. In other words the higher frequencies are aliased losing the integer multiples of sampling frequency $f_s$, as consequence they appear as low. The aliasing effect does not stop there though. If the integer part of the ratio of the frequency of the signal with the sampling frequency is 0, then aliasing takes effect as follows.

$$\cos 2\pi(f_0/f_s)n = \cos 2\pi(f'_0/f_s)n \quad (4.5), \text{ with}$$
$$f'_0 = f_s - f_0 < f_s/2 \text{ or in digital } \lambda'_0 = 1 - \lambda_0). \quad (4.6)$$

Consequently, when $f_0$ is between $f_s/2$ και $f_s$, then it is aliased so that it is lower than $f_s/2$. Furthermore, this frequency appears as its mirror frequency over $f_s/2$, meaning $f'_0 = f_s - f_0$. As a final result, the digital frequency $\lambda'_0$ of the sinus satisfies the realation $0 \leq \lambda'_0 \leq 0.5$. Based on this phenomenon the Shannon theorem was expressed:

If a signal $x_\alpha(t)$ in continuous time domain that does not contain frequencies higher than $f_m$ it can be reconstructed exactly from the samples $x_n = x_\alpha(nT_s)$, if the sampling frequency satisfies $f_s \geq 2f_m$.

The value $f_s = 2f_m$ is called Nyquist frequency and is the least the possible sampling frequency that permits the exact reconstruction of an analog signal of finite bandwidth, from its samples. Generally, the signal that is sampled can contain noise in frequencies greater than $f_s/2$ so the aliasing effect can alter the frequencies contained in the information signal. To avoid this problem, when sampling is done with $f_s$ then it is mandatory to filter the analog signal from all the frequencies greater than $f_s/2$.

The Basic Hypothesis established above allows the removal of noise with linear, time invariable systems. These systems are called filters, and the corresponding process filtering. There two kinds of filters, corresponding to the kind of signal they refer to. Thus, analog and digital filters exist. The filters that will be used are time invariable and linear, so they can be completely described by their

impulse response h(t) (for analog), and $h_n$ (for digital), or equivalently from the transfer function H(s) and H(z) correspondingly. Of major importance is the behavior of the filters in the frequency domain, thus important role in designing a filter will play the frequency response of it, H(jΩ) (for analog) and H(ejω) (for digital). The specifications and the design of the filters takes place in the frequency domain.

The main differences between the two kind of filters are the following:

- The analog filters are implemented with electrical analog circuits and the processing od the corresponding signals is immediate without delay. On the other hand, the digital filters are implemented with digital processors and the result of the processing is extracted with delay., which is a function of the processing complexity and the processors speed.
- The characteristics of the analog filters is a function of the values of the electric elements of the circuit. These values can vary with temperature, moisture, age, consequently the responses of the filters vary. The digital filters are totally invariable since the processor functions correctly.
- With digital filters it is possible to approach the ideal specifications with any desired precision. Analogous property the analog filters cannot have.
- The digital filters (the finite impulse response specifically) have great flexibility in their design. Practically it is possible to design a filter with any response. The analog filters are limited to only implementation of classical filters only.
- Some filters, like antialiasing and the reconstruction, can be only analog

To begin with, in the analog filters that are implemented with classic electrical circuits the relation between the source (input) and voltage or current in the circuit (output), in the Laplace domain is:

$$\mathcal{H}(s) = \frac{b_0 s^K + b_1 s^{K-1} + \cdots + b_K}{s^L + a_1 s^{L-1} + \cdots + a_L}$$

(4.7)

The transfer function of the circuit is a fraction of polynomial function of variable s. The degree of the polynomial of the denominator L is a parameter of great importance and called the order of the filter. The transfer function can only take this form. Notably, not every fraction of functions of s is implementable with classic circuits.

In the case of digital filters if $h_n$, $x_n$, $y_n$ are the impulse response, the input and the output and H(z), X(z), Y(z) are the transfer function and Z transform of input and output then the following relations are valid:

$$y_n = \sum_{l=-\infty}^{\infty} h_l x_{n-l}$$

$$Y(z) = \mathcal{H}(z) X(z).$$

(4.8) & (4.9)

Prerequisite for the implementation of digital processing is the finite number of calculations per output sample. Two general categories, satisfy this constrain. The first category refers to the

equation, where the impulse response is finite, of the form $h_0, \ldots, h_{L-1}$. Then the output of the filter is:

$$y_n = h_0 x_n + h_1 x_{n-1} + \cdots + h_{L-1} x_{n-L+1} \quad (4.10)$$

These filters are called Finite Impulse Response (FIR) and L parameter is the length of the filter.

The second category is referred to the equation where, H(z) is a fraction of functions of $z^{-1}$.

$$\mathcal{H}(z) = \frac{b_0 + b_1 z^{-1} + \cdots + b_K z^{-K}}{1 + a_1 z^{-1} + \cdots + a_L z^{-L}}$$

(4.11)

Replacing Y (z) = H(z)X(z) we reach to the relation:

$$Y(z) = -a_1 z^{-1} Y(z) - \cdots - a_L z^{-L} Y(z) + b_0 X(z) + \cdots + b_K z^{-K} X(z) \quad (4.12)$$

Using the reverse Z transform on the above equation the relation between input and output is extracted.

$$y_n = -a_1 y_{n-1} - \cdots - a_L y_{n-L} + b_0 x_n + \cdots + b_K x_{n-K} \quad (4.13)$$

L parameter is called the order of the filter. Due to the fact that, in this case the impulse response of such systems is infinite, these filters are called Infinite Impulse Response filters (IIR).

IIR filters are not any linear systems of infinite impulse response, but systems whose impulse response, when K<L, is a fraction of polynomials.

Since filters are systems, of major importance is their causality and their stability. Based on the kind of the filter the constraints are the following:

- FIR filters are always stable.
- The IIR digital filters are stable, when the poles of the transfer function are internally of a unit cycle.

The IIR analog filters are stable when the poles of transfer function are on the negative complex half-plane.

The first step for designing a filter is the definition of the ideal response of it. Since the filters of interest are linear and time invariant, the effect of them on the frequencies that constitute a signal are:
$Y(j\Omega) = D(j\Omega)X(j\Omega)$ for analog,
$Y(e^{j\omega}) = D(e^{j\omega})X(e^{j\omega})$ for digital,

Where X(·) denotes the frequencies contained in the signal to be processed (input), D(·) is the ideal impulse response of the filter and Y (·) denotes the frequencies contained in the output of thee filter. Choosing the ideal response D(·) to be 0 at the frequencies of noise and 1 at the frequencies of the information signal, the desired result is achieved, the removal of noise. An example of a signal with frequencies $X(e^{j\omega})$ and its ideal filter for removing noise $D(e^{j\omega})$ is presented.

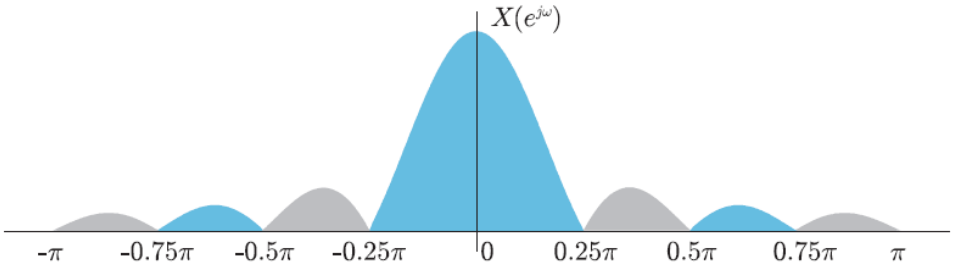



**Figure 4.4 Spectral information**

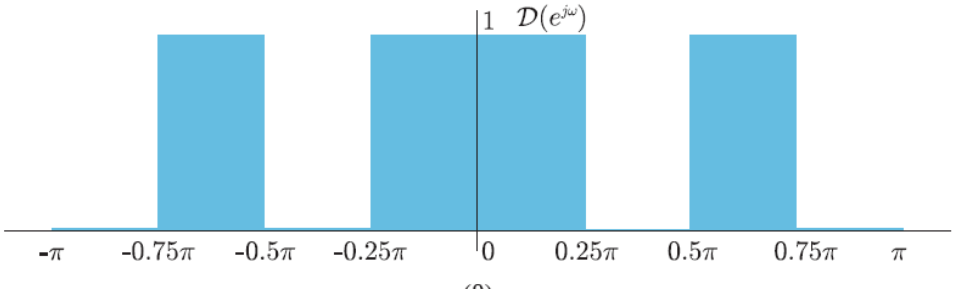



**Figure 4.5 Optimal desired filter**

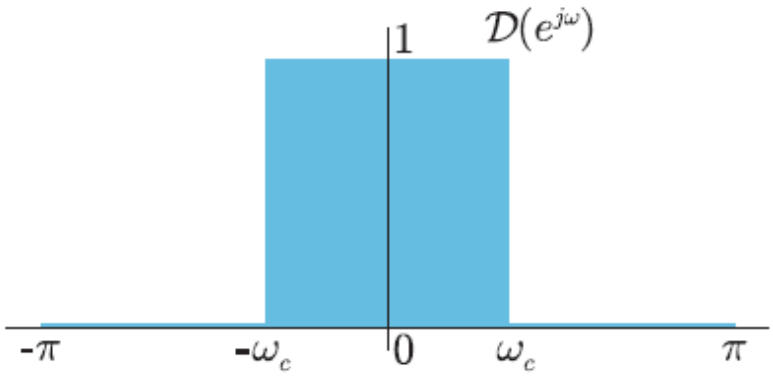
The noise frequenciea are colored grey, while the information is contained in the cyan colored frequencies. In the figure above the ideal freuency response is depicted, which removes the noise completely, through the relation.

$$Y(e^{j\omega}) = D(e^{j\omega})X(e^{j\omega}). \quad (4.14)$$

The ideal filters are usually of window shape, in other words frequency windows that are applied on the frequency bandwidths of the information. Consequently, the ideal specifications of a filter are comprised of the definition of pass bands, cut off bands and the ideal characteristics in the pass bands.

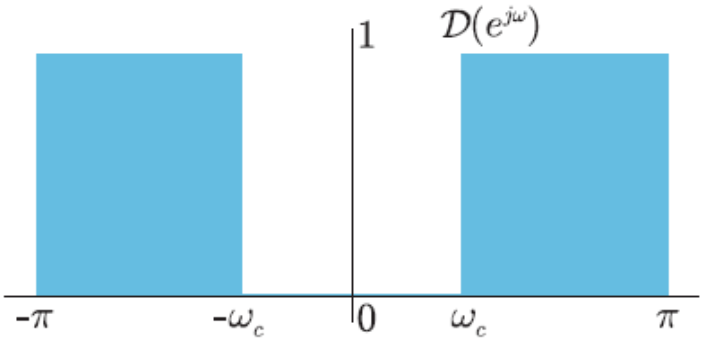
Mentioning the kinds of filters, corresponding to the shape of their ideal impulse response:

Low-Pass Filters:



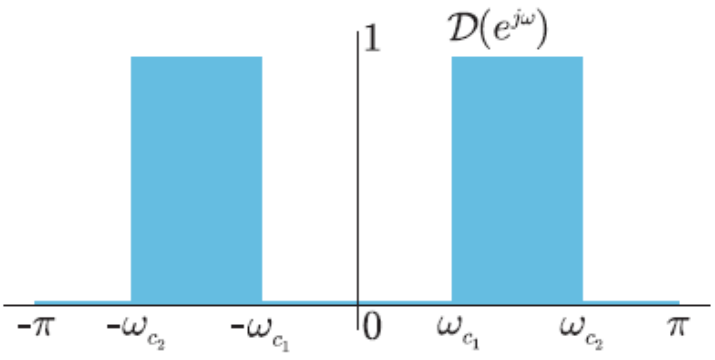

Where $\omega_c$ is the called the cut off frequency.

High-Pass Filters:



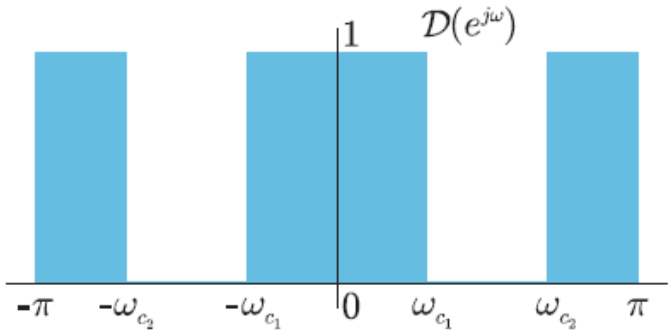

Where $\omega_c$ is the cut-off frequency.

Band-pass Filters:



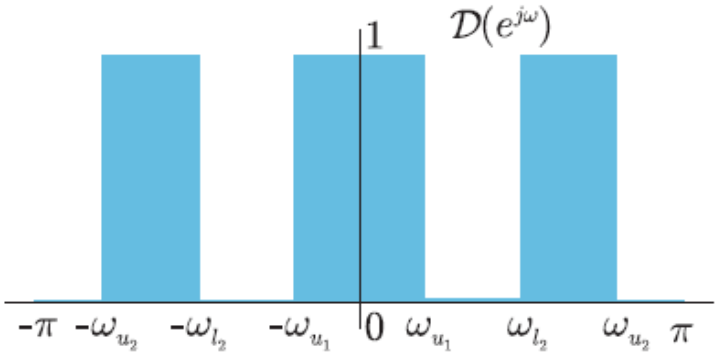

Where $\omega_{c1}$ , $\omega_{c2}$ are the cut-off frequencies.

Stop-Band Filters




Where $\omega_{c1}$ , $\omega_{c2}$ are the cut-off frequencies.

Multi Band-Pass Filters




Where $\omega_{li}$ , $\omega_{ui}$ are the low and the upper cut-off frequency of the i-th pass band.

The ideal impulse response of a filter is obvious from the previous figures that has discontinuities on the cut-off frequencies. The implementation of such a function is impossible so in order the real filters to approach the ideal ones, a third category of bands is introduced. These bands are called transition bands and intermediate between any points of discontinuity of the ideal response. The transition bands are part of the specifications, thus they are defined by designer. However, the designer cannot control the behavior of the function in the transition bands, but only where they start and stop.

The ideal impulse response is approximated with the function $R(e^{j\omega})$, which is the function that will be implemented, so basically it is the filter with transfer function:

$$H(e^{j\omega}) = e^{j\varphi(\omega)}R(e^{j\omega}) \quad (4.15)$$

With $\varphi(\omega)$ the phase difference between the input and output signal. In other words the delay the filter adds as function of the input frequency.

By using $R(e^{j\omega})$ it is not possible to achieve the values defined by the ideal frequency response in the pass-bands and stop-bands. However it it can approach 1or 0 so it is necessary define which approximation is satisfactory. By defining a limit for the absolute value of the error between $D(e^{j\omega})$

and $R(e^{j\omega})$ it is possible to measure when the approximation is satisfactory. Explaining further $R(e^{j\omega})$ is sufficient when the following relation is true:

$$|D(e^{j\omega}) - R(e^{j\omega})| \leq \delta_i \ \gamma\iota\alpha \ \omega_{li} \leq \omega \leq \omega_{ui} \quad (4.16)$$

The index i denotes, to which band of interest the error term refers to.

Instead of defining the maximum tolerable error per band of interest, a weight function $W(\omega)$ can be used for defining the quality if the approximation. This weight function is defined for every frequency in the pass-band or the stop-band. This function satisfies $W(\omega) \geq 1$ and its values is greater at the point we desire the error term to be smaller. The weight function can be accompanied by a quantity $\delta_{max}$, which is the maximum tolerable approximation error. The approximation is considered efficient when,

$$W(\omega)|D(e^{j\omega}) - R(e^{j\omega})| \leq \delta_{max} \quad (4.17)$$

In the analog world only one kind of filter is implementable, the IIR filters, so there is not dilemma on what kind of filter to use. On the other hand, in the digital world, the filters designed can either be FIR or IIR. The FIR filters can be calculated optimally, by minimizing distance criteria between $D(e^{j\omega})$ and $R(e^{j\omega})$. Corresponding methodology for the IIR does not exist. It is not possible to calculate the coefficients of the nominator and denominator polynomials by minimizing a distance criterion. With FIR filters responses of general shape can be designed. The IIR are mainly used in cases of 0-1 filtering. The FIR can be of linear phase without extra cost. The IIR cannot be of linear phase. As far as the number of calculations is concerned, for the same quality of filter the IIR need less calculations than FIR filters.

As mentioned above the output of a FIR filter every moment is

$y_n = h_0x_n + h_1x_{n-1} + \cdot \cdot \cdot + h_{L-1}x_{n-L+1}$

$h_n$ are the coefficients of the filter. Consequently, the problem that need to be solved is the defining of these values so that the absolute difference between the ideal response and the real filter response in minimized.

Let the ideal response be:

$D(e^{j\omega}) = D_r(e^{j\omega}) + jD_i(e^{j\omega})$ (4.18)

Where $D_r(e^{j\omega})$, $jD_i(e^{j\omega})$ are the real and the imaginary part of the $D(ej\omega)$. Due to the symmetry of the conjugate $D(e^{-j\omega}) = D(e^{j\omega})$ it is concluded that $D_r(e^{j\omega})$ is an even while $D_i(e^{j\omega})$ is an odd function of $\omega$.

If $\{h_n\}$ is the sequence of the filter coefficients, then it is possible to define the sequences $\{\alpha_n\}$, $\{\beta_n\}$ with same length as $\{h_n\}$. Explaining further, the coefficients of the filter $h_0, \ldots, h_{L-1}$ are combined with the same coefficients written backwards. The element by element half-sum of the two sequences defines $\{\alpha_n\}$, while the half-difference the $\{\beta_n\}$. Specifically, when $L = 2N + 1$, meaning the length of the filter is an odd number, then the sequences have the form

$$\alpha_N, \ldots, \alpha_1, \alpha_0, \alpha_1, \ldots, \alpha_N \ \kappa\alpha\iota \ \beta_N, \ldots, \beta_1, 0, -\beta_1, \ldots, -\beta_N$$

with the first to be of even symmetry and the second of odd relative to the central element. The elements of the two sequences are given by the relations:

$$\alpha_n = \frac{h_{N-n} + h_{N+n}}{2}, \quad \beta_n = \frac{h_{N-n} - h_{N+n}}{2}, \; n = 0, \ldots, N$$

(4.19) & (4.20)

While the fitler coefficients as a function of $\{\alpha_n\}$, $\{\beta_n\}$ are :

$$h_n = \begin{cases} \alpha_{N-n} + \beta_{N-n} & 0 \le n \le N \\ \alpha_{n-N} - \beta_{n-N} & N+1 \le n \le 2N. \end{cases}$$

(4.21)

In case though of an even L=2N:

$$\alpha_n = \frac{h_{N-n} + h_{N-1+n}}{2}, \quad \beta_n = \frac{h_{N-n} - h_{N-1+n}}{2}, \; n = 1, \ldots, N,$$

(4.22) & (4.23)

$$h_n = \begin{cases} \alpha_{N-n} + \beta_{N-n} & 0 \le n \le N-1 \\ \alpha_{n-N+1} - \beta_{n-N+1} & N \le n \le 2N-1. \end{cases}$$

(4.24)

In the end the filter response for L=2N+1 is:

$$\mathcal{H}(e^{j\omega}) = e^{-jN\omega}\{(\alpha_0 + 2\alpha_1 \cos\omega + \cdots + 2\alpha_N \cos N\omega) + j(2\beta_1 \sin\omega + \cdots + 2\beta_N \sin N\omega)\}$$

(4.25)

Consequently the function $R(e^{j\omega})$ can be described as the sum of real part and an imaginary.

For the case of the odd length L=2N+1:

$$R_r(e^{j\omega}) = \alpha_0 + 2\alpha_1 \cos\omega + \cdots + 2\alpha_N \cos N\omega \quad (4.26)$$

$$R_i(e^{j\omega}) = 2\beta_1 \sin\omega + \cdots + 2\beta_N \sin N\omega \quad (4.27)$$

$$\varphi(\omega) = -N\omega = -((L-1)/2)\omega \quad (4.28)$$

For L=2N:

$$R_r(e^{j\omega}) = 2\alpha_1 \cos 0.5\omega + \cdots + 2\alpha_N \cos(N-0.5)\omega \quad (4.29)$$

$$R_i(ej\omega) = 2\beta_1 \sin 0.5\omega + \cdots + 2\beta_N \sin(N-0.5)\omega \quad (4.30)$$

$$\varphi(\omega) = -(N - 0.5)\omega = -((L - 1)/2\ )\omega \quad (4.31)$$

The characteristics of the FIR filters are:

- The phase $\varphi(\omega)$ is linear and corresponds to latency $(L-1)/2$ number of samples.
- In the case of real ideal frequency response, meaning $D_i(e^{j\omega}) = 0$, the coefficients of the filter have even symmetry and $\beta_n = 0$
- In the case of imaginary ideal frequency response, meaning $D_r(e^{j\omega}) = 0$, the coefficients of the filter have odd symmetry and $a_n = 0$.

Knowing which coefficients need to calculated, the choice of the criterion they need to be calculated with comes in play. The simplest from mathematic perspective, difference between two functions is the mean square error, which is the first criterion to minimize. It is defined as follows:

$$\begin{aligned}\mathscr{E}^2(\alpha_0,\ldots,\alpha_N,\beta_1,\ldots,\beta_N) &= \int_{-\pi}^{\pi} |\mathcal{D}(e^{j\omega}) - \mathcal{R}(e^{j\omega})|^2 d\omega \\ &= \int_{-\pi}^{\pi} \left[\mathcal{D}_r(e^{j\omega}) - \mathcal{R}_r(e^{j\omega})\right]^2 d\omega + \int_{-\pi}^{\pi} \left[\mathcal{D}_i(e^{j\omega}) - \mathcal{R}_i(e^{j\omega})\right]^2 d\omega.\end{aligned}$$

(4.32)

The minimum mean suqare can be calcualted from the minimization of two independent criterions.

$$\mathscr{E}_r^2(\alpha_0,\ldots,\alpha_N) = \int_{-\pi}^{\pi} \left[\mathcal{D}_r(e^{j\omega}) - \mathcal{R}_r(e^{j\omega})\right]^2 d\omega$$
$$\mathscr{E}_i^2(\beta_1,\ldots,\beta_N) = \int_{-\pi}^{\pi} \left[\mathcal{D}_i(e^{j\omega}) - \mathcal{R}_i(e^{j\omega})\right]^2 d\omega,$$

(4.33) & (4.34)

And the following relation can be stated:

$$\min_{\alpha_0,\ldots,\alpha_N,\beta_1,\ldots,\beta_N} \mathscr{E}^2(\alpha_0,\ldots,\alpha_N,\beta_1,\ldots,\beta_N) = \min_{\alpha_0,\ldots,\alpha_N} \mathscr{E}_r^2(\alpha_0,\ldots,\alpha_N) + \min_{\beta_1,\ldots,\beta_N} \mathscr{E}_i^2(\beta_1,\ldots,\beta_N)$$

(4.35)

In order to calculate the optimal coefficient by deriving very error with respect to its corresspponding parameters and equate to 0. This process results to the following equations for the optimal parameters.

For L=2N+1

$$\alpha_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathcal{D}_r(e^{j\omega})\cos n\omega\, d\omega, \quad n = 0, \ldots, N$$
$$\beta_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathcal{D}_i(e^{j\omega})\sin n\omega\, d\omega, \quad n = 1, \ldots, N,$$

(4.36) & (4.37)

While for L=2N

$$\alpha_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathcal{D}_r(e^{j\omega})\cos (n-0.5)\omega\, d\omega, \quad n = 1, \ldots, N.$$
$$\beta_n = \frac{1}{2\pi}\int_{-\pi}^{\pi} \mathcal{D}_i(e^{j\omega})\sin (n-0.5)\omega\, d\omega, \quad n = 1, \ldots, N.$$

(4.38) & (4.39)

Although this method is simple, it has a great drawback. In the response of the filter appear irreducible waveforms no matterr the order of the filter. An example that will illustrate problem is given. This problem is known as Gibbs phenomenon. Lets say a low-pass filter has cut-off frequency $\omega_c$=0,6π. Thus, thee ideal filter response is defined as:

$$\mathcal{D}(e^{j\omega}) = \begin{cases} 1 & -0.6\pi \le \omega \le 0.6\pi \\ 0 & \text{αλλού,} \end{cases}$$

(4.40)

The fitler coefficients are calculated as mentiond above. The ideal response is real so the $R(e^{j\omega})$ has to be too, thus $\beta_n$ =0. In the following figure the responses of the filter for orders L=11, 21,51 and the ideal based on which it was designed are presented.

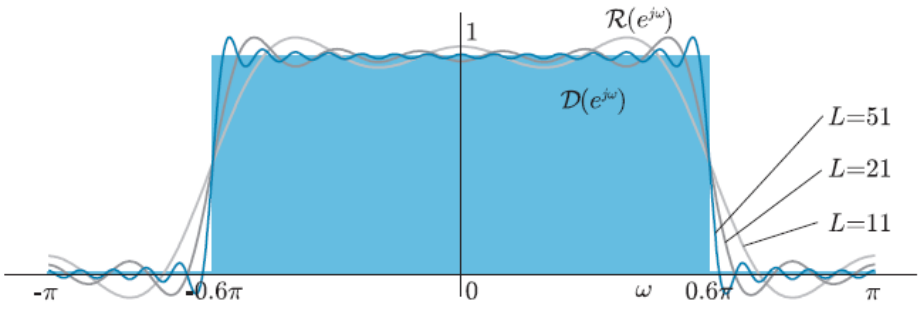



**Figure 4.6: Gibbs Phenomenon**

The Gibbs phenomenon appears in all the cases of filter designing, where the ideal frequency response $D(e^{j\omega})$ has discontinuities. The phenomenon Gibbs derives from the fact that the finite sequence is created by the multiplication, element by element, of the infinite sequence with a rectangle window. This phenomenon is eliminated by using alternative windows. Consequently, if $h_0, h_1, \ldots, h_{L-1}$ are thee optimal coefficients and are calculated with the method described and $\varpi_0, \varpi_1, \ldots, \varpi_{L-1}$ are the coefficients of the window that will be applied, then the final coefficients of the filter will be :

$$h^{o}_{n} = \varpi_n h_n,\ n = 0, \ldots, L-1. \quad (4.41)$$

In the following figure the aforementioned example is given after the application of a Hamming window. The waveform appears to be diminished, though the transition from 1 to 0 is less steep comparing to the case of the rectangle window. This is the price for diminishing Gibbs phenomenon.

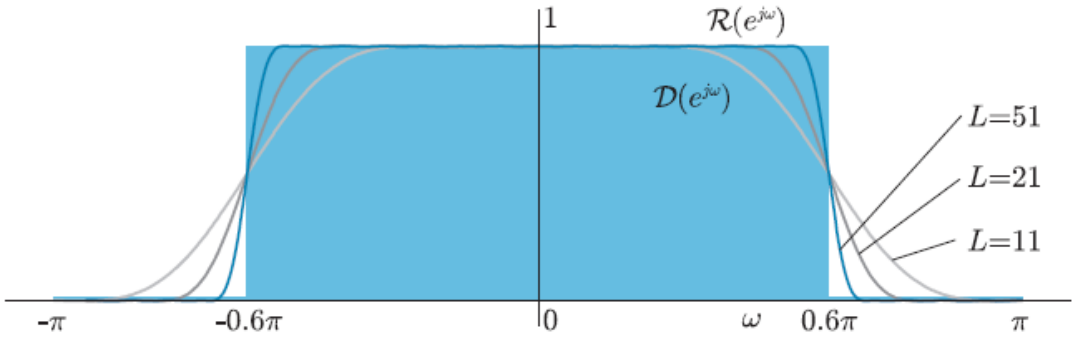



**Figure 4.7: Gibbs phenomenon elimination**

A variation of this method is the designing of the filters with the use of the practical specifications instead of the ideal. If T denotes the union of bands of interest, pass-band and stop-band, then the ideal response $D(e^{j\omega})$ is fully defined in this ensemble. Thus, the mean square error can be defined by limiting the points of integration to T without caring for the transition bands.

$$\bar{\mathscr{E}}^2(\alpha_0, \ldots, \alpha_N, \beta_1, \ldots, \beta_N) = \int_{\mathscr{T}} W^2(\omega)|\mathcal{D}(e^{j\omega}) - \mathcal{R}(e^{j\omega})|^2 d\omega$$
$$= \int_{\mathscr{T}} W^2(\omega)\left[\mathcal{D}_r(e^{j\omega}) - \mathcal{R}_r(e^{j\omega})\right]^2 d\omega + \int_{\mathscr{T}} W^2(\omega)\left[\mathcal{D}_i(e^{j\omega}) - \mathcal{R}_i(e^{j\omega})\right]^2 d\omega.$$

(4.42)

Similarly, with the previous method, the minimization of thee criterion is done by minimizing two independent criterions for the real and the imaginary part.

$$\min_{\alpha_0,\ldots,\alpha_N,\beta_1,\ldots,\beta_N} \bar{\mathscr{E}}^2(\alpha_0, \ldots, \alpha_N, \beta_1, \ldots, \beta_N) =$$
$$\min_{\alpha_0,\ldots,\alpha_N} \bar{\mathscr{E}}_r^2(\alpha_0, \ldots, \alpha_N) + \min_{\beta_1,\ldots,\beta_N} \bar{\mathscr{E}}_i^2(\beta_1, \ldots, \beta_N)$$

(4.43)

The minimization problems defined with the previous relation have explicit solution and the coefficients are calculated by solving two linnear simultaneous equations. By deriving every criterion with respect to its parameters and equate with 0, then the following equations result for the case of L=2N+1.

For $\{\alpha_n\}$ coefficients:

$$\int_{\mathscr{T}} \mathcal{W}^2(\omega)\mathcal{D}_r(e^{j\omega}) \cos n\omega \, d\omega =$$
$$\alpha_0 \int_{\mathscr{T}} \mathcal{W}^2(\omega) \cos n\omega \, d\omega + 2\alpha_1 \int_{\mathscr{T}} \mathcal{W}^2(\omega) \cos\omega \cos n\omega \, d\omega$$
$$+ \cdots + 2\alpha_N \int_{\mathscr{T}} \mathcal{W}^2(\omega) \cos N\omega \cos n\omega \, d\omega, \; n = 0, \ldots, N$$

(4.44)

For $\{\beta_n\}$ coefficients:

$$\int_{\mathscr{T}} \mathcal{W}^2(\omega)\mathcal{D}_i(e^{j\omega}) \sin n\omega \, d\omega = 2\beta_1 \int_{\mathscr{T}} \mathcal{W}^2(\omega) \sin\omega \sin n\omega \, d\omega$$
$$+ \cdots + 2\beta_N \int_{\mathscr{T}} \mathcal{W}^2(\omega) \sin N\omega \sin n\omega \, d\omega, \; n = 1, \ldots, N.$$

(4.45)

In the case of L=2N, $\{\alpha_n\}$ coefficients:

$$\int_{\mathscr{T}} \mathcal{W}^2(\omega)\mathcal{D}_r(e^{j\omega}) \cos (n-0.5)\omega \, d\omega = 2\alpha_1 \int_{\mathscr{T}} \mathcal{W}^2(\omega) \cos 0.5\omega \cos (n-0.5)\omega \, d\omega$$
$$+ \cdots + 2\alpha_N \int_{\mathscr{T}} \mathcal{W}^2(\omega) \cos (N-0.5)\omega \cos (n-0.5)\omega \, d\omega, \; n = 1, \ldots, N.$$

(4.46)

For $\{\beta_n\}$ coefficients:

$$\int_{\mathscr{T}} \mathcal{W}^2(\omega)\mathcal{D}_i(e^{j\omega}) \sin (n-0.5)\omega \, d\omega = 2\beta_1 \int_{\mathscr{T}} \mathcal{W}^2(\omega) \sin 0.5\omega \sin (n-0.5)\omega \, d\omega$$
$$+ \cdots + 2\beta_N \int_{\mathscr{T}} \mathcal{W}^2(\omega) \sin (N-0.5)\omega \sin (n-0.5)\omega \, d\omega, \; n = 1, \ldots, N.$$

(4.47)

The methods used so far took the filters size for granted, thus there was no control over the error of the filter. However, using specifications for the error term on the pass and stop bands, $\delta_p$ then $\delta_s$, through the following relation the order of the filter can be calculated.

$$L \cong \frac{-20\log_{10}(\sqrt{\frac{\delta_p}{4}\frac{\delta_s}{4}}) - 13}{14.6\frac{|\omega_s - \omega_p|}{2\pi}} = \frac{-20\log_{10}(\sqrt{\delta_p\delta_s}) - 1}{14.6\frac{|\omega_s - \omega_p|}{2\pi}}$$

(4.48)

Moving on to the IIR filters, they do not have flexibility in their design with optimal output minimizing a criterion. This is due to the fact that, when someone tries to define the coefficients optimally, then non-linear problems arise, consequently the optimizing algorithms do not converge always. Thus, the methods used for designing IIR filters are not optimal, but this not a practical drawback, since IIR filters are of a low order generally. The design of any IIR is deduced from the design of an analog IIR low-pass filter and then through transforms of the function the desired filter is extracted. An analog IIR filter has a transfer function of the form:

$$\mathcal{H}(s) = \frac{b_0 s^K + b_1 s^{K-1} + \cdots b_K}{s^L + a_1 s^{L-1} + \cdots a_L}$$

(4.49)

It is necessary to define the general specifications of such a filter:

- $[0, \ \Omega_p]$ is the pass band
- $[\ \Omega_s \ \infty]$ is the stop band
- $\delta_p$ is the maximum tolerable error in the pass band
- $\delta_s$ is the maximum tolerable error in the stop band

The goal is to approach the ideal response with the function $R(j\Omega) = |H(j\Omega)|$, ensuring that the filter is stable. According to the form of the function there are four basic analog IIR filters, Butterwoth, Chebyshev of type I&II, elliptical filters.

The Butterworth filter has the amplitude response:

$$|\mathcal{H}(j\Omega)|^2 = \frac{1}{1 + \left(\frac{\Omega}{\Omega_c}\right)^{2L}}$$

(4.50)

$$\Omega_c = \Omega_p^{\frac{\log(a_s)}{\log(\frac{a_s}{a_p})}} \Omega_s^{\frac{\log(a_p)}{\log(\frac{a_p}{a_s})}}, \quad L = \left\lceil \frac{\log(\frac{a_s}{a_p})}{2\log(\frac{\Omega_s}{\Omega_p})} \right\rceil$$

(4.51) & (4.52)

$a_p = 1/(1 - \delta_p)^2 - 1$, $a_s = (1/\delta^2_s) - 1$.

The Chebyshev type 2 have the amplitude response:

$$|\mathcal{H}(j\Omega)|^2 = \frac{1}{1 + \frac{1}{\epsilon^2 T_L^2(\frac{\Omega_c}{\Omega})}}$$

(4.53)

The elliptical filters have the amplitude response:

$$|\mathcal{H}(j\Omega)|^2 = \frac{1}{1+\epsilon^2 J_L^2(\frac{\Omega}{\Omega_c})}$$

(4.54)

The Chebyshev type 1 have the amplitude response:

$$|\mathcal{H}(j\Omega)|^2 = \frac{1}{1+\epsilon^2 T_L^2(\frac{\Omega}{\Omega_c})}$$

(4.55)

$$T_k(x) = \begin{cases} \cos(k\cos^{-1}(x)) & \gamma\iota\alpha\ |x| \leq 1 \\ \cosh(k\cosh^{-1}(x)) & \gamma\iota\alpha\ |x| > 1 \end{cases}$$

(4.56)

The filter ‘s transfer function is:

$$\mathcal{H}(s) = \frac{\frac{\Omega_c^L}{\epsilon 2^{L-1}}}{(s-s_0)\cdots(s-s_{L-1})}$$

(4.57)

The poles of the system is

$s_n = \sigma_n + j\Omega_n$, n = 0, . . . ,L – 1 (4.58)

where $\sigma_n = A_c \cos(\frac{\pi}{2} + \frac{(2n+1)\pi}{2L})$ and $\Omega_n = B_c \sin(\frac{\pi}{2} + \frac{(2n+1)\pi}{2L})$

However, the values for $\epsilon$, $\Omega_c$ and L need to be defined.
For $\Omega > \Omega_c$ the response is genuinely decreasing. For $\Omega = \Omega c$:

$|H(j\Omega_c)| = 1/(1+\epsilon^2)^{1/2}$ , (4.59)

Which is the smaller value of the amplitude response in the range $[0, \Omega_c]$. This means that $[0, \Omega_c]$ is pass band, which leads to the choice $\Omega_c = \Omega_p$.

The term ϵ is calculated through the relation:

$$\epsilon = \sqrt{\frac{1}{(1-\delta p)^{\wedge}2} - 1}$$

(4.60)

From the monotony of the amplitude response for Ω > Ωc, consequently the constraint of the maximum tolerable error is satisfied for every Ω in the stop band since Ω = $\Omega_s$. Subsequently it is necessary the following expression to be true:

$$\frac{1}{1+\epsilon^2 T_L^2(\frac{\Omega_s}{\Omega_p})} \leq \delta_s^2$$

(4.61)

The order of the filter is calculated from:

$$L = \left\lceil \frac{\cosh^{-1}\left(\frac{1}{\epsilon}\sqrt{\frac{1}{\delta_s^2}-1}\right)}{\cosh^{-1}(\frac{\Omega_s}{\Omega_p})} \right\rceil$$

(4.62)

So, the transfer function of the IIR filter is completely defined.

In this diploma thesis a specific category of IIR fitler was designed. The digital notch filter. This kind of filter tries to remove only one frequency. In reality frequencies in a very narrow area around the desired frequency to eliminate. In this case the unwanted frequency is the 50 Hz. The desired transfer function needs to have the behavior:

$$H(e^{j\omega}) = \begin{cases} 0, & \omega_0 \\ 1, & otherwise \end{cases}$$

Implementing it with 2nd order transfer function it takes the form:

$$H(z) = \frac{(z-e^{j\omega_0})(z-e^{-j\omega_0})}{(z-re^{j\omega_0})(z-re^{-j\omega_0})}$$

(4.63)

The unit circle is selected for the frequency domain of a cycle, in order to prevent the filter coefficients to appear complex, must be in plane fourth quadrantal
symmetry position in response to the conjugate zero $e^{-j\omega 0}$ and $re^{-j\omega 0}$ conjugate poles. The pole placement in zero radial distance from the origin of R, therefore 0 ≤ r<1. R, pole is close to the unit circle, the frequency response in deeper, narrower in width. In the zero-point, frequency response appeared minima, said system " Valley ", in the pole frequency response has a maximum value, the " peak " point representation system. Valley point frequency cutoff, peak frequency through. According to the frequency response of the zero and pole allocation, the reverse design of notch filter. Let z= $e^{j\omega}$, when ω=$\omega_0$, infinite attenuation. Using Euler formula $e^{jx}$=cosx+jsinx the transfer function is simplified as:

$$H(z)=\frac{1-2\cos\omega_0 z^{-1}+z^{-2}}{1-2r\cos\omega_0 z^{-1}+r^2 z^{-2}}$$

(4.64)

By equating the following general form of the transfer function of a 2nd order IIR

$$H(z)=\frac{Y(z)}{X(z)}=\frac{b_0+b_1z^{-1}+b_2z^{-2}}{1-a_1z^{-1}-a_2z^{-2}}$$

(4.65)

With the simplified version and using reverse Z transform the output of the filter is in samples is:

$$y(n)=a_1y(n-1)+a_2y(n-2)+b_0x(n)+b_1x(n-1)+b_2x(n-2)$$

(4.66)

The following figure shows how the parameter r affects the transfer function.

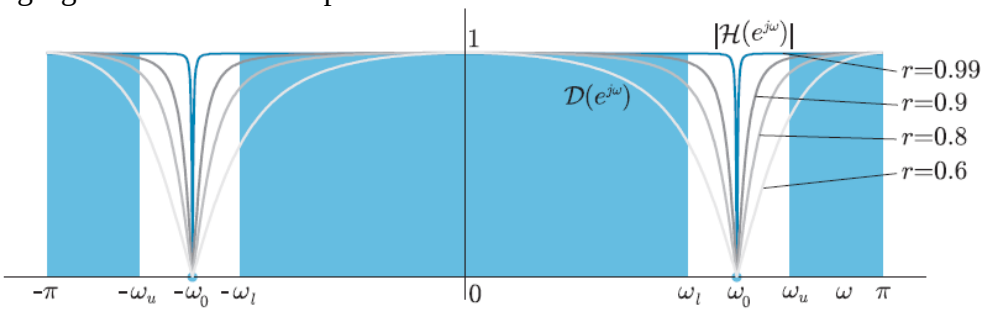



**Figure 4.8: Notch filter r parameter**

Based on the theory explained in this chapter the FIR and IIR filters, digital and analog, required for the whole process are designed.

# 5.Support Vector Machines

The goal of the diploma thesis is the recognition of different gestures done by the user from 4 channels of EMG signal. The pattern recognition algorithm used for this purpose is the Support Vector Machines. An analytic explanation of the algorithm and the math involved is presented in this chapter so that, the reader has an understanding of how this method works. The Support Vector Machine (SVM) is a classification method introduced in 1992 by Boser, Guyon, and Vapnik. The SVM classifier is widely used in bioinformatics (and other disciplines) due to its high accuracy, ability to deal with high-dimensional data and flexibility in modeling diverse sources of data. SVMs belong to the general category of kernel methods. A kernel method is an algorithm that depends on the data only through dot-products. When this is the case, the dot product can be replaced by a kernel function which computes a dot product in some possibly high dimensional feature space. This has two advantages: First, the ability to generate non-linear decision boundaries using methods designed for linear classifiers. Second, the use of kernel functions allows the user to apply a classifier to data that have no obvious fixed-dimensional vector space representation. Using SVMs effectively requires an understanding of how they work. When training an SVM the practitioner needs to make a number of decisions: how to preprocess the data, what kernel to use, and finally, setting the parameters of the SVM and the kernel.

Support vector machines are an example of a linear two-class classifier. The data for a two-class learning problem consists of objects labeled with one of two labels corresponding to the two classes, for convenience we assume the labels are +1(positive examples) or−1 (negative examples). In what follows bold face **x** denotes a vector with components $x_i$. The notation $x_i$ will denote the $i^{th}$ vector in a dataset$\{(x_i,y_i)\}_n^{i=1}$, where $y_i$ is the label associated with $x_i$. The objects $x_i$ are called patterns or examples. We assume the examples belong to some set X. Initially we assume the examples are vectors, but once we introduce kernels this assumption will be relaxed, at which point they could be any continuous/discrete object. A key concept required for defining a linear classifier is the dot product between two vectors, also referred to as an inner productor scalar product, defined as $w^Tx=\sum_i w_i x_i$. A linear classifier is based on a linear discriminant function of the form

$$f(x) = w^Tx+b. \quad (5.1)$$

The vector w is known as the weight vector, and b is called the bias. Consider the case b= 0 first. The set of points x such that $w^Tx= 0$ are all points that are perpendicular to w and go through the origin — a line in two dimensions, a plane in three dimensions, and more generally, a hyperplane. The bias b translates the hyperplane away from the origin. The hyperplane

$$\{x:f(x) = w^Tx+b= 0\} \quad (5.2)$$

divides the space into two: the sign of the discriminant function f(x) denotes the side of the hyperplane a point is on. The boundary between regions classified as positive and negative is called the decision boundary of the classifier. The decision boundary defined by a hyperplane is said to be linear because it is linear in the input examples. A classifier with a linear decision boundary is called a linear classifier. Conversely, when the decision boundary of a classifier depends on the data in a non-linear way the classifier is said to be non-linear.

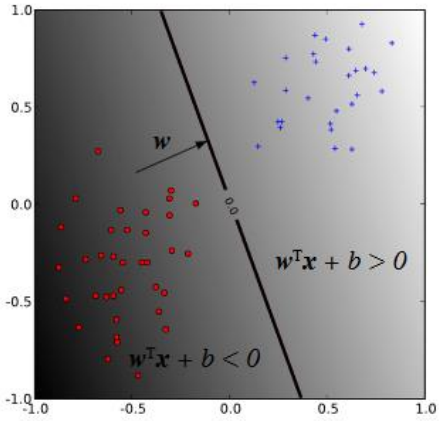



**Figure 5.1: A linear classifier. The decision boundary (points x such that $w^T x+b= 0$) divides the plane into two sets depending on the sign of $w^T x+b$.**

In many applications a non-linear classifier provides better accuracy. And yet, linear classifiers have advantages, one of them being that they often have simple training algorithms that scale well with the number of examples. The naive way of making a non-linear classifier out of a linear classifier is to map our data from the input space X to a feature space F using a non-linear function $\varphi: X \rightarrow F$. In the space F the discriminant function is:

$$f(x) = w^T\varphi(x) + b \quad (5.3)$$

To illustrate the situation better consider the case of a two-dimensional input-space with the mapping

$$\varphi(x) = (x^2_1, \sqrt{2}x_1x_2, x^2_2)^T \quad (5.4)$$

which represents a vector in terms of all degree-2 monomials. In this case

$$w^T\varphi(x) = w_1x^2_1 + \sqrt{2}w_2x_1x_2 + w_3x^2_2, \quad (5.5)$$

resulting in a decision boundary for the classifier, $f(x) = w^Tx+b= 0$, which is a conic section (e.g., an ellipse or hyperbola).

The approach of explicitly computing non-linear features does not scale well with the number of input features: when applying the mapping from the above example the dimensionality of the feature space F is quadratic in the dimensionality of the original space. This results in a quadratic increase in memory usage for storing the features and a quadratic increase in the time required to compute the discriminant function of the classifier. This quadratic complexity is feasible for low dimensional data, but when handling data that can have thousands of dimensions, quadratic complexity in the number of dimensions is not acceptable. Kernel methods solve this issue by avoiding the step of explicitly mapping the data to a high dimensional feature-space. Suppose the weight vector can be expressed as a linear combination of the training examples, i.e. $w=\sum^n_{i=1}\alpha_i x_i$. Then:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i^T \mathbf{x} + b. \quad (5.6)$$

In the feature space, F this expression takes the form:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i \phi(\mathbf{x}_i)^\mathsf{T} \phi(\mathbf{x}) + b.$$

(5.7)

The representation in terms of the variables $\alpha_i$ is known as the dual representation of the decision boundary. As indicated above, the feature space F may be high dimensional, making this trick impractical unless the kernel function k(x,x′) defined as

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^\mathsf{T} \phi(\mathbf{x}')$$

(5.8)

can be computed efficiently. In terms of the kernel function the discriminant function is:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b\,.$$

(5.9)

the kernel can be computed without explicitly computing the mapping φ.

One of the kernels that is widely used is the degree-d polynomial kernel:

$$k(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\mathsf{T} \mathbf{x}' + 1)^d.$$

(5.10)

The feature space for this kernel consists of all monomials up to degree d, i.e. features of the form: $x_1^{d_1} x_2^{d_2} \cdots x_m^{d_m}$ where $\sum_{i=1}^{m} d_i \leq d$. The kernel with d= 1 is the linear kernel, and in that case the additive constant in the above equation is usually omitted. The flexibility of the classifier is increased as the degree of the polynomial is increased. The other widely used kernel is the Gaussian kernel or Radial basis function is defined by:

$$k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma ||\mathbf{x} - \mathbf{x}'||^2)$$

(5.11)

Where γ >0 is a parameter that controls the width of Gaussian. It plays a similar role as the degree of the polynomial kernel in controlling the flexibility of the resulting classifier.

In what follows the term linearly separable is used to denote data for which there exists a linear decision boundary that separates positive from negative example. Initially linearly separable data is assumed, and later is indicated how to handle data that is not linearly separable. Firstly, the notion of margin needs to be defined. For a given hyperlane with $x_+$($x_-$) is denoted the closest point to the hyperpalne among the positive (negative) examples. The norm of a vector w denoted by ||w|| is its length, and is given by $\sqrt{\mathbf{w}^\mathsf{T}\mathbf{w}}$. A unit vector $\hat{\mathbf{w}}$ in the direction of w is given by w/||w|| and has $||\hat{\mathbf{w}}||$= 1. From simple geometric considerations the margin of a hyperplane f with respect to a dataset D can be seen to be:

$$m_{\mathcal{D}}(f) = \frac{1}{2} \hat{\mathbf{w}}^\mathsf{T} (\mathbf{x}_+ - \mathbf{x}_-)$$

(5.12)

Where $\hat{w}$ is a unit vector in the direction of w, and we assume that $x_+$ and $x_-$ are equidistant from the decision boundary i.e.

$$\begin{aligned} f(\mathbf{x}_+) &= \mathbf{w}^\mathsf{T}\mathbf{x}_+ + b = a \\ f(\mathbf{x}_-) &= \mathbf{w}^\mathsf{T}\mathbf{x}_- + b = -a \end{aligned}$$

(5.13) & (5.14)

for some constant a >0. Note that multiplying the data points by a fixed number will increase the margin by the same amount, whereas in reality, the margin hasn't really changed — we just changed the "units" with which we measure it. To make the geometric margin meaningful the value of the decision function is fixed at the points closest to the hyperplane, and set a= 1 in the above equations. Adding the two equations and dividing by ||w|| we obtain:

$$m_{\mathcal{D}}(f) = \frac{1}{2}\hat{\mathbf{w}}^\mathsf{T}(\mathbf{x}_+ - \mathbf{x}_-) = \frac{1}{||\mathbf{w}||}\,.$$

(5.15)

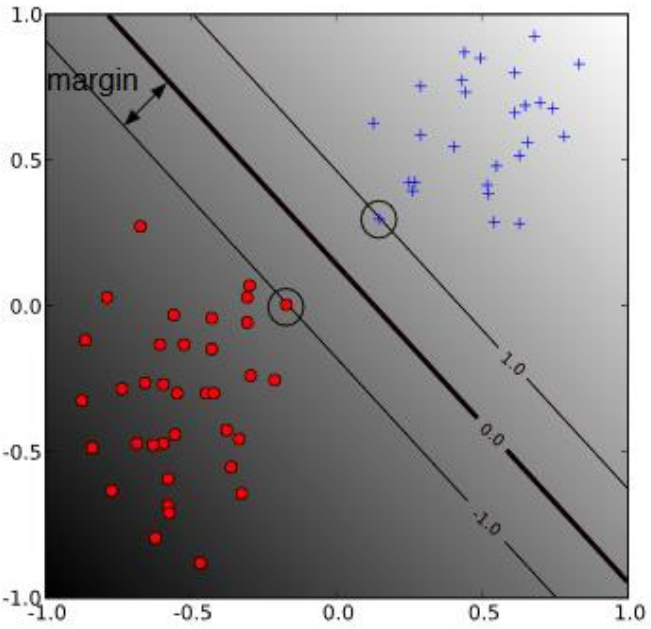



**Figure 5.2: A linear SVM. The circled data points are the support vectors—the examples that are closest to the decision boundary. They determine the margin with which the two classes are separated**.

With the concept of margin established the maximum margin classifier can be formulated. Firstly, the hard margin SVM is defined, applicable to a linearly separable dataset, and then modified to handle non-separable data. The maximum margin classifier is the discriminant function that maximizes the geometric margin 1/||w||which is equivalent to minimizing||w||2. This leads to the following constrained optimization problem:

$$\begin{aligned} &\underset{\mathbf{w},b}{\text{minimize}} && \tfrac{1}{2}||\mathbf{w}||^2 \\ &\text{subject to:} && y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 \quad i = 1, \ldots, n\,. \end{aligned}$$

(5.16)

The constraints in this formulation ensure that the maximum margin classifier classifies each example correctly, which is possible since we assumed that the data is linearly separable. In practice, data is often not linearly separable; and even if it is, a greater margin can be achieved by allowing the classifier to misclassify some points. To allow errors we replace the inequality constraints in the above equation with:

$$y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 - \xi_i \quad i = 1, \ldots, n$$

(5.17)

Where $\xi_i \geq 0$ are slack variables that allow an example to be in the margin ($0 \leq \xi_i \leq 1$, also called a margin error) or to be misclassified ($\xi_i > 1$). Since an example is misclassified if the value of its slack variable is greater than 1, $\sum_i \xi_i$ is a bound on the number of misclassified examples. Our objective of maximizing the margin, i.e. minimizing $(1/2)\|w\|^2$ will be augmented with a term $C\sum_i \xi_i$ to penalize misclassification and margin errors. The optimization problem now becomes:

$$\begin{aligned} \underset{\mathbf{w},b}{\text{minimize}} \quad & \tfrac{1}{2}||\mathbf{w}||^2 + C\sum_{i=1}^{n} \xi_i \\ \text{subject to:} \quad & y_i(\mathbf{w}^\mathsf{T}\mathbf{x}_i + b) \geq 1 - \xi_i, \quad \xi_i \geq 0. \end{aligned}$$

(5.18)

The constant C >0 sets the relative importance of maximizing the margin and minimizing the amount of slack. This formulation is called the soft-margin SVM, and was introduced by Cortes and Vapnik. Using the method of Lagrange multipliers, we can obtain the dual formulation which is expressed in terms of variables $\alpha_i$:

$$\begin{aligned} \underset{\alpha}{\text{maximize}} \quad & \sum_{i=1}^{n} \alpha_i - \tfrac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n} y_i y_j \alpha_i \alpha_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j \\ \text{subject to:} \quad & \sum_{i=1}^{n} y_i \alpha_i = 0, \quad 0 \leq \alpha_i \leq C. \end{aligned}$$

(5.19)

The dual formulation leads to an expansion of the weight vector in terms of the input examples:

$$\mathbf{w} = \sum_{i=1}^{n} y_i \alpha_i \mathbf{x}_i.$$

(5.20)

The examples $x_i$ for which $\alpha_i > 0$ are those points that are on the margin, or within the margin when a soft-margin SVM is used. These are the so-called support vectors. The expansion in terms of the support vectors is often sparse, and the level of sparsity (fraction of the data serving as support vectors) is an upper bound on the error rate of the classifier. The dual formulation of the SVM optimization problem depends on the data only through dot products. The dot product can therefore be replaced with a non-linear kernel function, there by performing large margin separation in the feature-space of the kernel. The SVM optimization problem was traditionally solved in the dual formulation, and only recently it was shown that the primal formulation, can lead to efficient kernel-based learning.

Training an SVM finds the large margin hyperplane, i.e. sets the parameters $a_i$ and b. The SVM has another set of parameters called hyperparameters: The soft margin constant, C, and any parameters the kernel function may depend on (width of a Gaussian kernel or degree of a polynomial kernel).

In this section the effects of the hyperparameters on the decision boundary of an SVM using two-dimensional examples is illustrated through. The soft margins role is illustrated the figures below. For a large value of C a large penalty is assigned to errors/margin errors. This is seen in the left panel of following figure, where the two points closest to the hyperplane affect its orientation, resulting in a hyperplane that comes close to several other data points. When C is decreased (right panel of the figure), those points become margin errors, the hyperplane's orientation is changed, providing a much larger margin for the rest of the data. Kernel parameters also have a significant effect on the decision boundary. The degree of the polynomial kernel and the width parameter of the Gaussian kernel control the flexibility of the resulting classifier. The lowest degree polynomial is the linear kernel, which is not sufficient when a non-linear relationship between features exists.

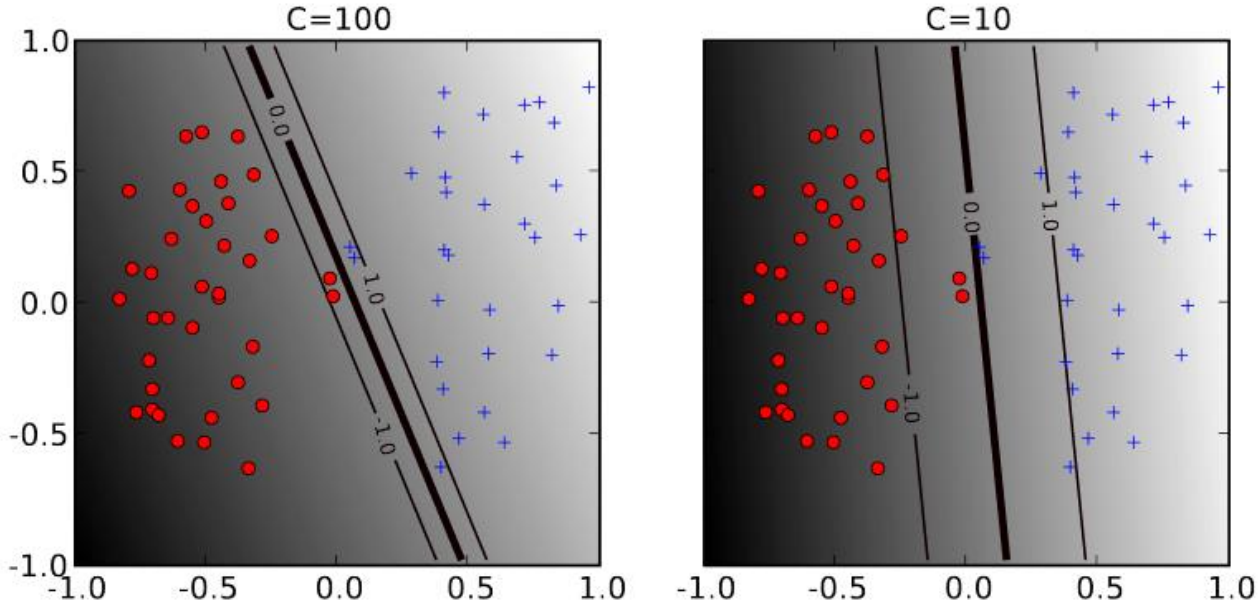



**Figure 5.3: The effect of the soft-margin constant, C, on the decision boundary. A smaller value of C(right) allows to ignore points close to the boundary, and increases the margin. The decision boundary between negative examples (red circles) and positive examples (blue crosses) is shown as a thick line. The lighter lines are on the margin (discriminant value equal to -1 or +1). The grayscale level represents the value of the discriminant function, dark for low values and a lightshade for high values.**

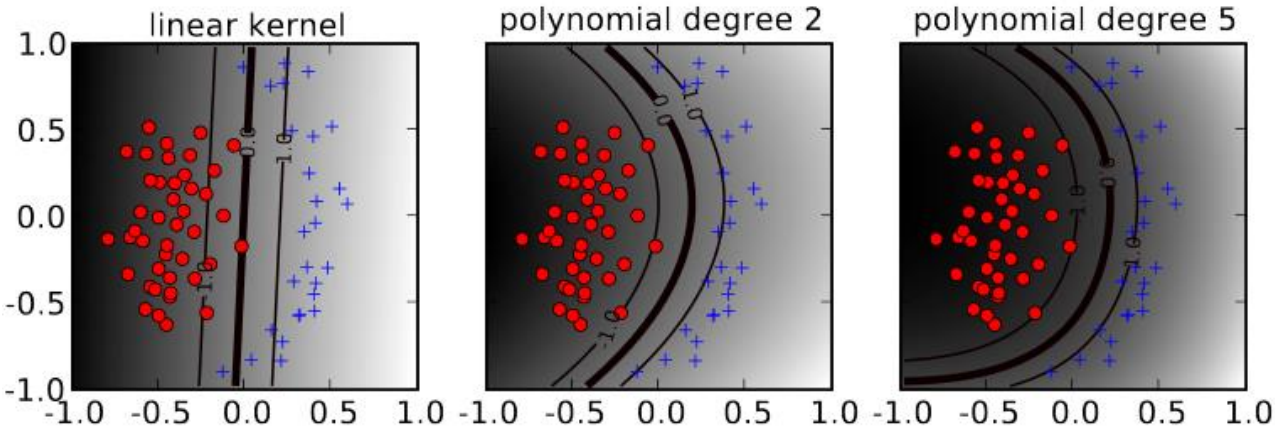
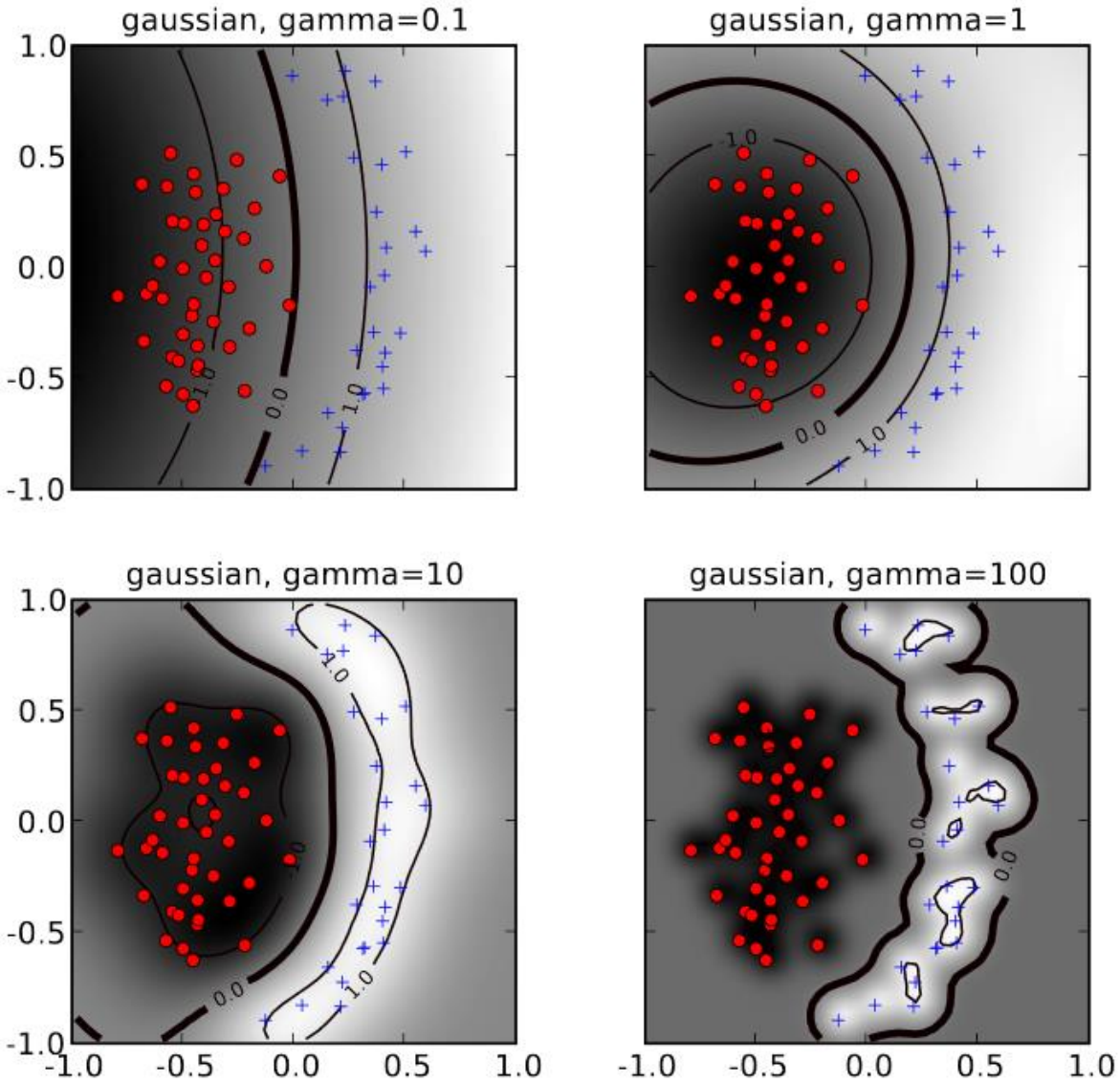



**Figure 5.4: The effect of the degree of a polynomial kernel. Higher degree polynomial kernels allow a more flexible decision boundary**

The Gaussian kernel: k(x, x′) = exp(−γ||x−x′||2) is essentially zero if the distance between x and x′ is much larger than 1/√γ; i.e. for a fixed x′ it is localized to a region around x′. The support vector expansion

$$f(\mathbf{x}) = \sum_{i=1}^{n} \alpha_i k(\mathbf{x}, \mathbf{x}_i) + b \tag{5.21}$$

is thus a sum of Gaussian "bumps" centered around each support vector. When γ is small (top left panel in following figure) a given data point x has a non-zero kernel value relative to any example in the set of support vectors. Therefore, the whole set of support vectors affects the value of the discriminant function at x, resulting in a smooth decision boundary. As γ is increased the locality of the support vector expansion increases, leading to greater curvature of the decision boundary. When γ is large the value of the discriminant function is essentially constant outside the close proximity of the region where the data are concentrated (bottom right panel in following figure). In this regime of the γ parameter the classifier is clearly overfitting the data.




The effect of the inverse-width parameter of the Gaussian kernel (γ) for a fixed value of the soft-margin constant. For small values of γ (upper left) the decision boundary is nearly linear. As γ increases the flexibility of the decision boundary increases. Large values of γ lead to overfitting (bottom).

As seen from the examples in the above figures the γ parameter of the Gaussian kernel and the degree of polynomial kernel determine the flexibility of the resulting SVM in fitting the data. If this complexity parameter is too large, overfitting will occur. A question frequently posed by practitioners is "which kernel should I use for my data?" There are several answers to this question. The first is that it is, like most practical questions in machine learning, data-dependent, so several kernels should be tried. That being said, we typically follow the following procedure: Try a linear kernel first, and then see if we can improve on its performance using a non-linear kernel. The linear kernel provides a useful baseline, and in many bio-informatics applications provides the best results: The flexibility of the Gaussian and polynomial kernels often leads to overfitting in high dimensional datasets with a small number of examples, microarray datasets being a good example. Furthermore, an SVM with a linear kernel is easier to tune since the only parameter that affects performance is the soft-margin constant. Once a result using a linear kernel is available it can serve as a baseline that you can try to improve upon using a non-linear kernel. Between the Gaussian and polynomial kernels, experience shows that the Gaussian kernel usually outperforms the polynomial kernel in both accuracy and convergence time.

The dependence of the SVM decision boundary on the SVM hyperparameters translates into a dependence of classifier accuracy on the hyperparameters. When working with a linear classifier the only hyperparameter that needs to be tuned is the SVM soft-margin constant. For the polynomial and Gaussian kernels the search space is two-dimensional. The standard method of exploring this two-dimensional space is via grid-search, the grid points are generally chosen on a logarithmic scale and classifier accuracy is estimated for each point on the grid. This is illustrated in figure below. A classifier is then trained using the hyperparameters that yield the best accuracy on the grid. The accuracy landscape in next figure has an interesting property: there is a range of parameter values that yield optimal classifier performance, furthermore, these equivalent points in parameter space fall along a "ridge" in parameter space. This phenomenon can be understood as follows. Considering a particular value of (γ, C). Decreasing the value of γ, decreases the curvature of the decision boundary, if then the value of C increases the decision boundary is forced to curve to accommodate the larger penalty for errors/margin errors. This is illustrated in following figure for two-dimensional data.

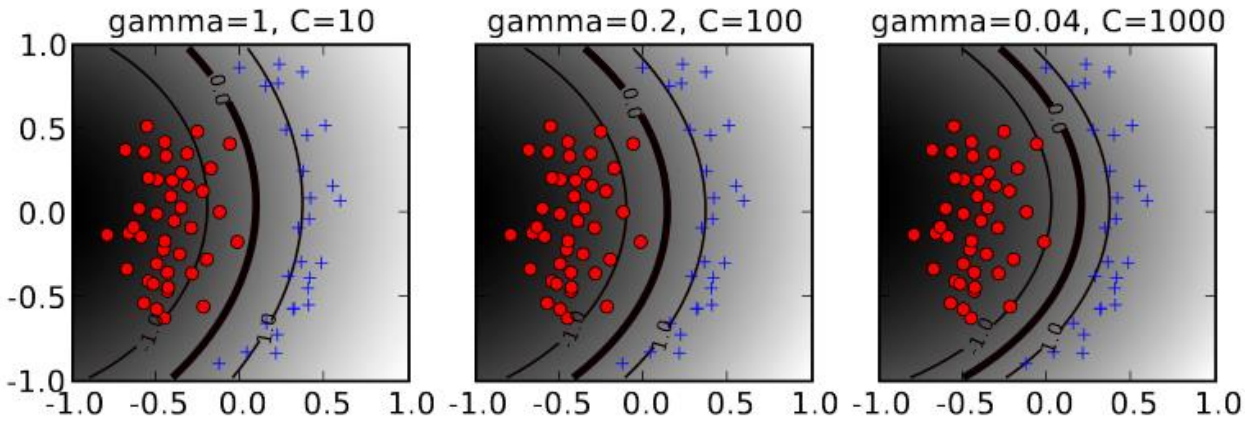



**Figure 5.5: Similar decision boundaries can be obtained using different combinations of SVM hyper parameters. The values of C and γ are indicated on each panel.**

Many datasets encountered in bioinformatics and other areas of application are unbalanced, i.e.one class contains a lot more examples than the other. Unbalanced datasets can present a challenge when training a classifier and SVMs. A good strategy for producing a high-accuracy classifier on imbalanced data is to classify any example as belonging to the majority class, this is called the majority-class classifier. While highly accurate under the standard measure of accuracy such a

classifier is not very useful. When presented with an unbalanced dataset that is not linearly separable, an SVM that follows the formulation

$$f(\mathbf{x}) = \mathbf{w}^{\mathsf{T}}\mathbf{x} + b$$

will often produce a classifier that behaves similarly to the majority-class classifier. An illustration of this phenomenon is provided in figure below. The crux of the problem is that the standard notion of accuracy (the success rate, or fraction of correctly classified examples) is not a good way to measure the success of a classifier applied to unbalanced data, as is evident by the fact that the majority-class classifier performs well under it. The problem with the success rate is that it assigns equal importance to errors made on examples belonging the majority class and errors made on examples belonging to the minority class. To correct for the imbalance in the data we need to assign different costs for misclassification to each class. Before introducing the balanced success rate we note that the success rate can be expressed as:

P(success|+)P(+) +P(success|−)P(−), where P(success|+) (P(success|−))
(5.22)

is an estimate of the probability of success in classifying positive (negative) examples, and P(+) (P(−)) is the fraction of positive (negative) examples. The balanced success rate modifies this expression to:

BSR= (P(success|+) +P(success|−))/2, (5.23)

which averages the success rates in each class. The majority-class classifier will have a balanced-success-rate of 0.5. A balanced error-rate is defined as 1−BSR. The BSR, as opposed to the standard success rate, gives equal overall weight to each class in measuring performance. A similar effect is obtained in training SVMs by assigning different misclassification costs (SVM soft-margin constants) to each class. The total misclassification cost,

$$C\sum_{i=1}^{n}\xi_i \longrightarrow C_+\sum_{i\in I_+}\xi_i + C_-\sum_{i\in I_-}\xi_i\,,$$
(5.24)

Where $C_+$ ($C_-$) is the soft-margin constant for the positive (negative) examples and $I_+$($I_-$) are the sets positive (negative) examples. To give equal overall weight to each class we want the total penalty for each class to be equal. Assuming that the number of misclassified examples from each class is proportional to the number of examples in each class, we choose $C_+$ and $C_-$such that

$C_+n_+=C_-n_-$ (5.25)

Where $n_+$($n_-$) is the number of positive (negative) examples. Or in other words:

$$\frac{C_+}{C_-} = \frac{n_-}{n_+}$$

This provides a method for setting the ratio between the soft-margin constants of the two classes, leaving one parameter that needs to be adjusted. This method for handling unbalanced data is implemented in several SVM software packages, e.g. LIBSVM and `PyML`.

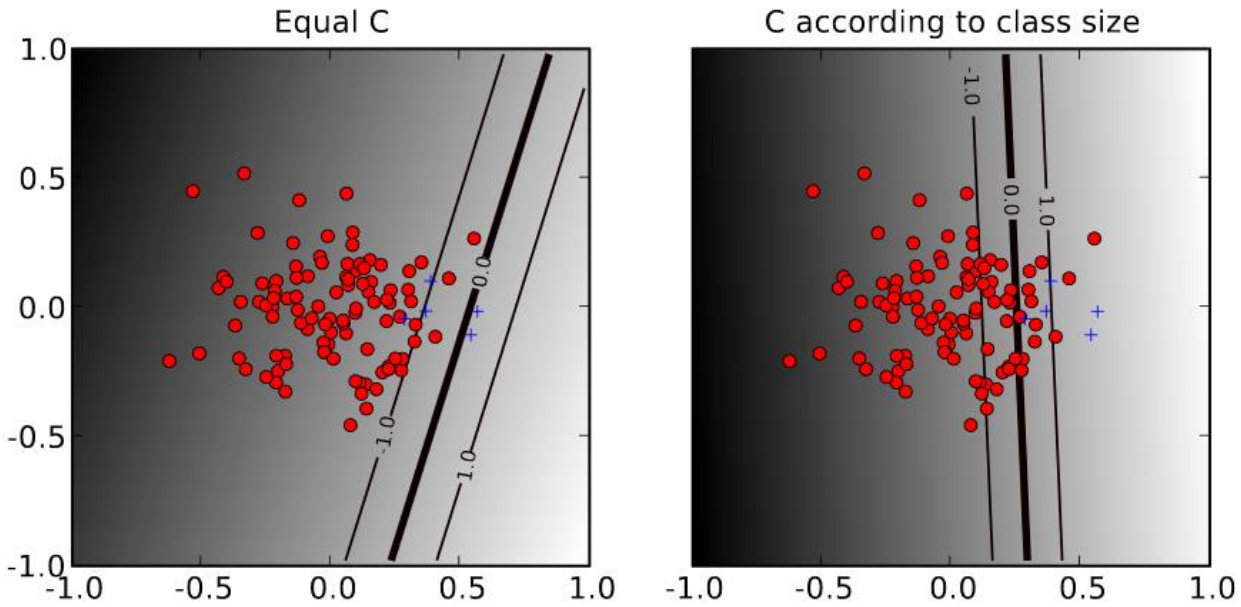



**Figure 5.6: When data is unbalanced and a single soft-margin is used, the resulting classifier (left) will tend to classify any example to the majority-class. The solution (right panel) is to assign a different soft-margin constant to each class (see text for details).**

Large margin classifiers are known to be sensitive to the way features are scaled. Therefore, it is essential to normalize either the data or the kernel itself. This observation carries over to kernel-based classifiers that use non-linear kernel functions: The accuracy of an SVM can severely degrade if the data is not normalized. Some sources of data, e.g. microarray or mass-spectrometry data require normalization methods that are technology-specific. In what follows we only consider normalization methods that are applicable regardless of the method that generated the data. Normalization can be performed at the level of the input features or at the level of the kernel (normalization in feature space). In many applications the available features are continuous values, where each feature is measured in a different scale and has a different range of possible values. In such cases it is often beneficial to scale all features to a common range, e.g. by standardizing the data (for each feature, subtracting its mean and dividing by its standard deviation). Standardization is not appropriate when the data is sparse since it destroys sparsity since each feature will typically have a different normalization constant. Another way to handle features with different ranges is to bin each feature and replace it with indicator variables that indicate which bin it falls in. An alternative to normalizing each feature separately is to normalize each example to be a unit vector. If the data is explicitly represented as vectors you can normalize the data by dividing each vector by its norm such that $||x||= 1$ after normalization. Normalization can also be performed at the level of the kernel, i.e. normalizing in feature-space, leading to$||\Phi(x)||= 1$ (or equivalently $k(x,x) = 1$). This is accomplished using the cosine kernel which normalizes a kernel $k(x,x')$ to:

$$k_{cosine}(\mathbf{x}, \mathbf{x}') = \frac{k(\mathbf{x}, \mathbf{x}')}{\sqrt{k(\mathbf{x}, \mathbf{x})k(\mathbf{x}', \mathbf{x}')}}$$

(5.26)

Note that for the linear kernel cosine normalization is equivalent to division by the norm. The use of the cosine kernel is redundant for the Gaussian kernel since it already satisfies $K(x,x) = 1$. This does not mean that normalization of the input features to unit vectors is redundant: Our experience shows that the Gaussian kernel often benefits from it. Normalizing data to unit vectors reduces the dimensionality of the data by one since the data is projected to the unit sphere. Therefore, this may not be a good idea for low dimensional data.

The popularity of SVMs has led to the development of a large number of special purpose solvers for the SVM optimization problem (Convex Problem). One of the most common SVM solvers is LIBSVM (which was used in this thesis under the sci kit learn of Python).The complexity of training of non-linear SVMs with solvers such as LIBSVM has been estimated to be quadratic in the number of training examples, which can be prohibitive for datasets with hundreds of thousands of examples. Researchers have therefore explored ways to achieve faster training times. For linear SVMs very efficient solvers are available which converge in a time which is linear in the number of examples. Approximate solvers that can be trained in linear time without a significant loss of accuracy were also developed. There are two types of software that provide SVM training algorithms. The first type are specialized software whose main objective is to provide an SVM solver. LIBSVM and SVM light are two popular examples of this class of software. The other class of software are machine learning libraries that provide a variety of classification methods and other facilities such as methods for feature selection, preprocessing etc. The user has a large number of choices, and the following is an incomplete list of environments that provide an SVM classifier: Orange, The Spider Elefant, Plearn, Weka, Lush, Shogun, Rapid Miner, and `PyML`. The SVM implementation in several of these are wrappers for the LIBSVM library.

# 6. Hardware: Analog Circuit

Having established the background knowledge needed for implementing the hardware and the software, the next chapters focus on explaining the implementation of the procedure. The circuits were designed taking into consideration the characteristics of the myoelectric signal found in bibliography. Based on these sources the voltages produced by the muscles are around 10μV and the information of the muscle activations is in range of (0, 300] Hz. The analog signal constructed for the acquisition of the myoelectric signal is separated into two circuits. The first one, whose schematic is pictured below, creates the VCC (V+) and the reference voltage Vref=1.8V required for the operational amplifiers of the second circuit. The Analog to Digital Converter (ADC) of the microprocessor works with voltages in range of [0, 3.6] V. Thus, the voltage 1.8V was chosen to be in the middle, to use the best of the dynamic range. The two operational amplifiers used in this circuit are part of an MCP602 chip.

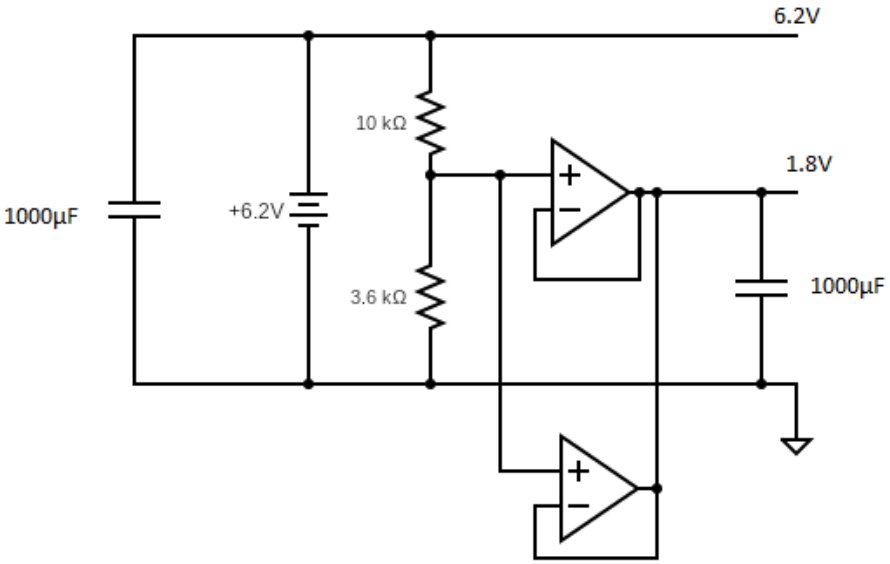



**Figure 6.1: Power supply and reference voltage circuit**

The second circuit is pictured in the following page. It's work is based on differential amplification meaning the circuit receives the signal from a pair of electrodes and subsequently it is amplified and filtered. The amplification happens in three stages and after each stage the signal is filtered. This is done because the main artifact the 50Hz of the power lines is much stronger than the signal of interest. Consequently, the hi-gain of each stage results in a so much amplified artifact that it surpasses the dynamic range. For the first stage of the amplification an INA128 chip is used. This choice is done because this chip has a common mode rejection of 100db at 50 Hz, making suitable for this case. In this stage a gain G=5 is achieved. Then before the next amplification the signal is filtered with a high-pass filter at 72Hz and then with a low-pass at 338Hz. Afterwards at the second stage of amplification the signal is amplified with a gain of 48 and then filtered. The output of the second stage goes to the third stage of amplification where it is amplified again with a gain of 48 and then filtered again. In other words, the third stage is a repetition of the 2$^{nd}$ stage. The three stages of amplification result in a net gain of 11520. In the oscilloscope it was observed that it was enough to place the signal in range of [0, 3.6] V. The circuit that was just explained is the implementation for only one channel. However, in this thesis 4 channels for recording muscle activity. Thus 4 similar circuits were constructed. These 4 circuits used a common electrode placed on the upper arm for the differential amplification connected on the V- pin of INA128 and the V+ was connected with a muscle of interest different for each channel. Finally, an electrode connected with the 1.8 Voltage was placed on a neutral area close to the belly so that a common potential exists across the skin. If not, recordings are incorrect.

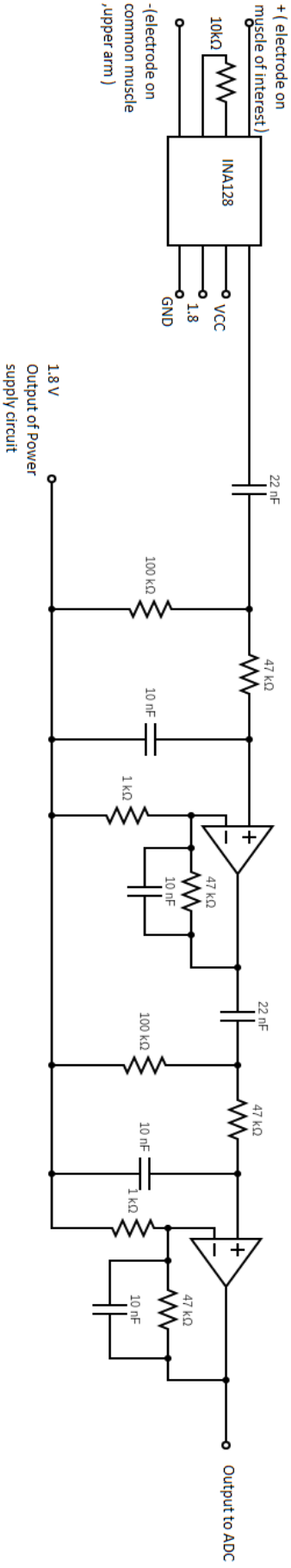



**Figure 6.2: Circuit for amplification and filtering EMG for one channel**

The operational amplifiers in the above circuit are part of AD604. Meaning that 2 such chips were used for the 4-channel implementation. The high-pass filter is implemented in the segments of the above circuit with the 100KΩ resistance having a common node with the 22nF capacitor. On the other hand, the low-pass filtered is implemented by the 47KΩ resistance and the 10nF capacitor. A real photograph of the complete implemented circuit is given below. Both the circuits designed (power supply with Vref and 4-channel amplification) were implemented on a single breadboard.

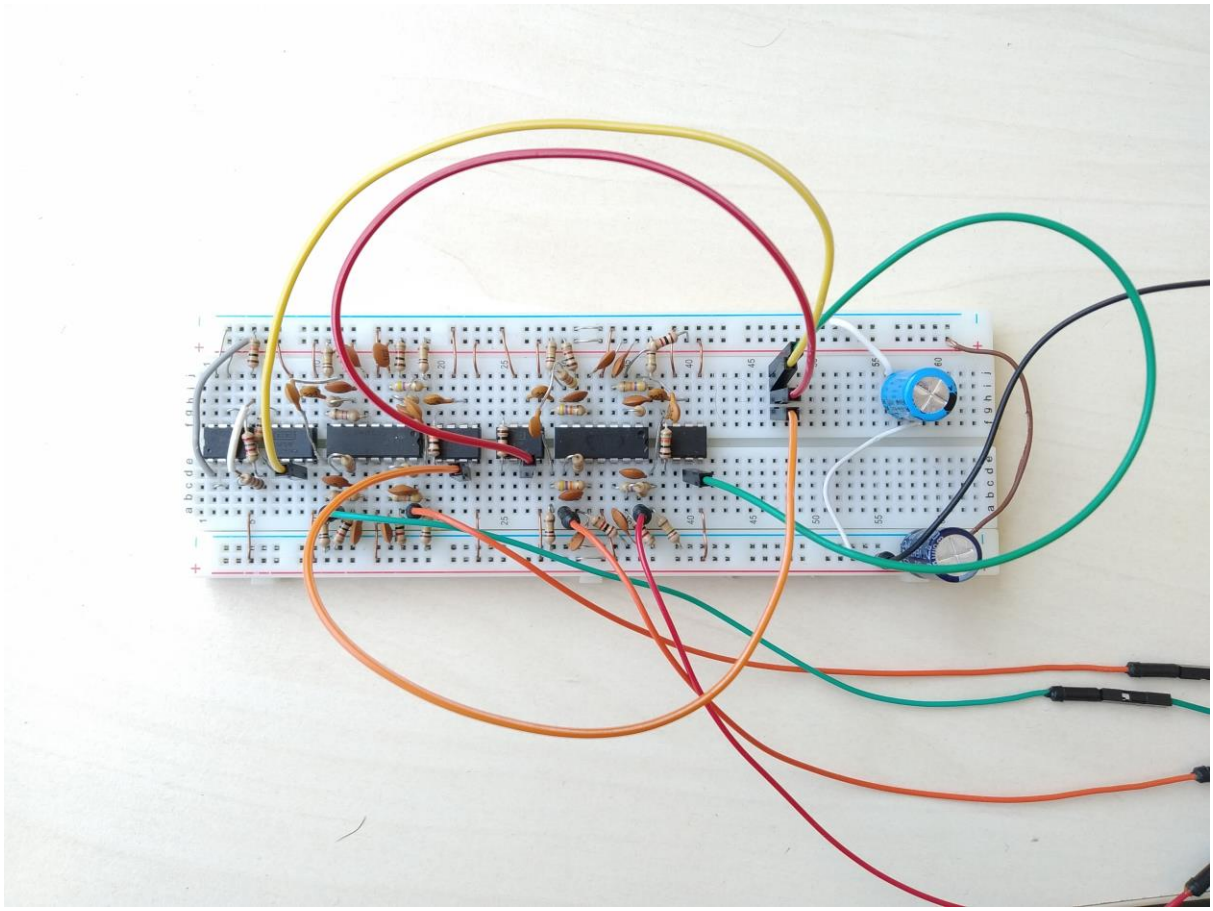


**Figure 6.3: Implementation of both circuits on one breadboard**

# 7. Software

This chapter focuses on explaining the software developed for the microprocessor, using C++ and the mbed library, and the PC using python. The microprocessor used for the signal digitalization is a Nucleo-F334R8 developing board, and was chosen due to the precision and quality of the ADC it is equipped with.

The analog circuit explained in the previous chapter produces 4 analog signals as output. This signal is sampled at 2048Hz. This means that the cables of output from the previous stage is connected with 4 pins of the Analog to Digital Converter (ADC) on the microprocessor. Afterwards the DC value is removed from each channel from the recorded signals. Subsequently these new signals are filtered with an IIR digital notch filter of 2 poles and 2 zeros removing the 50Hz contamination that the imperfect analog circuit failed to totally remove. The output of that IIR filter for each channel is stored for 128 samples so that the Root Mean Square (Rms) of each channel is computed. That means 4 buffers for storing these values are required. When the values have been accumulated the Rms is calculated and sent via Usart to a PC as a vector in the form

[$1^{st}$ channel Rms, $2^{nd}$ channel Rms, $3^{rd}$ channel Rms, $4^{th}$ channel Rms].

This form is necessary, because as mentioned in the chapter of Support Vectors Machines these vectors are used for the classification.

In the next step the data is collected. The vectors sent while the user holds a specific gesture are stored in .txt files using the application Teraterm on a PC. Each gesture is stored in a separate txt file. This was done for easier creation of training data sets comprised of specific gestures for experimentation without the need of new recordings very often. Further explaining the data collection stage, the data set is composed of 5 sessions of recordings. Each session is a collection of txt files, each one containing recordings of a specific gesture. The number of the recordings in these txt files corresponds to 10 seconds of holding a gesture.

Having created the data, the next step includes the Python code that does the training of the SVM and the classification. The code written in python is divided in 3 segments. The first segment reads the data from the files and loads them to a matrix. Next the data are normalized in range (-1, 1) using the mean value and the standard deviation. Subsequently a y matrix is created containing the labels for each vector corresponding to each gesture (numbers 0 to 8). Afterwards the matrices containing the data (data matrix) and the labels (y matrix) are segmented randomly, using a python function, into the training set and the test set. The test set does not partake at all in the training of the SVM and is used only in the end for measuring efficiency. The python library scikit-learn contains useful functions that not only implement the training of the SVM, but also can use grid search training for a number of parameters that need to be defined by the user before proceeding to the training. Further explaining, if Radial basis function (rbf) kernel is chosen for training a non-linear SVM, then the parameters C and gamma need to be defined beforehand. With the grid search tool of this library the user defines a number of values for each parameter and the function trains an SVM for each combination of them keeping their efficiency score. Furthermore, these functions contain a cross validation tool, where the training set is randomly divided into a smaller training set and test set. Next an SVM is trained with that smaller set and tested with the smaller training set that originated from the segmentation of the first bigger training set. This process (the random segmentation and training of the first training set) is repeated as many times as the user defines, by giving value to a parameter called cv. The mean value of the scores of each SVM is presented for each combination of values C and gamma. Consequently, 5-fold cross-validation, 10-fold and so on is possible. In the end, after cross-validation is performed for every parameter combination, the original training set that has not been used for training at all is tested with the SVM whose

parameters scored the best in the previous cross-validation and parameter-combination trials. It is notable that there are many metrics by which the performance of the classification of the SVM can be measured. Such metrics are:

Recall: the fraction of relevant instances that have been retrieved over the total amount of relevant instances.

Precision: the fraction of relevant instances among the retrieved instances.

$F_1$ score (also F-score or F-measure): a measure of a test's accuracy. It considers both the precision *p* and the recall *r* of the test to compute the score. The $F_1$ score is the harmonic average of the precision and the recall, where an $F_1$ score reaches its best value at 1 (perfect precision and recall) and worst at 0.

$$F_1 = \left( \frac{\mathrm{recall}^{-1} + \mathrm{precision}^{-1}}{2} \right)^{-1} = 2 \cdot \frac{\mathrm{precision} \cdot \mathrm{recall}}{\mathrm{precision} + \mathrm{recall}}.$$

(7.1)

In my case the metric used for defining the best is the recall. Finally, the performance of this SVM with the unfamiliar training set is presented. The second segment of python code does the same work as the one just described but the range of the parameter values for the grid search are close to the best value found in the previous segment. To illustrate the situation better an example will be given. The first segment will train an SVM for each of the following combination of values for parameters C and gamma:

gamma: $[2^3,2^2,2^1,2^0,2^{-1}, 2^{-2},2^{-3},2^{-4},2^{-5}]$
C: $[2^{-10},2^{-9},2^{-8},2^{-7},2^{-6}, 2^{-5},2^{-4},2^{-3}, 2^{-2}, 2^{-1}, 2^0,2^1,2^2,2^3, 2^4,2^5,2^6,2^7,2^8,2^9,2^{10}]$

Assuming that the best performing SVM is found for parameter values $C= 2^2 =4$ and $gamma=2^{-1}=0.5$. Then in the beginning of the second segment the user needs to define a grid around these areas with much smaller difference between two values. Consequently, and hopefully the optimal SVM for classifying the gestures is trained. Notably this the procedure suggested by the authors of the library LibSVM which is the one executed below the python scikit-learn library. The next and final segment of python code is for the real time classification. An SVM is trained with the best parameters found in the previous stage. A serial connection is established using a usb to serial cable, which connects the PC and the microprocessor. Additionally, a wireless connection via Bluetooth is established among the PC and the NXT. Thus, there is an infinite loop running in which vectors are read from the serial port and are classified with the SVM (trying to predict which gesture the user holds) and based on that commands are sent to the NXT. The commands are simple byte streams that command a motor to rotate clockwise or anticlockwise or stop. A more accurate and detailed explanation is given in the following paragraphs dedicated to the NXT. An example though to illustrate the situation is the following the command that starts the motor connected to port A at half power.

serial.print (0x0C 0x00 0x00 0x04 0x00 0x64 0x07 0x00 0x00 0x20 0x00 0x00 0x00 0x00 0x00)

As an example, the SVM is trained to recognize and classify vectors among 5 gestures (labels generally). Thus, when gesture0 is recognized byte streams are sent that stop the motors. This is done so that when the user wants to have a relaxed hand and not to give commands. When the gesture3 is recognized a byte stream to start motor in port A clockwise and a byte stream to stop motor in port B are sent. When gesture4 is recognized a byte stream to start motor in port A anti-

clockwise and a byte stream to stop motor in port B are sent. When gesture6 is recognized a byte stream to start motor in port B clockwise and a byte stream to stop motor in port A are sent. When gesture8 is recognized a byte stream to start motor in port B anti-clockwise and a byte stream to stop motor in port A are sent. The gesture numbers are not in consequence because they are part of a bigger collection of gestures, but only among them the classification was most successful.

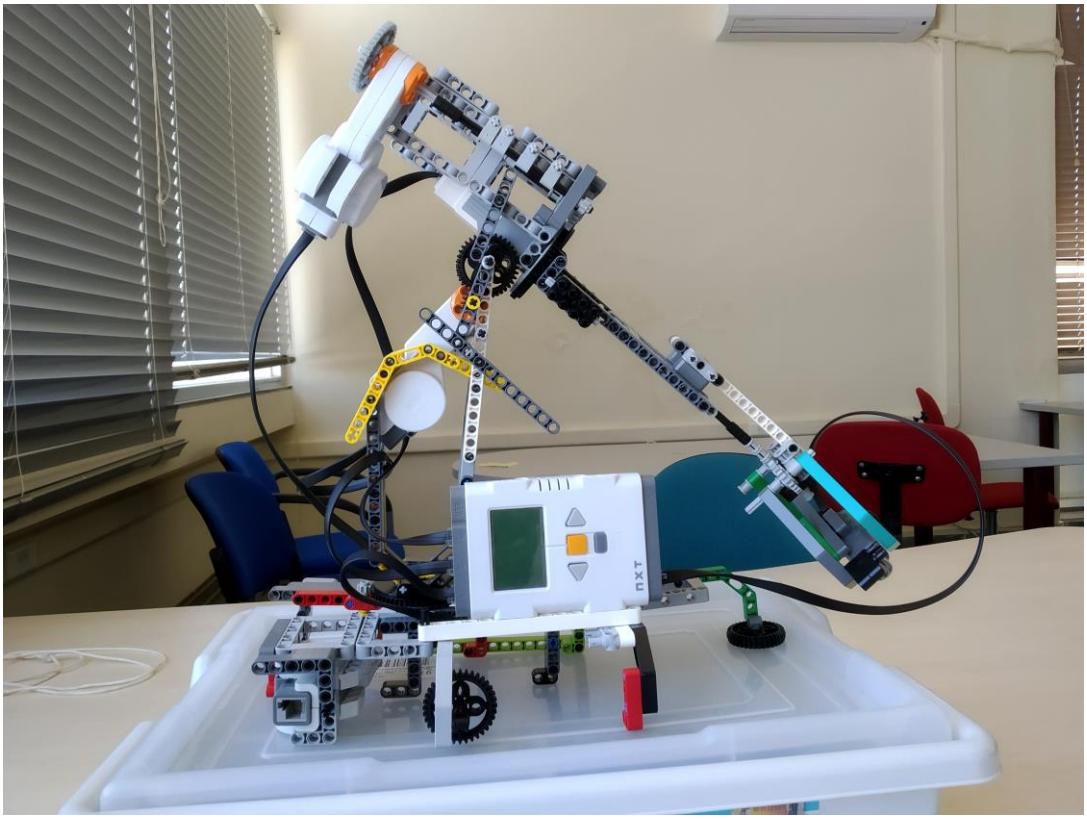


**Figure 7.1: Photograph of the robotic arm**

The robotic arm mentioned in the previous chapters is illustrated above. Lego blocks were used to build a robot arm with 4 degrees of freedom. Three of them are rotational degrees so that the end-effector was positioned in the robot ‘s workspace. The fourth one is also rotational but is used to close and open the gripper.

The figure below shows the schematic of the robot and its degrees of freedom. The arrows denote the axis z of rotation.

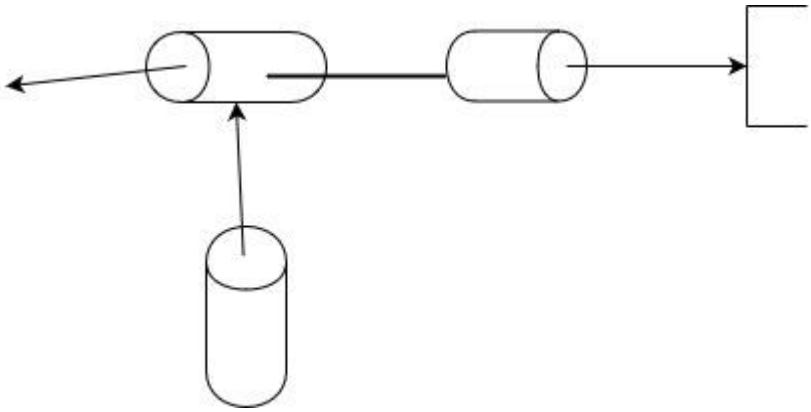


**Figure 7.2: Schematic of the robot ‘s degrees of freedom**


**Figure 7.3: Lego 's microprocessor for controlling the motors**

The motors of the robot are controlled through the microprocessor manufactured by Lego company, the NXT (photograph above). The following figure provides a graphical overview of how different functions are connected and controlled within the NXT. The figure includes only- high level units. The NXT has three output ports for controlling the actuators connected to it and 4 input ports for measuring different parametres in the psysical world using different sensors. The NXT was programmed using the NXC language and the Bricx Command Center 3.3 . Bricx Command Center (BricxCC) is a 32-bit Windows program commonly known as an integrated development environment (IDE) for programming LEGO MINDSTORMS robots. NXC stands for Not eXactly C. It is a simple language for programming the LEGO MINDSTORMS NXT product. The NXT has a bytecode interpreter (provided by LEGO), which can be used to execute programs. The NXC compiler translates a source program into NXT bytecodes, which can then be executed on the target itself. Although the preprocessor and control structures of NXC are very similar to C, NXC is not a general-purpose programming language - there are many restrictions that stem from limitations of the NXT bytecode interpreter.

What is notable is that NXT can communicate via Bluetooth with other devices equipped with Bluetooth like mobile phones, computers, and embedded processors (like my case). Through Bluetooth the NXT can receive direct commands. This functionality of the NXT is a sub protocol of the main Lego Mindstorms communication protocol designed for direct commands which make it possible to control the NXT brick from outside devices with a Serial Port Profile (SPP). This sub-protocol provides simple interface for these outside devices to utilize basic brick functionality (sensor readings, motor control, power management) without the need to write or run specialized remote controled programs on the brick. The protocol also includes an interface to send arbitrary messages to a target brick, which may be picked up and used in a user defined NXT program. Sending direct commands to the NXT basically is sending a series of bytes from the commanding device that will be interpreted appropriately. The figure below shows the general protocol architecture for the communication via Bluetooth.

| Length, LSB | Length, MSB | Command Type | Command | Byte 5 | Byte 6 | Etc. |
|---|---|---|---|---|---|---|

**Figure 7.4: Byte stream format needed for communication between the NXT and other Bluetooth devices.**

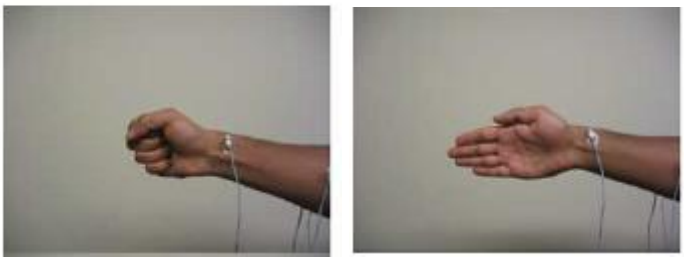
All Bluetooth messages need to have two bytes in front of the message itself which indicates how many bytes the message includes. The length of the package is counted without the two length

bytes. An example of how it works will be given in the section where the procedure followed in the diploma thesis is explained step by step. Another memorable characteristic is that the NXT must be connected as slave to the other Bluetooth device in in order to receive direct commands.

# 8.Results of the off-line Evaluation

A number of experiments were done, measuring the recall, precision and fi-score (their definition given on the previous chapter) of SVMs trained with data-sets composed of different number gestures and slightly different electrode placement on the muscles mentioned in the introduction. Also, it was investigated how different kind of electrodes affect scores. Adult electrodes and infant electrodes were used. In some experiments a combination of these two was used. The recall score is the one with the greatest importance since the SVMs were trained with parameters to maximize this value. The three most indicative will be presented in this chapter because they illustrate the recognizing capabilities of a minimal system comprised of 4 input electrodes. The rest of the experiments are presented in the Appendix A.

In the first experiment classification among 3 gestures was done:

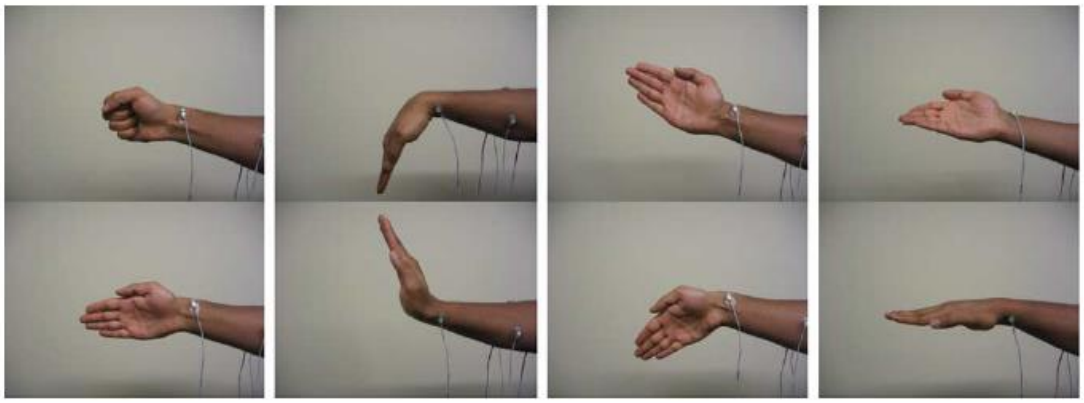


The third gesture, 0 named, is the complete relaxed hand gesture. Gesture 1 is the left and gesture 2 is the right one.
The results are:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 1.00 | 1.00 |
| 1 | 0.80 | 0.55 | 0.66 |
| 2 | 0.60 | 0.82 | 0.70 |

From this experiment and from similar others trying to classify among these gestures the distinguishability of gesture 0 is obvious. However, it is concluded that with 4 electrodes and for that selection of muscles the efficiency of the classification is rather low and cannot proceed to testing in real-time control.

In the next experiment classification among 9 gestures is done:



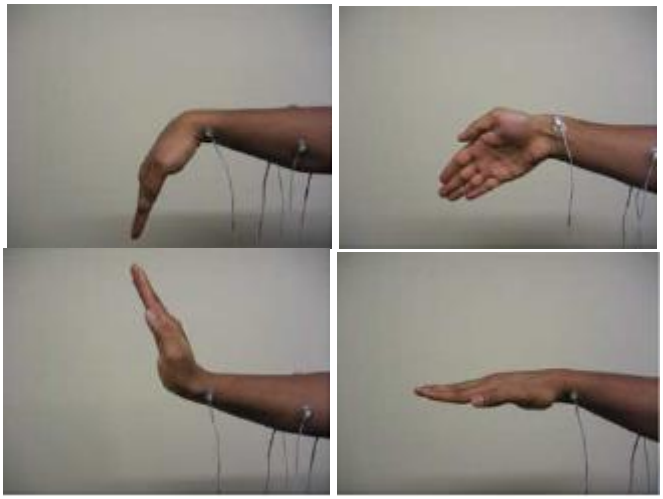
The ninth gesture named 0 is the complete relaxed hand gesture.

The results are:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 1 | 0.64 | 0.37 | 0.47 |
| 2 | 0.41 | 0.42 | 0.42 |
| 3 | 0.89 | 0.91 | 0.90 |
| 4 | 0.89 | 0.84 | 0.86 |
| 5 | 0.65 | 0.78 | 0.71 |
| 6 | 0.64 | 0.73 | 0.68 |
| 7 | 0.75 | 0.81 | 0.78 |
| 8 | 0.90 | 0.91 | 0.90 |
| avg | 0.75 | 0.75 | 0.75 |

With this experiment it becomes apparent that, the control of all four motors of the robot cannot be achieved with only 4 electrodes.

In the third experiment the data-set was composed of gestures that were better recognized in other experiments. So, the gestures that were classified are:



Plus, the gesture 0, the relaxed hand gesture.
The results of this experiment are:

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 |
| 3 | 0.99 | 0.97 | 0.98 |
| 4 | 1.00 | 0.99 | 1.00 |
| 6 | 0.96 | 0.98 | 0.97 |
| 8 | 0.97 | 0.96 | 0.96 |
| avg | 0.98 | 0.98 | 0.98 |

This time the results are comparable to the results produced by the research, this thesis was based on. Such results indicate that real-control of robot can be achieved.

Finally taking into consideration all the experiments, it is concluded that the use of 3 infant electrodes and 1 adult on pronator quadratus for detecting the signal yields better results than using only adult or only infant electrodes. Generally, the use of infant electrodes is suggested because they cover a smaller area and thus better spatial information of the signal is obtained.

# 9.Conclusion

Although, the experiments high rates of correct classification for a number of gestures the real time control is still a challenge. The greatest issue is the transition signal. The transition signal appears when the user changes from gesture to gesture. In this time, no matter how small it is, the vectors produced for classification appear to be random. This results to movements of the degrees of freedom that cannot be anticipated. When the user has reached the desired position of the end effector then he needs to change gesture to make the motors stop or start moving in a different direction. However, the transition signal is unavoidable, so the end effector slips from the desired position. This leads to a vicious circle. A solution is to stop giving commands while the transition signal is detected. For automatic detection further research is needed. A work-around that beats the purpose of real time control, is to use a user button stop sending commands, while the user changed between gestures.

Another major issue is the muscle fatigue. The experiments showed that after some minutes of use after the training the signals produced from the muscles has changed enough so that the classifier fails to recognize some of the gestures. After a short break, relaxing the arm, gestures that had stopped being recognized, started to be again. This is blissful on one hand, since no training is required again, on the other hand though it is frustrating since the user loses the control unconsciously. Such unexpected lack of control is detrimental and unacceptable in real time tasks. This problem is insoluble with systems statically trained, meaning they are not trained while the user operates them.

Lastly, the need of frequent training and definitely when new electrodes are placed on the skin becomes obvious. The process of recording the gestures requires the minimum 6,3 minutes for 7 gestures. This may not be conceived as a lot, but it is tedious process and becomes frustrating when it needs to be done almost every hour and half.

To conclude with, this technology and methods presented in this thesis are not ripe enough for the designing of product that is convenient to use and has the reliability necessary for its operation. However, these techniques can be used as building block or a supplement to other methodologies that will result in a complete product that will place a new standard for the robotic prosthetic devices for our fellow citizens in need.

# References


- https://www.khanacademy.org/test-prep/mcat/organ-systems/neuron-membrane-potentials/a/neuron-action-potentials-the-creation-of-a-brain-signal
- https://www.sciencedirect.com/topics/medicine-and-dentistry/excitation-contraction-coupling
- https://biologydictionary.net/muscle-cell/
- Lodish, H., Berk, A., Kaiser, C. A., Krieger, M., Scott, M. P., Bretscher, A., . . . Matsudaira, P. (2008). *Molecular Cell Biology 6th. ed.* New York: W.H. Freeman and Company.
- Reece, J. B., Urry, L. A., Cain, M. L., Wasserman, S. A., Minorsky, P. V., & Jackson, R. B. (2014). *Campbell Biology, Tenth Edition* (Vol. 1). Boston: Pearson Learning Solutions.
- Blausen.com staff (2014). "Medical gallery of Blausen Medical 2014". WikiJournal of Medicine 1 (2). DOI:10.15347/wjm/2014.010. ISSN 2002-4436.
- Methods of Acquisition and Signal Processing for Myoelectric Control of Artificial Arms, Eduard Franti, Lucian Milea
- http://www.philohome.com/nxtmotor/nxtmotor.htm
- NXC tutorial
- NXC User Guide
- Lego Mindstorms NXT communiction Protocol
- Lego Mindstorms NXT Direct Commands
- Online Electromyographic Control of a Robotic Prosthesis, Pradeep Shenoy_, Kai J. Miller, Beau Crawford, and Rajesh P. N. Rao
- "Basic Techniques for Digital Signal Processing", by Mr. Moustakidis, professor of the Electrical Engineering Department of University of Patras.
- Kernel Parameter Selection for Support Vector Machine Classification, Zhiliang Liu* and Hongbing Xu
- Methods of Acquisition and Signal Processing for Myoelectric Control of Artificial Arms, Eduard FRANTI , Lucian MILEA, Verona BUTU
- Support Vector Networks, Corrina Cortes & Vladimir Vapnik
- Frequency Parameters of the Myoelectric Signal as a Measure of Muscle Conduction Velocity, FOSTERB.STULEN & CARLOJ.DELUCA
- A Practical Guide to Support Vector Classification, Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin

# Appendix A

In the following experiments the parameters C and gamma that optimized recall score were selected.

Grid search parameters gamma and C using radial basis function and linear kernel
Searching grid:

gamma: [$2^3$,$2^2$,$2^1$,$2^0$,$2^{-1}$, $2^{-2}$,$2^{-3}$,$2^{-4}$,$2^{-5}$]
C: [$2^{-10}$,$2^{-9}$,$2^{-8}$,$2^{-7}$,$2^{-6}$, $2^{-5}$,$2^{-4}$,$2^{-3}$, $2^{-2}$, $2^{-1}$, $2^0$,$2^1$,$2^2$,$2^3$, $2^4$,$2^5$,$2^6$,$2^7$,$2^8$,$2^9$,$2^{10}$]

The above-mentioned grid is used by the first python segment code described in the previous chapter. After that, the second segment is executed searching for better parameters close to the ones found previously. Notably, the gamma parameter is not used for linear kernel SVMs.

## Experiment 1 with Recordings from Baby electrodes

Recording 4 channels with baby sized electrodes and 3 electrodes being connected with plain cable.
Classification between 2 gestures:


Best parameters set found on development set:
C: 4, gamma: 0.5, kernel: rbf

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 0.89 | 0.84 | 0.86 |
| 2 | 0.85 | 0.90 | 0.88 |
| avg | 0.87 | 0.87 | 0.87 |

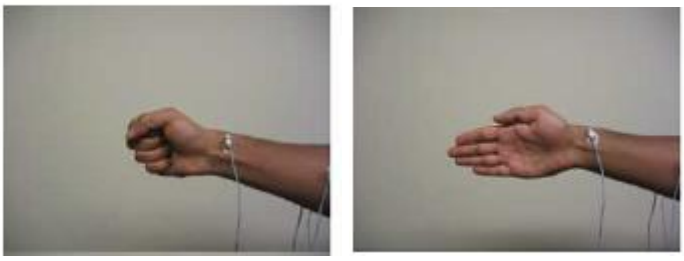
More detailed Grid Search in the Vicinity of previously found parameters
gamma: [1,8] with step 1
C: [1/8, 1] step 0.01

Best parameters set found on development set:
C: 2, gamma: 0.7050000000000005, kernel: rbf
Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 0.88 | 0.85 | 0.86 |
| 2 | 0.86 | 0.89 | 0.87 |
| avg | 0.87 | 0.87 | 0.87 |

## Experiment 2 with Recordings from adult electrodes.

Recording 4 channels with adult sized electrodes and 3 electrodes being connected with plain cable.
Classification between 2 gestures:



Best parameters set found on development set:
C: 1024, gamma: 1, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 0.82 | 0.62 | 0.70 |
| 2 | 0.68 | 0.86 | 0.76 |
| avg | 0.73 | 0.73 | 0.73 |

More detailed Grid Search in the Vicinity of previously found parameters

C =[1000,1050] step 3
gamma =[0.55,1.5] step 0.01

Best parameters set found on development set:
C: 1015, gamma: 1.3499999999999999, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 0.81 | 0.66 | 0.73 |
| 2 | 0.70 | 0.84 | 0.77 |
| avg | 0.75 | 0.75 | 0.75 |

# Experiment 3 with Recordings from baby electrodes, a different day with a slightly different placement of electrodes

Recording 4 channels with baby sized electrodes and 4 electrodes being connected with plain cable
Classification between 2 gestures:


Best parameters set found on development set:
C: 128, gamma: 0.125, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 0.69 | 0.55 | 0.61 |
| 2 | 0.67 | 0.79 | 0.72 |
| avg | 0.68 | 0.68 | 0.68 |

More detailed Grid Search in the Vicinity of previously found parameters

C: [100,150] step 1
gama : [0.001,1] step 0.01

Best parameters set found on development set:
C: 116, gamma: 0.13099999999999998, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 1 | 0.68 | 0.54 | 0.60 |
| 2 | 0.66 | 0.78 | 0.72 |
| avg | 0.67 | 0.67 | 0.67 |

# Experiment 4 with Recordings from baby electrodes

Classification among 3 gestures:


The third gesture named gesture0 is the complete relaxed hand gesture

Best parameters set found on development set:
C: 128, gamma: 0.25, kernel: rbf
Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 1.00 | 1.00 |
| 1 | 0.80 | 0.55 | 0.66 |
| 2 | 0.60 | 0.82 | 0.70 |
| avg | 0.77 | 0.77 | 0.77 |

More detailed Grid Search in the Vicinity of previously found parameters
Best parameters set found on development set:

C: 148, gamma: 0.24000000000000002, kernel: rbf

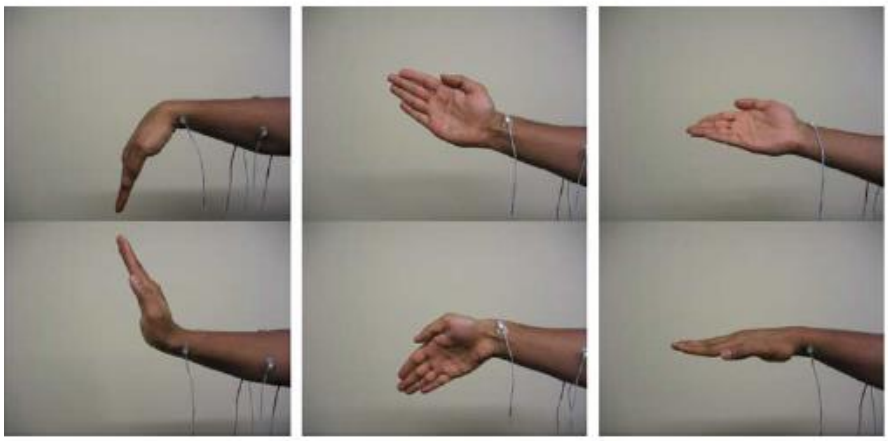
Detailed classification report:
The model is trained on the full development set.

The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 1.00 | 1.00 |
| 1 | 0.80 | 0.56 | 0.66 |
| 2 | 0.60 | 0.82 | 0.70 |
| avg | 0.77 | 0.77 | 0.77 |

# Experiment 5 with Recordings from baby electrodes

Classification among 7 gestures:



The seventh  gesture named gesture0 is the complete relaxed hand gesture

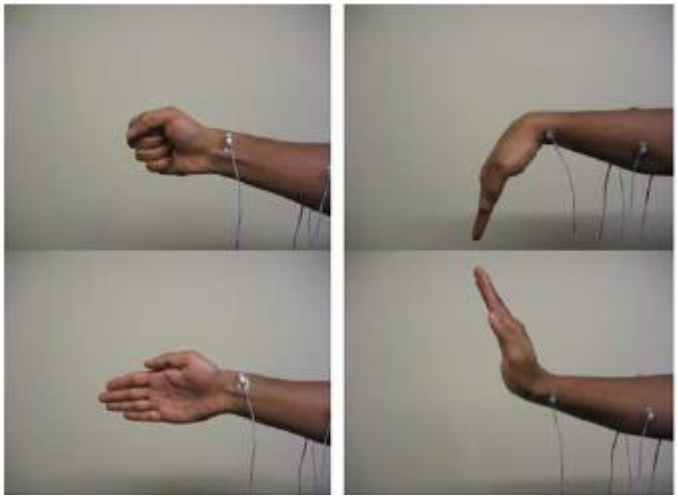
Best parameters set found on development set:
C: 16, gamma: 0.25, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 |
| 3 | 0.89 | 0.87 | 0.88 |
| 4 | 0.95 | 0.83 | 0.88 |
| 5 | 0.79 | 0.87 | 0.83 |
| 6 | 0.88 | 0.82 | 0.85 |
| 7 | 0.86 | 0.89 | 0.88 |
| 8 | 0.88 | 0.92 | 0.90 |
| avg | 0.88 | 0.88 | 0.88 |

# Experiment 6 with Recordings from baby electrodes

Classification among 5 gestures:



The fifth gesture named gesture0 is the complete relaxed hand gesture

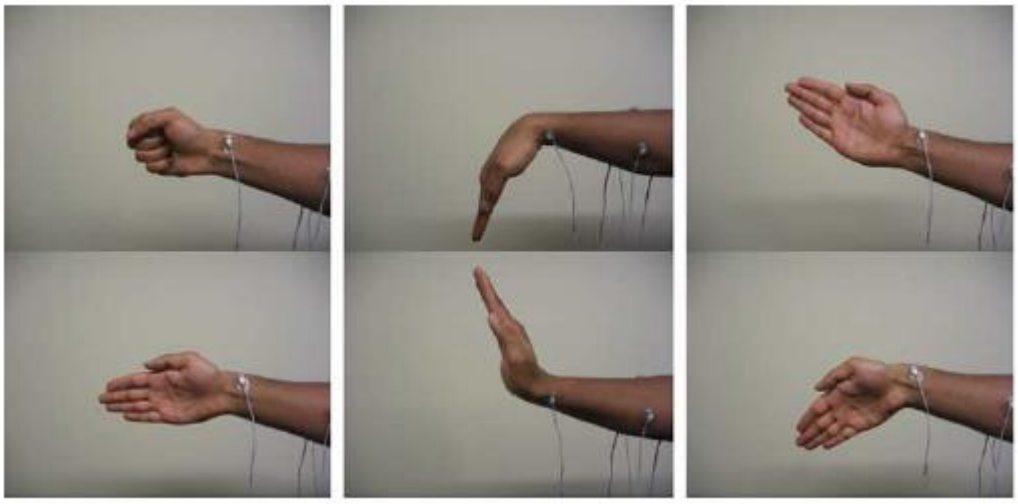
Best parameters set found on development set:
C: 2, gamma: 4, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.99 | 1.00 |
| 1 | 0.67 | 0.50 | 0.58 |
| 2 | 0.66 | 0.77 | 0.71 |
| 3 | 0.98 | 0.98 | 0.98 |
| 4 | 0.93 | 0.97 | 0.95 |
| avg | 0.85 | 0.85 | 0.85 |

# Experiment 7 with Recordings from baby electrodes

Classification among 7 gestures:



The seventh  gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 8, gamma: 1, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.99 | 1.00 |
| 1 | 0.72 | 0.40 | 0.51 |
| 2 | 0.49 | 0.56 | 0.52 |
| 3 | 0.88 | 0.93 | 0.91 |
| 4 | 0.88 | 0.85 | 0.87 |
| 5 | 0.67 | 0.74 | 0.71 |
| 6 | 0.62 | 0.75 | 0.68 |
| avg | 0.74 | 0.74 | 0.74 |

More detailed Grid Search in the Vicinity of previously found parameters

C: [2,10] step 0.5
gamma: [0.1,1.5] step 0.1

Best parameters set found on development set:

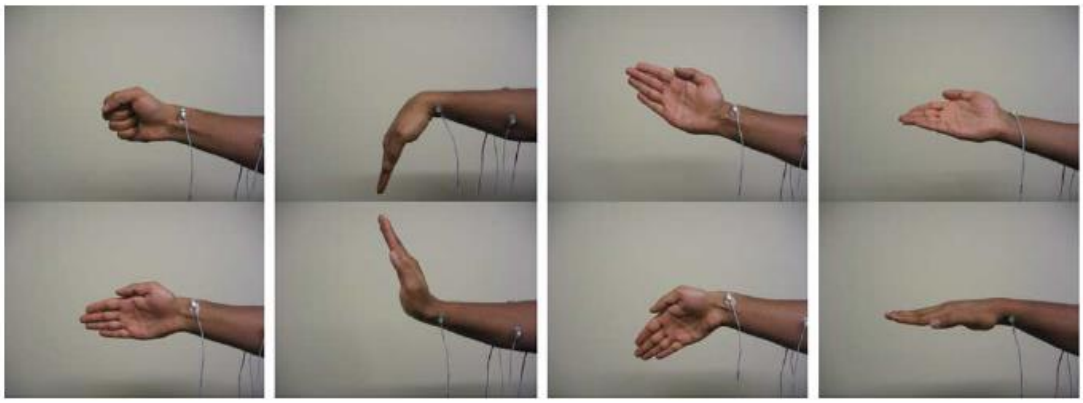
C: 5.5, gamma: 1.0, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.99 | 1.00 |
| 1 | 0.73 | 0.40 | 0.51 |
| 2 | 0.49 | 0.57 | 0.53 |
| 3 | 0.89 | 0.96 | 0.92 |
| 4 | 0.90 | 0.86 | 0.88 |
| 5 | 0.68 | 0.74 | 0.71 |
| 6 | 0.64 | 0.76 | 0.69 |
| avg | 0.75 | 0.75 | 0.75 |

# Experiment 8 with Recordings from baby electrodes

Classification among 9 gestures:



The ninth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 2, gamma: 1, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 1 | 0.64 | 0.37 | 0.47 |
| 2 | 0.41 | 0.42 | 0.42 |
| 3 | 0.89 | 0.91 | 0.90 |
| 4 | 0.89 | 0.84 | 0.86 |
| 5 | 0.65 | 0.78 | 0.71 |
| 6 | 0.64 | 0.73 | 0.68 |
| 7 | 0.75 | 0.81 | 0.78 |
| 8 | 0.90 | 0.91 | 0.90 |
| avg | 0.75 | 0.75 | 0.75 |

More detailed Grid Search in the Vicinity of previously found parameters

C: [0.5,8] step 0.3
Gama: [0.1,1.5] step 0.1

Best parameters set found on development set:

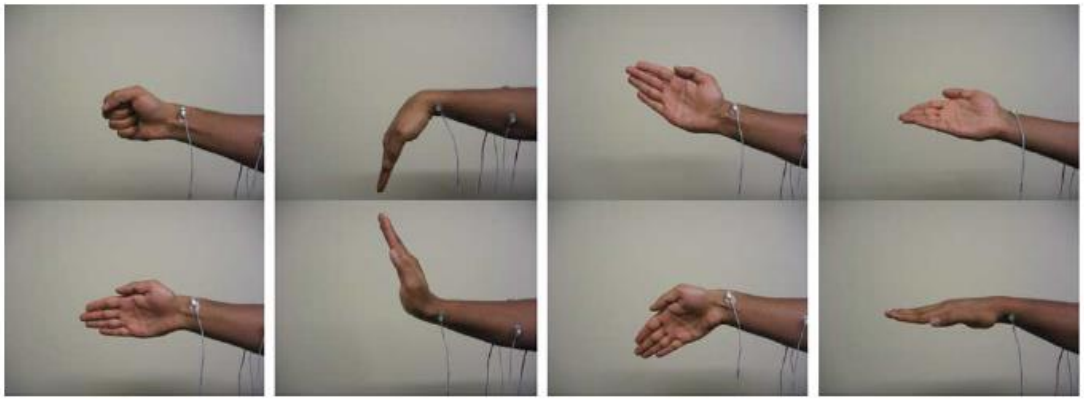
C: 1.7000000000000002, gamma: 1.2000000000000002, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 1 | 0.68 | 0.40 | 0.50 |
| 2 | 0.41 | 0.42 | 0.41 |
| 3 | 0.88 | 0.89 | 0.88 |
| 4 | 0.90 | 0.81 | 0.85 |
| 5 | 0.64 | 0.82 | 0.72 |
| 6 | 0.65 | 0.75 | 0.69 |
| 7 | 0.77 | 0.78 | 0.77 |
| 8 | 0.88 | 0.94 | 0.91 |
| avg | 0.75 | 0.75 | 0.75 |

# Experiment 9 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

Classification among 9 gestures:



The ninth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 8, gamma: 1, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 0.99 | 0.99 |
| 1 | 0.65 | 0.48 | 0.55 |
| 2 | 0.70 | 0.68 | 0.69 |
| 3 | 0.85 | 0.79 | 0.82 |
| 4 | 0.94 | 0.88 | 0.91 |
| 5 | 0.74 | 0.76 | 0.75 |
| 6 | 0.78 | 0.81 | 0.79 |
| 7 | 0.58 | 0.70 | 0.64 |
| 8 | 0.74 | 0.85 | 0.79 |
| avg | 0.77 | 0.77 | 0.76 |

More detailed Grid Search in the Vicinity of previously found parameters

C: [4,12,0.3]
gama: [0.1,2,0.1]

Best parameters set found on development set:

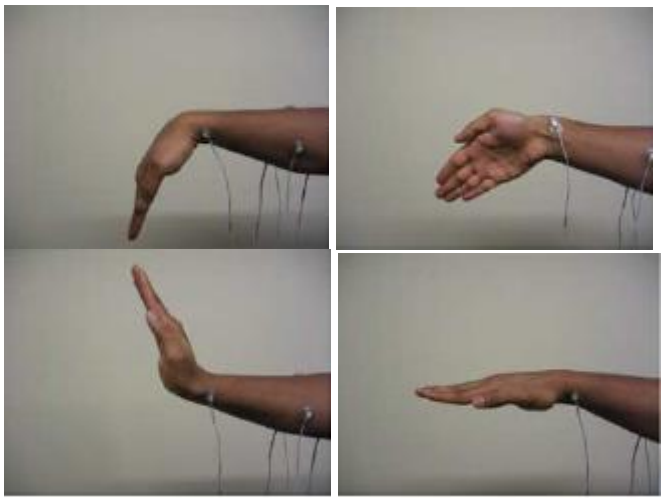
C: 4.0, gamma: 1.4000000000000001, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.99 | 1.00 |
| 1 | 0.64 | 0.48 | 0.55 |
| 2 | 0.69 | 0.68 | 0.69 |
| 3 | 0.87 | 0.80 | 0.83 |
| 4 | 0.95 | 0.90 | 0.93 |
| 5 | 0.71 | 0.76 | 0.74 |
| 6 | 0.78 | 0.86 | 0.82 |
| 7 | 0.57 | 0.65 | 0.61 |
| 8 | 0.76 | 0.83 | 0.79 |
| avg | 0.77 | 0.77 | 0.77 |

# Experiment 10 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

Classification among 5 gestures:



The fifth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 8, gamma: 2, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 |
| 3 | 0.92 | 0.96 | 0.94 |
| 4 | 0.99 | 0.96 | 0.97 |
| 6 | 0.90 | 0.94 | 0.92 |
| 8 | 0.95 | 0.92 | 0.94 |
| avg | 0.95 | 0.95 | 0.95 |

More detailed Grid Search in the Vicinity of previously found parameters

c: [4,16] step 0.3
gama: [1,3] step 0.1

Best parameters set found on development set:

C: 13.899999999999995, gamma: 2.000000000000001, kernel': rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 |
| 3 | 0.92 | 0.96 | 0.94 |
| 4 | 0.99 | 0.94 | 0.96 |
| 6 | 0.94 | 0.96 | 0.95 |
| 8 | 0.95 | 0.95 | 0.95 |
| avg | 0.96 | 0.96 | 0.96 |

In linear kernel for the same gestures the results:

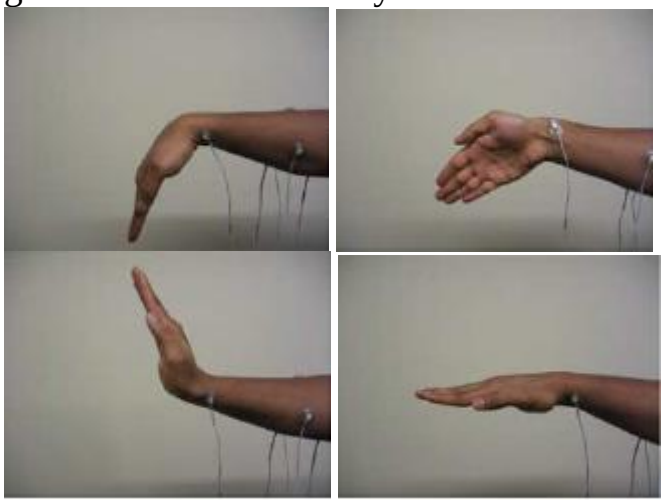
Best parameters set found on development set:

C: 13.899999999999995, kernel': linear
Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score | support |
|---|---|---|---|---|
| 0 | 0.99 | 0.98 | 0.99 | |
| 3 | 0.87 | 0.76 | 0.81 | |
| 4 | 0.94 | 0.94 | 0.94 | |
| 6 | 0.88 | 0.93 | 0.90 | |
| 8 | 0.87 | 0.91 | 0.89 | |
| avg | 0.91 | 0.91 | 0.91 | |

# Experiment 11 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

Experiment 11 is attempt to recreate the success of experiment 10 with recording the gestures in a different day.



The fifth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 32, gamma: 2, kernel: rbf

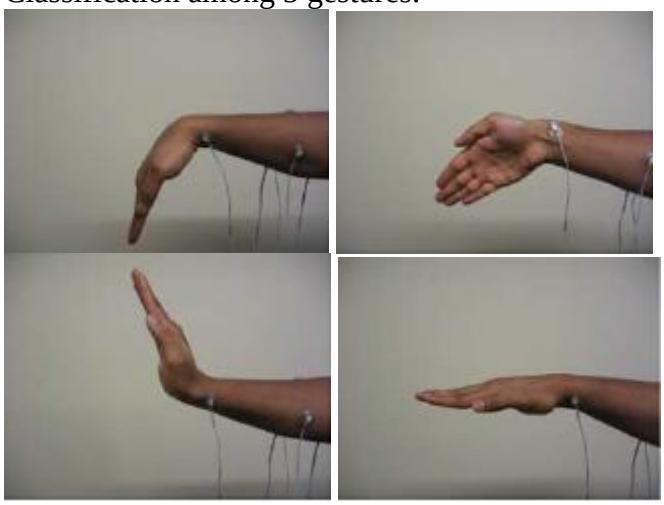
Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 |
| 3 | 0.88 | 0.90 | 0.89 |
| 4 | 0.99 | 1.00 | 0.99 |
| 6 | 0.71 | 0.71 | 0.71 |
| 8 | 0.73 | 0.71 | 0.72 |
| avg | 0.85 | 0.85 | 0.85 |

# Experiment 12 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This time connection of the three electrodes that was plain cable now it was replaced with snap on cable suitable for this job.
Experiment 12 was an attempt to recreate experiment 10. Recreation was successful. However, the values for C and gamma are different.

Classification among 5 gestures:



The fifth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 16, gamma: 4, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 3 | 0.99 | 0.96 | 0.98 |
| 4 | 0.99 | 1.00 | 1.00 |
| 6 | 0.98 | 0.93 | 0.95 |
| 8 | 0.91 | 0.98 | 0.94 |
| avg | 0.97 | 0.97 | 0.97 |

More detailed Grid Search in the Vicinity of previously found parameters

C: [16, 19] step 0.1
Gamma: [2,4] step 0.1

Best parameters set found on development set:

C: 16.60000000000001, gamma: 3.200000000000001, kernel : rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

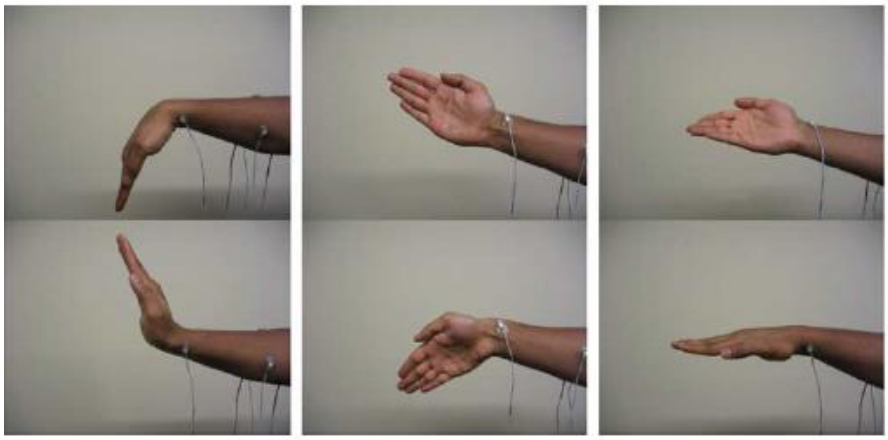
| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 3 | 1.00 | 0.96 | 0.98 |
| 4 | 0.99 | 1.00 | 1.00 |
| 6 | 0.98 | 0.94 | 0.96 |
| 8 | 0.91 | 0.98 | 0.94 |
| avg | 0.97 | 0.97 | 0.97 |

To be noted that in real time classification after a while the efficiency was dropping in time until it dropped to zero. It was classifying everything as gesture4.

# Experiment 13 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This time connection of the three electrodes that was plain cable now it was replaced with snap on cable suitable for this job.

Classification among 7 gestures:



The seventh  gesture named gesture0 is the complete relaxed hand gesture

Best parameters set found on development set:

C: 32, gamma: 1, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

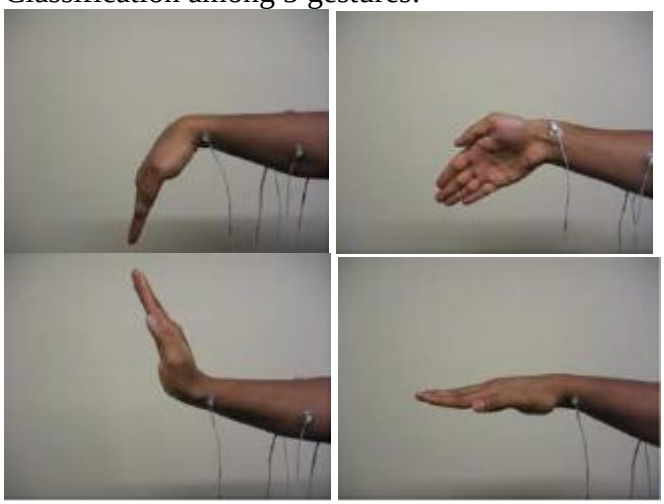
| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 3 | 0.94 | 0.97 | 0.95 |
| 4 | 1.00 | 0.98 | 0.99 |
| 5 | 0.96 | 0.97 | 0.96 |
| 6 | 0.91 | 0.91 | 0.91 |
| 7 | 0.66 | 0.70 | 0.68 |
| 8 | 0.67 | 0.63 | 0.65 |
| avg | 0.88 | 0.88 | 0.88 |

More detailed Grid Search in the Vicinity of previously found parameters
No better results were found.

# Experiment 14 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This time connection of the three electrodes that was plain cable now it was replaced with snap on cable suitable for this job.

Classification among 5 gestures:



The fifth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 64, gamma: 4, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 1.00 | 1.00 |
| 3 | 0.97 | 0.96 | 0.97 |
| 4 | 1.00 | 1.00 | 1.00 |
| 6 | 0.85 | 0.86 | 0.86 |
| 8 | 0.84 | 0.84 | 0.84 |
| avg | 0.93 | 0.93 | 0.93 |

More detailed Grid Search in the Vicinity of previously found parameters

C:[32,100] step 1
gama : [2,8] step 0.3

Best parameters set found on development set:

C: 71, gamma: 4.999999999999998, kernel': rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 3 | 0.95 | 0.97 | 0.96 |
| 4 | 1.00 | 1.00 | 1.00 |
| 6 | 0.86 | 0.85 | 0.86 |
| 8 | 0.84 | 0.84 | 0.84 |
| avg | 0.93 | 0.93 | 0.93 |

In real time classification fails to recognize every gesture.

So I record the gesures again and retrain to see if parameters c and gama have changed in time without changing electrodes that means that the signal has changed within an hour.
After the new recordings the resuts are:

Best parameters set found on development set:

C 4, gamma: 4, kernel: rbf

Detailed classification report:
The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.96 | 0.96 | 0.96 |
| 3 | 0.95 | 0.95 | 0.95 |
| 4 | 0.81 | 0.82 | 0.82 |
| 6 | 0.84 | 0.84 | 0.84 |
| 8 | 0.82 | 0.81 | 0.81 |
| avg | 0.87 | 0.87 | 0.87 |

Still failure to work in real time. So I record again a thrid time to see if parameters changed within half an hour at such extend.

Thirs recordings results with the same electrodes.
Best parameters set found on development set:

C: 2, gamma: 8, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

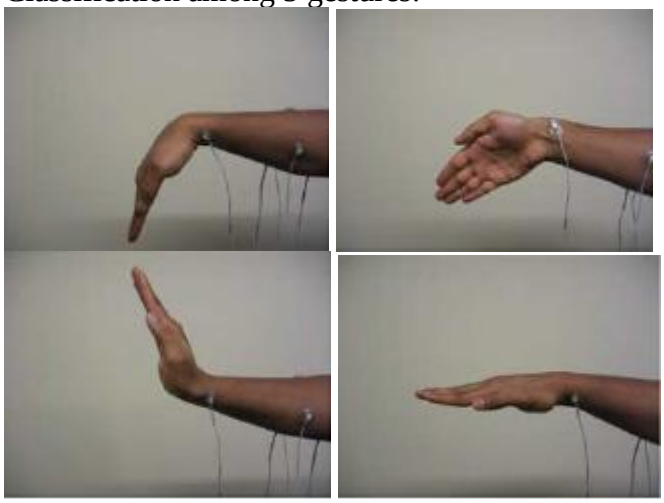
| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.98 | 0.99 |
| 3 | 0.68 | 0.87 | 0.76 |
| 4 | 0.97 | 0.75 | 0.85 |
| 6 | 0.70 | 0.67 | 0.68 |
| 8 | 0.68 | 0.68 | 0.68 |
| avg | 0.79 | 0.79 | 0.79 |

This time it recognizes gesture 3 and 4 and a few time 6 and 8. Gesture0 is not recognized at all. Battery has fallen to 5.9 V. Thus the range the signal can be amplified to  has decreased.

# Experiment 15 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This time connection of the three electrodes that was plain cable now it was replaced with snap on cable suitable for this job.

Classification among 5 gestures:



The fifth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 512, gamma: 0.125, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.97 | 0.99 | 0.98 |
| 3 | 0.99 | 0.97 | 0.98 |
| 4 | 1.00 | 0.99 | 1.00 |
| 6 | 0.96 | 0.98 | 0.97 |
| 8 | 0.97 | 0.96 | 0.96 |
| avg | 0.98 | 0.98 | 0.98 |

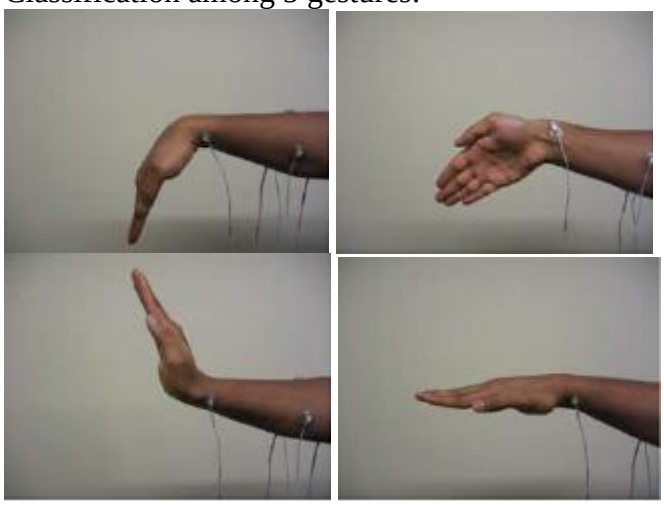
No matter the high success rate at offline testing. In real time it failed to recognize the gesture0. However, when the electrode on 1.7 V dropped by chance it started recognizing gesture0. Meaning that this electrode affects the real time processing. Thus, in the next experiment a DC filter is implemented and gestures rerecorded.

# Experiment 16 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This time connection of the three electrodes that was plain cable now it was replaced with snap on cable suitable for this job.

In this experiment a DC filter of the form x[n]-x[n-1] was implemented.

Classification among 5 gestures:



The fifth gesture named gesture0 is the complete relaxed hand gesture.

Best parameters set found on development set:

C: 256, gamma: 0.25,kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

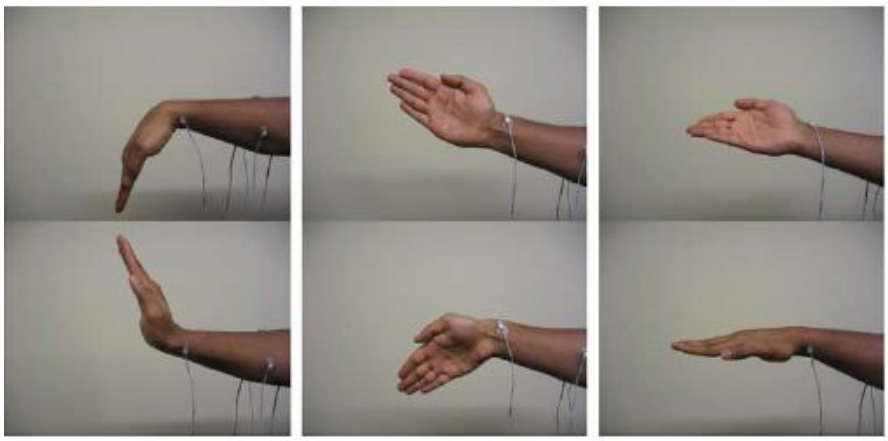
| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.99 | 0.97 | 0.98 |
| 3 | 0.90 | 0.95 | 0.93 |
| 4 | 1.00 | 0.95 | 0.98 |
| 6 | 0.90 | 0.87 | 0.89 |
| 8 | 0.90 | 0.94 | 0.92 |
| avg | 0.94 | 0.94 | 0.94 |

In real time it recognizes all gestures.

# Experiment 17 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This experiment is an attempt to control 3 motors in real time after the DC filter.

Classification among 7 gestures:



The seventh gesture named gesture0 is the complete relaxed hand gesture

Best parameters set found on development set:

C: 128, gamma: 0.0625, kernel: rbf

The model is trained on the full development set.
The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 1.00 | 1.00 |
| 3 | 0.98 | 0.93 | 0.95 |
| 4 | 0.98 | 0.99 | 0.99 |
| 5 | 0.98 | 0.96 | 0.97 |
| 6 | 0.88 | 0.92 | 0.90 |
| 7 | 0.86 | 0.88 | 0.87 |
| 8 | 0.96 | 0.98 | 0.97 |
| avg | 0.95 | 0.95 | 0.95 |

More detailed Grid Search in the Vicinity of previously found parameters

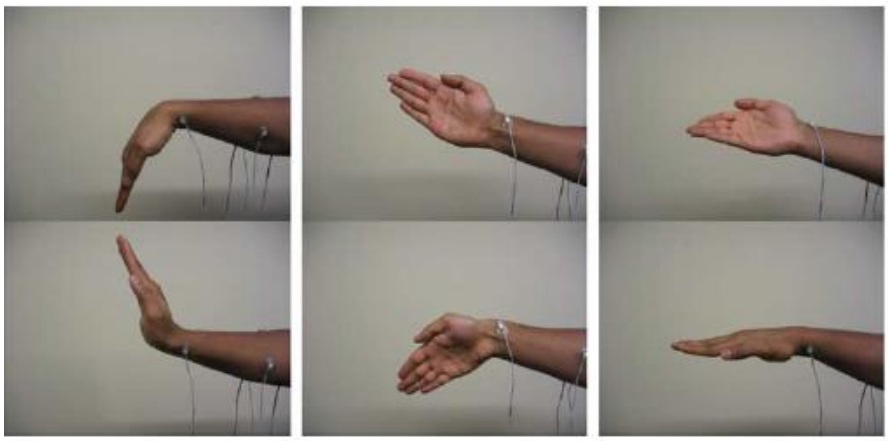
C: [64,200] step 1
Gamma: [ 0.001,0.1] step 0.001

One hour passed trying for this training and I had to stop it. There is a chance the signal has changed and needs to be retrained. Trying for the previous parameters though did not result in good real time results. These few misclassifications result in unpredictable movements. By observing the classification in real time, it is obvious that in time the classification decreases in efficiency.

# Experiment 18 Results from recording with 3 baby electrodes and 1 adult placed on the wrist.

This experiment is an attempt to control 3 motors in real time after the DC filter.

Classification among 7 gestures:



The seventh  gesture named gesture0 is the complete relaxed hand gesture

Best parameters set found on development set:

C: 256,  gamma: 0.0625, kernel': rbf
The model is trained on the full development set.

The scores are computed on the full evaluation set.

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 1.00 | 0.99 | 0.99 |
| 3 | 0.98 | 0.97 | 0.98 |
| 4 | 0.98 | 1.00 | 0.99 |
| 5 | 0.98 | 0.94 | 0.96 |
| 6 | 0.81 | 0.85 | 0.83 |
| 7 | 0.79 | 0.79 | 0.79 |
| 8 | 0.95 | 0.96 | 0.95 |
| avg | 0.92 | 0.92 | 0.92 |

In real time it was still unusable. What, is the most notable that gestures that were classified with good ratio, within 15 minutes of use the ratio dropped close to 0. Muscle fatigue that causes a little different signal has a great impact on the classification and the efficiency of the SVM. After a small rest of 5 minutes the gestures that were not recognized started to be recognized again.  Gesture 8 although it was recorded with the upper arm forming 90 degrees angle with the rest of the body, it stopped being recognized at this posture and started being recognized at 30 to 40 degrees. However, changing the training set only for gestures 0,3,4,6,8 gave really good real time performance. This is explained because the missclassification can lead to commands only to one other motor instead of two. Thus, the error for the position of the end effector is smaller and more predictable.

# Appendix B

In this part of the thesis the latest versions of the code segments explained in the software chapter will be presented.

## STM32 code written in C++ with Mbed library.

```
#include "mbed.h"

//sendingtheoutputoffilterforchannel1

#include "mbed.h"

#define pi         3.14159265358979323846

InterruptIn button(USER_BUTTON);
DigitalOut led(LED1);
Serial blu (PC_4, PA_10);
Ticker timer; // declaration of interruppt

//from which pins to sample. the last one is for the refference electrode
AnalogIn in1(PA_0);
AnalogIn in2(PA_1);
AnalogIn in3(PA_4);
AnalogIn in4(PB_0);
AnalogIn in5(PC_1);
AnalogIn in6(PC_0);
AnalogIn in7(PC_2);



//declaration of buffers for the notch filter for seven channels
float
xnbuffer1[2]={0},xnbuffer2[2]={0},xnbuffer3[2]={0},xnbuffer4[2]={0},xnbuffer5[2]={0},xnbuffe
r6[2]={0},xnbuffer7[2]={0}; //  here the previous values of analog in  will be stored for each
channel
float
ynbuffer1[2]={0},ynbuffer2[2]={0},ynbuffer3[2]={0},ynbuffer4[2]={0},ynbuffer5[2]={0},ynbuffe
r6[2]={0},ynbuffer7[2]={0};// here the previous values of the notch filter output will be stored for
each channel

//daclaration of variables to store the adc values from analog in
float xn1=0,xn2=0,xn3=0,xn4=0,xn5=0,xn6=0,xn7=0;
//x[n-1] buffer, x[n]buffer
float xn1prev=0,xn2prev=0,xn3prev=0,xn4prev=0;
float xn1now=0,xn2now=0,xn3now=0,xn4now=0;
//declaration of the filter output variables for each channel
float yn1=0,yn2=0,yn3=0,yn4=0,yn5=0,yn6=0,yn7=0;

//declaration of window buffers after the notch. Here 128 consecutive values from them notch filter
for each channel will be stored
```

```
float chan1[128]={0};
float chan2[128]={0};
float chan3[128]={0};
float chan4[128]={0};
float chan5[128]={0};
float chan6[128]={0};
float chan7[128]={0};

// declaration of feature vector to send
float x[7]={0};

//declaration of variable that is changed in the timer function and in button interrupt
int interval=0;
bool btn=0;

void button_press(){
   btn=!btn;
}

void timer_interrupt() {
   interval=1;
}

int main() {
blu.baud(9600); // with 9600 baud it loses samples if sending just the xn values

//declaration of main variables
int i=0; // 128 values  buffer index
int j;   //counter for rms for

// declaration of the 50Hz filter coefficients
float a1,a2,b1,r=0.99,w0=(50.0/1048.0)*pi;

//computing the coefficients
a1=2*r*cos(w0);
a2=r*r;
b1=2*cos(w0);

 // enabling the interrupts
timer.attach(&timer_interrupt, 1.0/2048.0);
button.fall(&button_press);


while (true) {

if(interval==1){
if(btn==1){

   //transfering  values from analog in objects in* to variables xn removing dc. Mbed analog in
returns values in [0,1]
   //implementaion of DC filter x[n]-x[n-1]
```

```
xn1now=(in1*3.6f);
xn2now=(in2*3.6f);
xn3now=(in3*3.6f);
xn4now=(in4*3.6f);

//xn1,2,3,4 is the output fo the DC filter
xn1=xn1now-xn1prev;
xn2=xn2now-xn2prev;
xn3=xn3now-xn3prev;
xn4=xn4now-xn4prev;
xn5=(in5*3.6f);
xn6=(in6*3.6f);
xn7=(in7*3.6f);

//filtering the 7 channels with the notch
yn1=a1*ynbuffer1[1]-a2*ynbuffer1[0]+xn1-b1*xnbuffer1[1]+xnbuffer1[0];
yn2=a1*ynbuffer2[1]-a2*ynbuffer2[0]+xn2-b1*xnbuffer2[1]+xnbuffer2[0];
yn3=a1*ynbuffer3[1]-a2*ynbuffer3[0]+xn3-b1*xnbuffer3[1]+xnbuffer3[0];
yn4=a1*ynbuffer4[1]-a2*ynbuffer4[0]+xn4-b1*xnbuffer4[1]+xnbuffer4[0];
yn5=a1*ynbuffer5[1]-a2*ynbuffer5[0]+xn5-b1*xnbuffer5[1]+xnbuffer5[0];
yn6=a1*ynbuffer6[1]-a2*ynbuffer6[0]+xn6-b1*xnbuffer6[1]+xnbuffer6[0];
yn7=a1*ynbuffer7[1]-a2*ynbuffer7[0]+xn7-b1*xnbuffer7[1]+xnbuffer7[0];



// sending the output of filter
//blu.printf("%0.5f\r\n",yn1);

//changing the values in the buffers for the next samples computation
//1st channel
ynbuffer1[0]=ynbuffer1[1];
ynbuffer1[1]=yn1;
xnbuffer1[0]=xnbuffer1[1];
xnbuffer1[1]=xn1;
//2nd channel
ynbuffer2[0]=ynbuffer2[1];
ynbuffer2[1]=yn2;
xnbuffer2[0]=xnbuffer2[1];
xnbuffer2[1]=xn2;
//3rd channel
ynbuffer3[0]=ynbuffer3[1];
ynbuffer3[1]=yn3;
xnbuffer3[0]=xnbuffer3[1];
xnbuffer3[1]=xn3;
//4th channel
ynbuffer4[0]=ynbuffer4[1];
ynbuffer4[1]=yn4;
xnbuffer4[0]=xnbuffer4[1];
xnbuffer4[1]=xn4;
//5th channel
ynbuffer5[0]=ynbuffer5[1];
ynbuffer5[1]=yn5;
```

```
xnbuffer5[0]=xnbuffer5[1];
xnbuffer5[1]=xn5;
//6th channel
ynbuffer6[0]=ynbuffer6[1];
ynbuffer6[1]=yn6;
xnbuffer6[0]=xnbuffer6[1];
xnbuffer6[1]=xn6;
//7th channel
ynbuffer7[0]=ynbuffer7[1];
ynbuffer7[1]=yn7;
xnbuffer7[0]=xnbuffer7[1];
xnbuffer7[1]=xn7;

//changing values for the x[n-1] buffer
xn1prev=xn1now;
xn2prev=xn2now;
xn3prev=xn3now;
xn4prev=xn4now;


// amassing the outputs of the 7 notches to the corresponding 128-values buffers
if(i<128){
  chan1[i]=pow(yn1,2);
  chan2[i]=pow(yn2,2);
  chan3[i]=pow(yn3,2);
  chan4[i]=pow(yn4,2);
  chan5[i]=pow(yn5,2);
  chan6[i]=pow(yn6,2);
  chan7[i]=pow(yn7,2);
  i++;
            }

 if(i==128){
   // rms computation
   // with this for Σxi^2 is computed   for each channel
     for(j=0;j<128;j++){
          x[0]=x[0]+chan1[j];
          x[1]=x[1]+chan2[j];
          x[2]=x[2]+chan3[j];
          x[3]=x[3]+chan4[j];
          x[4]=x[4]+chan5[j];
          x[5]=x[5]+chan6[j];
          x[6]=x[6]+chan7[j];
                      }
         // na mhn ksexasw na mhdenisw to x sto telos
           x[0]=sqrt(x[0]/128.0f);
           x[1]=sqrt(x[1]/128.0f);
           x[2]=sqrt(x[2]/128.0f);
           x[3]=sqrt(x[3]/128.0f);
           x[4]=sqrt(x[4]/128.0f);
           x[5]=sqrt(x[5]/128.0f);
           x[6]=sqrt(x[6]/128.0f);
```

```
            //blu.printf("%f,%f,%f,%f,%f,%f,%f\r\n",x[0],x[1],x[2],x[3],x[4],x[5],x[6]);
            //blu.printf("%f,%f,%f,%f\r\n",x[0],x[1],x[2],x[3]);
            blu.printf("%0.5f,%0.5f,%0.5f,%0.5f \r\n\0",x[0],x[1],x[2],x[3]);

            // re initialize x vector to zero
             for(j=0;j<7;j++){
              x[j]=0;        }//for j end

      i=0;

                        } // if i==128 end
                        }// if btn==1 end

 interval=0;

                        }// if interval ==1 end




   }// while true end
}// main end
```

## Python segment 1

```
import numpy as np
import sys
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
np.set_printoptions(threshold=sys.maxsize)
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn import preprocessing

# session 1 loading data to variables lists
s1g0="C:/Users/ece73/Desktop/recordings/session1/gesture0.txt"
s1g1="C:/Users/ece73/Desktop/recordings/session1/gesture1.txt"
s1g2="C:/Users/ece73/Desktop/recordings/session1/gesture2.txt"
s1g3="C:/Users/ece73/Desktop/recordings/session1/gesture3.txt"
s1g4="C:/Users/ece73/Desktop/recordings/session1/gesture4.txt"
s1g5="C:/Users/ece73/Desktop/recordings/session1/gesture5.txt"
```

```
s1g6="C:/Users/ece73/Desktop/recordings/session1/gesture6.txt"
s1g7="C:/Users/ece73/Desktop/recordings/session1/gesture7.txt"
s1g8="C:/Users/ece73/Desktop/recordings/session1/gesture8.txt"

ds1g0 = np.loadtxt(s1g0, delimiter=",")
ds1g1 = np.loadtxt(s1g1, delimiter=",")
ds1g2 = np.loadtxt(s1g2, delimiter=",")
ds1g3 = np.loadtxt(s1g3, delimiter=",")
ds1g4 = np.loadtxt(s1g4, delimiter=",")
ds1g5 = np.loadtxt(s1g5, delimiter=",")
ds1g6 = np.loadtxt(s1g6, delimiter=",")
ds1g7 = np.loadtxt(s1g7, delimiter=",")
ds1g8 = np.loadtxt(s1g8, delimiter=",")
##################################################

# session 2 loading data to variables lists
s2g0="C:/Users/ece73/Desktop/recordings/session2/gesture0.txt"
s2g1="C:/Users/ece73/Desktop/recordings/session2/gesture1.txt"
s2g2="C:/Users/ece73/Desktop/recordings/session2/gesture2.txt"
s2g3="C:/Users/ece73/Desktop/recordings/session2/gesture3.txt"
s2g4="C:/Users/ece73/Desktop/recordings/session2/gesture4.txt"
s2g5="C:/Users/ece73/Desktop/recordings/session2/gesture5.txt"
s2g6="C:/Users/ece73/Desktop/recordings/session2/gesture6.txt"
s2g7="C:/Users/ece73/Desktop/recordings/session2/gesture7.txt"
s2g8="C:/Users/ece73/Desktop/recordings/session2/gesture8.txt"

ds2g0 = np.loadtxt(s2g0, delimiter=",")
ds2g1 = np.loadtxt(s2g1, delimiter=",")
ds2g2 = np.loadtxt(s2g2, delimiter=",")
ds2g3 = np.loadtxt(s2g3, delimiter=",")
ds2g4 = np.loadtxt(s2g4, delimiter=",")
ds2g5 = np.loadtxt(s2g5, delimiter=",")
ds2g6 = np.loadtxt(s2g6, delimiter=",")
ds2g7 = np.loadtxt(s2g7, delimiter=",")
ds2g8 = np.loadtxt(s2g8, delimiter=",")
#####################################################################

# session 3 loading data to variables lists
s3g0="C:/Users/ece73/Desktop/recordings/session3/gesture0.txt"
s3g1="C:/Users/ece73/Desktop/recordings/session3/gesture1.txt"
s3g2="C:/Users/ece73/Desktop/recordings/session3/gesture2.txt"
s3g3="C:/Users/ece73/Desktop/recordings/session3/gesture3.txt"
s3g4="C:/Users/ece73/Desktop/recordings/session3/gesture4.txt"
s3g5="C:/Users/ece73/Desktop/recordings/session3/gesture5.txt"
s3g6="C:/Users/ece73/Desktop/recordings/session3/gesture6.txt"
s3g7="C:/Users/ece73/Desktop/recordings/session3/gesture7.txt"
s3g8="C:/Users/ece73/Desktop/recordings/session3/gesture8.txt"

ds3g0 = np.loadtxt(s3g0, delimiter=",")
ds3g1 = np.loadtxt(s3g1, delimiter=",")
ds3g2 = np.loadtxt(s3g2, delimiter=",")
ds3g3 = np.loadtxt(s3g3, delimiter=",")
```

```
ds3g4 = np.loadtxt(s3g4, delimiter=",")
ds3g5 = np.loadtxt(s3g5, delimiter=",")
ds3g6 = np.loadtxt(s3g6, delimiter=",")
ds3g7 = np.loadtxt(s3g7, delimiter=",")
ds3g8 = np.loadtxt(s3g8, delimiter=",")
#####################################################################

# session 4 loading data to variables lists
s4g0="C:/Users/ece73/Desktop/recordings/session4/gesture0.txt"
s4g1="C:/Users/ece73/Desktop/recordings/session4/gesture1.txt"
s4g2="C:/Users/ece73/Desktop/recordings/session4/gesture2.txt"
s4g3="C:/Users/ece73/Desktop/recordings/session4/gesture3.txt"
s4g4="C:/Users/ece73/Desktop/recordings/session4/gesture4.txt"
s4g5="C:/Users/ece73/Desktop/recordings/session4/gesture5.txt"
s4g6="C:/Users/ece73/Desktop/recordings/session4/gesture6.txt"
s4g7="C:/Users/ece73/Desktop/recordings/session4/gesture7.txt"
s4g8="C:/Users/ece73/Desktop/recordings/session4/gesture8.txt"

ds4g0 = np.loadtxt(s4g0, delimiter=",")
ds4g1 = np.loadtxt(s4g1, delimiter=",")
ds4g2 = np.loadtxt(s4g2, delimiter=",")
ds4g3 = np.loadtxt(s4g3, delimiter=",")
ds4g4 = np.loadtxt(s4g4, delimiter=",")
ds4g5 = np.loadtxt(s4g5, delimiter=",")
ds4g6 = np.loadtxt(s4g6, delimiter=",")
ds4g7 = np.loadtxt(s4g7, delimiter=",")
ds4g8 = np.loadtxt(s4g8, delimiter=",")
#####################################################################

# session 5 loading data to variables lists
s5g0="C:/Users/ece73/Desktop/recordings/session5/gesture0.txt"
s5g1="C:/Users/ece73/Desktop/recordings/session5/gesture1.txt"
s5g2="C:/Users/ece73/Desktop/recordings/session5/gesture2.txt"
s5g3="C:/Users/ece73/Desktop/recordings/session5/gesture3.txt"
s5g4="C:/Users/ece73/Desktop/recordings/session5/gesture4.txt"
s5g5="C:/Users/ece73/Desktop/recordings/session5/gesture5.txt"
s5g6="C:/Users/ece73/Desktop/recordings/session5/gesture6.txt"
s5g7="C:/Users/ece73/Desktop/recordings/session5/gesture7.txt"
s5g8="C:/Users/ece73/Desktop/recordings/session5/gesture8.txt"

ds5g0 = np.loadtxt(s5g0, delimiter=",")
ds5g1 = np.loadtxt(s5g1, delimiter=",")
ds5g2 = np.loadtxt(s5g2, delimiter=",")
ds5g3 = np.loadtxt(s5g3, delimiter=",")
ds5g4 = np.loadtxt(s5g4, delimiter=",")
ds5g5 = np.loadtxt(s5g5, delimiter=",")
ds5g6 = np.loadtxt(s5g6, delimiter=",")
ds5g7 = np.loadtxt(s5g7, delimiter=",")
ds5g8 = np.loadtxt(s5g8, delimiter=",")
#####################################################################

# Creating y lists for the classification for every session and every gesture
```

```
#session 1 gestures y matrices
ys1g0=[0 for i in range(len(ds1g0))]
ys1g1=[1 for i in range(len(ds1g1))]
ys1g2=[2 for i in range(len(ds1g2))]
ys1g3=[3 for i in range(len(ds1g3))]
ys1g4=[4 for i in range(len(ds1g4))]
ys1g5=[5 for i in range(len(ds1g5))]
ys1g6=[6 for i in range(len(ds1g6))]
ys1g7=[7 for i in range(len(ds1g7))]
ys1g8=[8 for i in range(len(ds1g8))]

#session 2 gestures y matrices
ys2g0=[0 for i in range(len(ds2g0))]
ys2g1=[1 for i in range(len(ds2g1))]
ys2g2=[2 for i in range(len(ds2g2))]
ys2g3=[3 for i in range(len(ds2g3))]
ys2g4=[4 for i in range(len(ds2g4))]
ys2g5=[5 for i in range(len(ds2g5))]
ys2g6=[6 for i in range(len(ds2g6))]
ys2g7=[7 for i in range(len(ds2g7))]
ys2g8=[8 for i in range(len(ds2g8))]

#session 3 gestures y matrices
ys3g0=[0 for i in range(len(ds3g0))]
ys3g1=[1 for i in range(len(ds3g1))]
ys3g2=[2 for i in range(len(ds3g2))]
ys3g3=[3 for i in range(len(ds3g3))]
ys3g4=[4 for i in range(len(ds3g4))]
ys3g5=[5 for i in range(len(ds3g5))]
ys3g6=[6 for i in range(len(ds3g6))]
ys3g7=[7 for i in range(len(ds3g7))]
ys3g8=[8 for i in range(len(ds3g8))]

#session 4 gestures y matrices
ys4g0=[0 for i in range(len(ds4g0))]
ys4g1=[1 for i in range(len(ds4g1))]
ys4g2=[2 for i in range(len(ds4g2))]
ys4g3=[3 for i in range(len(ds4g3))]
ys4g4=[4 for i in range(len(ds4g4))]
ys4g5=[5 for i in range(len(ds4g5))]
ys4g6=[6 for i in range(len(ds4g6))]
ys4g7=[7 for i in range(len(ds4g7))]
ys4g8=[8 for i in range(len(ds4g8))]

#session 5 gestures y matrices
ys5g0=[0 for i in range(len(ds5g0))]
ys5g1=[1 for i in range(len(ds5g1))]
ys5g2=[2 for i in range(len(ds5g2))]
ys5g3=[3 for i in range(len(ds5g3))]
ys5g4=[4 for i in range(len(ds5g4))]
ys5g5=[5 for i in range(len(ds5g5))]
ys5g6=[6 for i in range(len(ds5g6))]
```

```
ys5g7=[7 for i in range(len(ds5g7))]
ys5g8=[8 for i in range(len(ds5g8))]

#creating a complete dataset with y matrices for every session.
#session 1
ds1=np.concatenate((ds1g0,ds1g3,ds1g4,ds1g5,ds1g6,ds1g7,ds1g8))
ys1=np.concatenate((ys1g0,ys1g3,ys1g4,ys1g5,ys1g6,ys1g7,ys1g8))

#session 2
ds2=np.concatenate((ds2g0,ds2g3,ds2g4,ds2g5,ds2g6,ds2g7,ds2g8))
ys2=np.concatenate((ys2g0,ys2g3,ys2g4,ys2g5,ys2g6,ys2g7,ys2g8))

#session 3
ds3=np.concatenate((ds3g0,ds3g3,ds3g4,ds3g5,ds3g6,ds3g7,ds3g8))
ys3=np.concatenate((ys3g0,ys3g3,ys3g4,ys3g5,ys3g6,ys3g7,ys3g8))

#session 4
ds4=np.concatenate((ds4g0,ds4g3,ds4g4,ds4g5,ds4g6,ds4g7,ds4g8))
ys4=np.concatenate((ys4g0,ys4g3,ys4g4,ys4g5,ys4g6,ys4g7,ys4g8))

#session 5
ds5=np.concatenate((ds5g0,ds5g3,ds5g4,ds5g5,ds5g6,ds5g7,ds5g8))
ys5=np.concatenate((ys5g0,ys5g3,ys5g4,ys5g5,ys5g6,ys5g7,ys5g8))

#Creating the input for SVM. Complete data (x matrix) and y matrices.
datall=np.concatenate((ds1,ds2,ds3,ds4,ds5))
ysall=np.concatenate((ys1,ys2,ys3,ys4,ys5))
#
================================================================================
======
#
#
================================================================================
======

#selecting sessions for training
data=datall
y=ysall
#selecting testing session

meanvals=np.mean(data,0)
stds=np.std(data,0)
print (meanvals)
print(stds)

X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.3, random_state=0)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_transformed = scaler.transform(X_train)
X_test_transformed = scaler.transform(X_test)
```

```
# Split the dataset in two equal parts
#
# ============================================================================
# X_train, X_test, y_train, y_test = train_test_split(
#     data, y, test_size=0.2, random_state=0)
#
#
# ============================================================================
# Set the parameters by cross-validation
tuned_parameters = [{'kernel': ['rbf'], 'gamma': [2**(3),2**(2),2**(1),2**(0),2**(-1), 2**(-2),2**(-
3),2**(-4),2**(-5)],
                     'C': [2**(-10),2**(-9),2**(-8),2**(-7),2**(-6), 2**(-5),2**(-4),2**(-3),2**(-2),2**(-
1),2**(0),2**(1),2**(2),2**(3), 2**(4),2**(5),2**(6),2**(7),2**(8),2**(9),2**(10)],
                    'decision_function_shape':['ovr','ovo']},
                     {'kernel': ['linear'], 'C': [2**(-10),2**(-9),2**(-8),2**(-7),2**(-6), 2**(-5),2**(-
4),2**(-3),2**(-2),2**(-1),2**(0),2**(1),2**(2),2**(3),
2**(4),2**(5),2**(6),2**(7),2**(8),2**(9),2**(10)],
                     'decision_function_shape':['ovr','ovo']}]

scores = ['recall']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
    print()

    clf = GridSearchCV(SVC(), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train_transformed, y_train)

    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
    print()
    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test_transformed)
    print(classification_report(y_true, y_pred))
    print()
```

## Python segment 2

```
import numpy as np
import sys
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
np.set_printoptions(threshold=sys.maxsize)
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn import preprocessing

# session 1 loading data to variables lists
s1g0="C:/Users/ece73/Desktop/recordings/session1/gesture0.txt"
s1g1="C:/Users/ece73/Desktop/recordings/session1/gesture1.txt"
s1g2="C:/Users/ece73/Desktop/recordings/session1/gesture2.txt"
s1g3="C:/Users/ece73/Desktop/recordings/session1/gesture3.txt"
s1g4="C:/Users/ece73/Desktop/recordings/session1/gesture4.txt"
s1g5="C:/Users/ece73/Desktop/recordings/session1/gesture5.txt"
s1g6="C:/Users/ece73/Desktop/recordings/session1/gesture6.txt"
s1g7="C:/Users/ece73/Desktop/recordings/session1/gesture7.txt"
s1g8="C:/Users/ece73/Desktop/recordings/session1/gesture8.txt"

ds1g0 = np.loadtxt(s1g0, delimiter=",")
ds1g1 = np.loadtxt(s1g1, delimiter=",")
ds1g2 = np.loadtxt(s1g2, delimiter=",")
ds1g3 = np.loadtxt(s1g3, delimiter=",")
ds1g4 = np.loadtxt(s1g4, delimiter=",")
ds1g5 = np.loadtxt(s1g5, delimiter=",")
ds1g6 = np.loadtxt(s1g6, delimiter=",")
ds1g7 = np.loadtxt(s1g7, delimiter=",")
ds1g8 = np.loadtxt(s1g8, delimiter=",")
#####################################################

# session 2 loading data to variables lists
s2g0="C:/Users/ece73/Desktop/recordings/session2/gesture0.txt"
s2g1="C:/Users/ece73/Desktop/recordings/session2/gesture1.txt"
s2g2="C:/Users/ece73/Desktop/recordings/session2/gesture2.txt"
s2g3="C:/Users/ece73/Desktop/recordings/session2/gesture3.txt"
s2g4="C:/Users/ece73/Desktop/recordings/session2/gesture4.txt"
s2g5="C:/Users/ece73/Desktop/recordings/session2/gesture5.txt"
s2g6="C:/Users/ece73/Desktop/recordings/session2/gesture6.txt"
s2g7="C:/Users/ece73/Desktop/recordings/session2/gesture7.txt"
s2g8="C:/Users/ece73/Desktop/recordings/session2/gesture8.txt"

ds2g0 = np.loadtxt(s2g0, delimiter=",")
ds2g1 = np.loadtxt(s2g1, delimiter=",")
ds2g2 = np.loadtxt(s2g2, delimiter=",")
ds2g3 = np.loadtxt(s2g3, delimiter=",")
```

```
ds2g4 = np.loadtxt(s2g4, delimiter=",")
ds2g5 = np.loadtxt(s2g5, delimiter=",")
ds2g6 = np.loadtxt(s2g6, delimiter=",")
ds2g7 = np.loadtxt(s2g7, delimiter=",")
ds2g8 = np.loadtxt(s2g8, delimiter=",")
#####################################################################

# session 3 loading data to variables lists
s3g0="C:/Users/ece73/Desktop/recordings/session3/gesture0.txt"
s3g1="C:/Users/ece73/Desktop/recordings/session3/gesture1.txt"
s3g2="C:/Users/ece73/Desktop/recordings/session3/gesture2.txt"
s3g3="C:/Users/ece73/Desktop/recordings/session3/gesture3.txt"
s3g4="C:/Users/ece73/Desktop/recordings/session3/gesture4.txt"
s3g5="C:/Users/ece73/Desktop/recordings/session3/gesture5.txt"
s3g6="C:/Users/ece73/Desktop/recordings/session3/gesture6.txt"
s3g7="C:/Users/ece73/Desktop/recordings/session3/gesture7.txt"
s3g8="C:/Users/ece73/Desktop/recordings/session3/gesture8.txt"

ds3g0 = np.loadtxt(s3g0, delimiter=",")
ds3g1 = np.loadtxt(s3g1, delimiter=",")
ds3g2 = np.loadtxt(s3g2, delimiter=",")
ds3g3 = np.loadtxt(s3g3, delimiter=",")
ds3g4 = np.loadtxt(s3g4, delimiter=",")
ds3g5 = np.loadtxt(s3g5, delimiter=",")
ds3g6 = np.loadtxt(s3g6, delimiter=",")
ds3g7 = np.loadtxt(s3g7, delimiter=",")
ds3g8 = np.loadtxt(s3g8, delimiter=",")
#####################################################################

# session 4 loading data to variables lists
s4g0="C:/Users/ece73/Desktop/recordings/session4/gesture0.txt"
s4g1="C:/Users/ece73/Desktop/recordings/session4/gesture1.txt"
s4g2="C:/Users/ece73/Desktop/recordings/session4/gesture2.txt"
s4g3="C:/Users/ece73/Desktop/recordings/session4/gesture3.txt"
s4g4="C:/Users/ece73/Desktop/recordings/session4/gesture4.txt"
s4g5="C:/Users/ece73/Desktop/recordings/session4/gesture5.txt"
s4g6="C:/Users/ece73/Desktop/recordings/session4/gesture6.txt"
s4g7="C:/Users/ece73/Desktop/recordings/session4/gesture7.txt"
s4g8="C:/Users/ece73/Desktop/recordings/session4/gesture8.txt"

ds4g0 = np.loadtxt(s4g0, delimiter=",")
ds4g1 = np.loadtxt(s4g1, delimiter=",")
ds4g2 = np.loadtxt(s4g2, delimiter=",")
ds4g3 = np.loadtxt(s4g3, delimiter=",")
ds4g4 = np.loadtxt(s4g4, delimiter=",")
ds4g5 = np.loadtxt(s4g5, delimiter=",")
ds4g6 = np.loadtxt(s4g6, delimiter=",")
ds4g7 = np.loadtxt(s4g7, delimiter=",")
ds4g8 = np.loadtxt(s4g8, delimiter=",")
#####################################################################

# session 5 loading data to variables lists
```

```
s5g0="C:/Users/ece73/Desktop/recordings/session5/gesture0.txt"
s5g1="C:/Users/ece73/Desktop/recordings/session5/gesture1.txt"
s5g2="C:/Users/ece73/Desktop/recordings/session5/gesture2.txt"
s5g3="C:/Users/ece73/Desktop/recordings/session5/gesture3.txt"
s5g4="C:/Users/ece73/Desktop/recordings/session5/gesture4.txt"
s5g5="C:/Users/ece73/Desktop/recordings/session5/gesture5.txt"
s5g6="C:/Users/ece73/Desktop/recordings/session5/gesture6.txt"
s5g7="C:/Users/ece73/Desktop/recordings/session5/gesture7.txt"
s5g8="C:/Users/ece73/Desktop/recordings/session5/gesture8.txt"

ds5g0 = np.loadtxt(s5g0, delimiter=",")
ds5g1 = np.loadtxt(s5g1, delimiter=",")
ds5g2 = np.loadtxt(s5g2, delimiter=",")
ds5g3 = np.loadtxt(s5g3, delimiter=",")
ds5g4 = np.loadtxt(s5g4, delimiter=",")
ds5g5 = np.loadtxt(s5g5, delimiter=",")
ds5g6 = np.loadtxt(s5g6, delimiter=",")
ds5g7 = np.loadtxt(s5g7, delimiter=",")
ds5g8 = np.loadtxt(s5g8, delimiter=",")
######################################################################

# Creating y lists for the classification for every session and every gesture
#session 1 gestures y matrices
ys1g0=[0 for i in range(len(ds1g0))]
ys1g1=[1 for i in range(len(ds1g1))]
ys1g2=[2 for i in range(len(ds1g2))]
ys1g3=[3 for i in range(len(ds1g3))]
ys1g4=[4 for i in range(len(ds1g4))]
ys1g5=[5 for i in range(len(ds1g5))]
ys1g6=[6 for i in range(len(ds1g6))]
ys1g7=[7 for i in range(len(ds1g7))]
ys1g8=[8 for i in range(len(ds1g8))]

#session 2 gestures y matrices
ys2g0=[0 for i in range(len(ds2g0))]
ys2g1=[1 for i in range(len(ds2g1))]
ys2g2=[2 for i in range(len(ds2g2))]
ys2g3=[3 for i in range(len(ds2g3))]
ys2g4=[4 for i in range(len(ds2g4))]
ys2g5=[5 for i in range(len(ds2g5))]
ys2g6=[6 for i in range(len(ds2g6))]
ys2g7=[7 for i in range(len(ds2g7))]
ys2g8=[8 for i in range(len(ds2g8))]

#session 3 gestures y matrices
ys3g0=[0 for i in range(len(ds3g0))]
ys3g1=[1 for i in range(len(ds3g1))]
ys3g2=[2 for i in range(len(ds3g2))]
ys3g3=[3 for i in range(len(ds3g3))]
ys3g4=[4 for i in range(len(ds3g4))]
ys3g5=[5 for i in range(len(ds3g5))]
ys3g6=[6 for i in range(len(ds3g6))]
```

```
ys3g7=[7 for i in range(len(ds3g7))]
ys3g8=[8 for i in range(len(ds3g8))]

#session 4 gestures y matrices
ys4g0=[0 for i in range(len(ds4g0))]
ys4g1=[1 for i in range(len(ds4g1))]
ys4g2=[2 for i in range(len(ds4g2))]
ys4g3=[3 for i in range(len(ds4g3))]
ys4g4=[4 for i in range(len(ds4g4))]
ys4g5=[5 for i in range(len(ds4g5))]
ys4g6=[6 for i in range(len(ds4g6))]
ys4g7=[7 for i in range(len(ds4g7))]
ys4g8=[8 for i in range(len(ds4g8))]

#session 5 gestures y matrices
ys5g0=[0 for i in range(len(ds5g0))]
ys5g1=[1 for i in range(len(ds5g1))]
ys5g2=[2 for i in range(len(ds5g2))]
ys5g3=[3 for i in range(len(ds5g3))]
ys5g4=[4 for i in range(len(ds5g4))]
ys5g5=[5 for i in range(len(ds5g5))]
ys5g6=[6 for i in range(len(ds5g6))]
ys5g7=[7 for i in range(len(ds5g7))]
ys5g8=[8 for i in range(len(ds5g8))]

#creating a complete dataset with y matrices for every session.
#session 1
#session 1
ds1=np.concatenate((ds1g0,ds1g3,ds1g4,ds1g5,ds1g6,ds1g7,ds1g8))
ys1=np.concatenate((ys1g0,ys1g3,ys1g4,ys1g5,ys1g6,ys1g7,ys1g8))

#session 2
ds2=np.concatenate((ds2g0,ds2g3,ds2g4,ds2g5,ds2g6,ds2g7,ds2g8))
ys2=np.concatenate((ys2g0,ys2g3,ys2g4,ys2g5,ys2g6,ys2g7,ys2g8))

#session 3
ds3=np.concatenate((ds3g0,ds3g3,ds3g4,ds3g5,ds3g6,ds3g7,ds3g8))
ys3=np.concatenate((ys3g0,ys3g3,ys3g4,ys3g5,ys3g6,ys3g7,ys3g8))

#session 4
ds4=np.concatenate((ds4g0,ds4g3,ds4g4,ds4g5,ds4g6,ds4g7,ds4g8))
ys4=np.concatenate((ys4g0,ys4g3,ys4g4,ys4g5,ys4g6,ys4g7,ys4g8))

#session 5
ds5=np.concatenate((ds5g0,ds5g3,ds5g4,ds5g5,ds5g6,ds5g7,ds5g8))
ys5=np.concatenate((ys5g0,ys5g3,ys5g4,ys5g5,ys5g6,ys5g7,ys5g8))


#Creating the input for SVM. Complete data (x matrix) and y matrices.
datall=np.concatenate((ds1,ds2,ds3,ds4,ds5))
ysall=np.concatenate((ys1,ys2,ys3,ys4,ys5))
```

```
#
# ============================================================================
# ======
#
#
# ============================================================================
# ======

#selecting sessions for training
data=datall
y=ysall
#selecting testing session

meanvals=np.mean(data,0)
stds=np.std(data,0)
print (meanvals)
print(stds)

X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2, random_state=0)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_transformed = scaler.transform(X_train)
X_test_transformed = scaler.transform(X_test)



# Split the dataset in two equal parts
#
# ============================================================================
# ======
# X_train, X_test, y_train, y_test = train_test_split(
#     data, y, test_size=0.2, random_state=0)
#
#
# ============================================================================
# ======
# Set the parameters by cross-validation


c_grid=np.arange(64,200,1)
gama_grid=np.arange(0.001,0.1,0.001)




tuned_parameters = [{'kernel': ['rbf'], 'gamma': gama_grid,
                     'C':c_grid
                     }]

scores = ['recall']

for score in scores:
    print("# Tuning hyper-parameters for %s" % score)
```

```
    print()

    clf = GridSearchCV(SVC(), tuned_parameters, cv=5,
                       scoring='%s_macro' % score)
    clf.fit(X_train_transformed, y_train)


    print("Grid scores on development set:")
    print()
    means = clf.cv_results_['mean_test_score']
    stds = clf.cv_results_['std_test_score']
    for mean, std, params in zip(means, stds, clf.cv_results_['params']):
        print("%0.3f (+/-%0.03f) for %r"
              % (mean, std * 2, params))
    print()

    print("Detailed classification report:")
    print()
    print("The model is trained on the full development set.")
    print("The scores are computed on the full evaluation set.")
    print()
    y_true, y_pred = y_test, clf.predict(X_test_transformed)
    print(classification_report(y_true, y_pred))
    print()
    print("Best parameters set found on development set:")
    print()
    print(clf.best_params_)
```

## Python Segment 3

```
import serial
import time
import serial
import numpy as np
import sys
import matplotlib.pyplot as plt
from sklearn.model_selection import cross_val_score
np.set_printoptions(threshold=sys.maxsize)
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.svm import SVC
from sklearn import preprocessing

#opening the ports to be ready for comms
#ser.__del__()
#serout.__del__()
```

```
serout=serial.Serial('COM3', 9600) #com3 bluetooth out for NXT
ser=serial.Serial('COM12', 9600) #com9 prolific

# session 1 loading data to variables lists
s1g0="C:/Users/ece73/Desktop/recordings/session1/gesture0.txt"
s1g1="C:/Users/ece73/Desktop/recordings/session1/gesture1.txt"
s1g2="C:/Users/ece73/Desktop/recordings/session1/gesture2.txt"
s1g3="C:/Users/ece73/Desktop/recordings/session1/gesture3.txt"
s1g4="C:/Users/ece73/Desktop/recordings/session1/gesture4.txt"
s1g5="C:/Users/ece73/Desktop/recordings/session1/gesture5.txt"
s1g6="C:/Users/ece73/Desktop/recordings/session1/gesture6.txt"
s1g7="C:/Users/ece73/Desktop/recordings/session1/gesture7.txt"
s1g8="C:/Users/ece73/Desktop/recordings/session1/gesture8.txt"

ds1g0 = np.loadtxt(s1g0, delimiter=",")
ds1g1 = np.loadtxt(s1g1, delimiter=",")
ds1g2 = np.loadtxt(s1g2, delimiter=",")
ds1g3 = np.loadtxt(s1g3, delimiter=",")
ds1g4 = np.loadtxt(s1g4, delimiter=",")
ds1g5 = np.loadtxt(s1g5, delimiter=",")
ds1g6 = np.loadtxt(s1g6, delimiter=",")
ds1g7 = np.loadtxt(s1g7, delimiter=",")
ds1g8 = np.loadtxt(s1g8, delimiter=",")
#####################################################

# session 2 loading data to variables lists
s2g0="C:/Users/ece73/Desktop/recordings/session2/gesture0.txt"
s2g1="C:/Users/ece73/Desktop/recordings/session2/gesture1.txt"
s2g2="C:/Users/ece73/Desktop/recordings/session2/gesture2.txt"
s2g3="C:/Users/ece73/Desktop/recordings/session2/gesture3.txt"
s2g4="C:/Users/ece73/Desktop/recordings/session2/gesture4.txt"
s2g5="C:/Users/ece73/Desktop/recordings/session2/gesture5.txt"
s2g6="C:/Users/ece73/Desktop/recordings/session2/gesture6.txt"
s2g7="C:/Users/ece73/Desktop/recordings/session2/gesture7.txt"
s2g8="C:/Users/ece73/Desktop/recordings/session2/gesture8.txt"

ds2g0 = np.loadtxt(s2g0, delimiter=",")
ds2g1 = np.loadtxt(s2g1, delimiter=",")
ds2g2 = np.loadtxt(s2g2, delimiter=",")
ds2g3 = np.loadtxt(s2g3, delimiter=",")
ds2g4 = np.loadtxt(s2g4, delimiter=",")
ds2g5 = np.loadtxt(s2g5, delimiter=",")
ds2g6 = np.loadtxt(s2g6, delimiter=",")
ds2g7 = np.loadtxt(s2g7, delimiter=",")
ds2g8 = np.loadtxt(s2g8, delimiter=",")
#####################################################################

# session 3 loading data to variables lists
s3g0="C:/Users/ece73/Desktop/recordings/session3/gesture0.txt"
s3g1="C:/Users/ece73/Desktop/recordings/session3/gesture1.txt"
s3g2="C:/Users/ece73/Desktop/recordings/session3/gesture2.txt"
s3g3="C:/Users/ece73/Desktop/recordings/session3/gesture3.txt"
```

```
s3g4="C:/Users/ece73/Desktop/recordings/session3/gesture4.txt"
s3g5="C:/Users/ece73/Desktop/recordings/session3/gesture5.txt"
s3g6="C:/Users/ece73/Desktop/recordings/session3/gesture6.txt"
s3g7="C:/Users/ece73/Desktop/recordings/session3/gesture7.txt"
s3g8="C:/Users/ece73/Desktop/recordings/session3/gesture8.txt"

ds3g0 = np.loadtxt(s3g0, delimiter=",")
ds3g1 = np.loadtxt(s3g1, delimiter=",")
ds3g2 = np.loadtxt(s3g2, delimiter=",")
ds3g3 = np.loadtxt(s3g3, delimiter=",")
ds3g4 = np.loadtxt(s3g4, delimiter=",")
ds3g5 = np.loadtxt(s3g5, delimiter=",")
ds3g6 = np.loadtxt(s3g6, delimiter=",")
ds3g7 = np.loadtxt(s3g7, delimiter=",")
ds3g8 = np.loadtxt(s3g8, delimiter=",")
######################################################################

# session 4 loading data to variables lists
s4g0="C:/Users/ece73/Desktop/recordings/session4/gesture0.txt"
s4g1="C:/Users/ece73/Desktop/recordings/session4/gesture1.txt"
s4g2="C:/Users/ece73/Desktop/recordings/session4/gesture2.txt"
s4g3="C:/Users/ece73/Desktop/recordings/session4/gesture3.txt"
s4g4="C:/Users/ece73/Desktop/recordings/session4/gesture4.txt"
s4g5="C:/Users/ece73/Desktop/recordings/session4/gesture5.txt"
s4g6="C:/Users/ece73/Desktop/recordings/session4/gesture6.txt"
s4g7="C:/Users/ece73/Desktop/recordings/session4/gesture7.txt"
s4g8="C:/Users/ece73/Desktop/recordings/session4/gesture8.txt"

ds4g0 = np.loadtxt(s4g0, delimiter=",")
ds4g1 = np.loadtxt(s4g1, delimiter=",")
ds4g2 = np.loadtxt(s4g2, delimiter=",")
ds4g3 = np.loadtxt(s4g3, delimiter=",")
ds4g4 = np.loadtxt(s4g4, delimiter=",")
ds4g5 = np.loadtxt(s4g5, delimiter=",")
ds4g6 = np.loadtxt(s4g6, delimiter=",")
ds4g7 = np.loadtxt(s4g7, delimiter=",")
ds4g8 = np.loadtxt(s4g8, delimiter=",")
######################################################################

# session 5 loading data to variables lists
s5g0="C:/Users/ece73/Desktop/recordings/session5/gesture0.txt"
s5g1="C:/Users/ece73/Desktop/recordings/session5/gesture1.txt"
s5g2="C:/Users/ece73/Desktop/recordings/session5/gesture2.txt"
s5g3="C:/Users/ece73/Desktop/recordings/session5/gesture3.txt"
s5g4="C:/Users/ece73/Desktop/recordings/session5/gesture4.txt"
s5g5="C:/Users/ece73/Desktop/recordings/session5/gesture5.txt"
s5g6="C:/Users/ece73/Desktop/recordings/session5/gesture6.txt"
s5g7="C:/Users/ece73/Desktop/recordings/session5/gesture7.txt"
s5g8="C:/Users/ece73/Desktop/recordings/session5/gesture8.txt"

ds5g0 = np.loadtxt(s5g0, delimiter=",")
ds5g1 = np.loadtxt(s5g1, delimiter=",")
```

```
ds5g2 = np.loadtxt(s5g2, delimiter=",")
ds5g3 = np.loadtxt(s5g3, delimiter=",")
ds5g4 = np.loadtxt(s5g4, delimiter=",")
ds5g5 = np.loadtxt(s5g5, delimiter=",")
ds5g6 = np.loadtxt(s5g6, delimiter=",")
ds5g7 = np.loadtxt(s5g7, delimiter=",")
ds5g8 = np.loadtxt(s5g8, delimiter=",")
#########################################################################

# Creating y lists for the classification for every session and every gesture
#session 1 gestures y matrices
ys1g0=[0 for i in range(len(ds1g0))]
ys1g1=[1 for i in range(len(ds1g1))]
ys1g2=[2 for i in range(len(ds1g2))]
ys1g3=[3 for i in range(len(ds1g3))]
ys1g4=[4 for i in range(len(ds1g4))]
ys1g5=[5 for i in range(len(ds1g5))]
ys1g6=[6 for i in range(len(ds1g6))]
ys1g7=[7 for i in range(len(ds1g7))]
ys1g8=[8 for i in range(len(ds1g8))]

#session 2 gestures y matrices
ys2g0=[0 for i in range(len(ds2g0))]
ys2g1=[1 for i in range(len(ds2g1))]
ys2g2=[2 for i in range(len(ds2g2))]
ys2g3=[3 for i in range(len(ds2g3))]
ys2g4=[4 for i in range(len(ds2g4))]
ys2g5=[5 for i in range(len(ds2g5))]
ys2g6=[6 for i in range(len(ds2g6))]
ys2g7=[7 for i in range(len(ds2g7))]
ys2g8=[8 for i in range(len(ds2g8))]

#session 3 gestures y matrices
ys3g0=[0 for i in range(len(ds3g0))]
ys3g1=[1 for i in range(len(ds3g1))]
ys3g2=[2 for i in range(len(ds3g2))]
ys3g3=[3 for i in range(len(ds3g3))]
ys3g4=[4 for i in range(len(ds3g4))]
ys3g5=[5 for i in range(len(ds3g5))]
ys3g6=[6 for i in range(len(ds3g6))]
ys3g7=[7 for i in range(len(ds3g7))]
ys3g8=[8 for i in range(len(ds3g8))]

#session 4 gestures y matrices
ys4g0=[0 for i in range(len(ds4g0))]
ys4g1=[1 for i in range(len(ds4g1))]
ys4g2=[2 for i in range(len(ds4g2))]
ys4g3=[3 for i in range(len(ds4g3))]
ys4g4=[4 for i in range(len(ds4g4))]
ys4g5=[5 for i in range(len(ds4g5))]
ys4g6=[6 for i in range(len(ds4g6))]
ys4g7=[7 for i in range(len(ds4g7))]
```

```
ys4g8=[8 for i in range(len(ds4g8))]

#session 5 gestures y matrices
ys5g0=[0 for i in range(len(ds5g0))]
ys5g1=[1 for i in range(len(ds5g1))]
ys5g2=[2 for i in range(len(ds5g2))]
ys5g3=[3 for i in range(len(ds5g3))]
ys5g4=[4 for i in range(len(ds5g4))]
ys5g5=[5 for i in range(len(ds5g5))]
ys5g6=[6 for i in range(len(ds5g6))]
ys5g7=[7 for i in range(len(ds5g7))]
ys5g8=[8 for i in range(len(ds5g8))]

#creating a complete dataset with y matrices for every session.
#session 1
ds1=np.concatenate((ds1g0,ds1g3,ds1g4,ds1g5,ds1g6,ds1g7,ds1g8))
ys1=np.concatenate((ys1g0,ys1g3,ys1g4,ys1g5,ys1g6,ys1g7,ys1g8))

#session 2
ds2=np.concatenate((ds2g0,ds2g3,ds2g4,ds2g5,ds2g6,ds2g7,ds2g8))
ys2=np.concatenate((ys2g0,ys2g3,ys2g4,ys2g5,ys2g6,ys2g7,ys2g8))

#session 3
ds3=np.concatenate((ds3g0,ds3g3,ds3g4,ds3g5,ds3g6,ds3g7,ds3g8))
ys3=np.concatenate((ys3g0,ys3g3,ys3g4,ys3g5,ys3g6,ys3g7,ys3g8))

#session 4
ds4=np.concatenate((ds4g0,ds4g3,ds4g4,ds4g5,ds4g6,ds4g7,ds4g8))
ys4=np.concatenate((ys4g0,ys4g3,ys4g4,ys4g5,ys4g6,ys4g7,ys4g8))

#session 5
ds5=np.concatenate((ds5g0,ds5g3,ds5g4,ds5g5,ds5g6,ds5g7,ds5g8))
ys5=np.concatenate((ys5g0,ys5g3,ys5g4,ys5g5,ys5g6,ys5g7,ys5g8))

#Creating the input for SVM. Complete data (x matrix) and y matrices.
datall=np.concatenate((ds1,ds2,ds3,ds4,ds5))
ysall=np.concatenate((ys1,ys2,ys3,ys4,ys5))

data=datall
y=ysall

#splitting training and testng set randomly


#scaling the data sets
X_train, X_test, y_train, y_test = train_test_split(data, y, test_size=0.2, random_state=0)
scaler = preprocessing.StandardScaler().fit(X_train)
X_train_transformed = scaler.transform(X_train)
X_test_transformed = scaler.transform(X_test)

#SVM model traing and and testing the test set on it printing the results
#care for the parameters they are taken from previous recordings
```

```
model = SVC(kernel='rbf', C=256,gamma=0.0625,decision_function_shape='ovr')
model.fit(X_train_transformed,y_train)
y_true, y_pred = y_test, model.predict(X_test_transformed)
print(classification_report(y_true, y_pred))
print('model trained')


#reading rms vectors from microprocessors and deciding what the Nxt will do
#5th byte is the motor
#start motor on port A, 0x0D, 0x00, 0x80, 0x04, 0x00, 0x64, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00
#start motor on port B, 0x0D, 0x00, 0x80, 0x04, 0x01, 0x64, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00
#start motor on port C, 0x0D, 0x00, 0x80, 0x04, 0x02, 0x64, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00
#Stop  motor on port A, 0x0C, 0x00, 0x00, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00
#Stop  motor on port B, 0x0C, 0x00, 0x00, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00
#Stop  motor on port C, 0x0C, 0x00, 0x00, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00
#6th byte 0x64 for 50% power for anticlockwise direction and 0xCD for clockwise

while True:
 x = ser.readline()                        #reading a series of bytes
 g=x.decode("ascii")                         #decoding the byte to string
 g = np.fromstring( g, dtype=np.float, sep=',' )  #changing the string to an array of floats
 g=np.reshape(g,(1,-1))                           #reshaping to 2d array
 print(g)
 g_transformed=scaler.transform(g)              #normalizing vector with params from training set
 y_predg=model.predict(g_transformed)             #predicting which gesture this is
 print(y_predg)                            #printing prediction for me to know

 #If gesture0 detected stop all motor movements
 if y_predg==0:
   #stop port A
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
    #stop port B
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
    #stop port C
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
 #if gesture 3 detected open hand
 if   y_predg==3:
     #start port A anticlockwise
    packet = bytearray([  0x0D, 0x00, 0x80, 0x04, 0x00, 0x1E, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00])
```

```
    serout.write(packet)
    #stop port B
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
    #stop port C
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
  #if gesture 4 detected close hand
 if   y_predg==4:
     #start port A anticlockwise
    packet = bytearray([  0x0D, 0x00, 0x80, 0x04, 0x00, 0xE1, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00])
    serout.write(packet)
    #stop port B
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
    #stop port C
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
   #if gesture 5 detected go up
 if   y_predg==5:
     #start port B clockwise
    packet = bytearray([  0x0D, 0x00, 0x80, 0x04, 0x01, 0x1E, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00])
    serout.write(packet)
    #stop port A
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
     #stop port C
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
  #if gesture 6 detected go up
 if   y_predg==6:
     #start port B anticlockwise
    packet = bytearray([  0x0D, 0x00, 0x80, 0x04, 0x01, 0xE1, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00])
    serout.write(packet)
    #stop port A
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
     #stop port C
    packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x02, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
    serout.write(packet)
      #if gesture 8 detected go down
```

```
 if   y_predg==7:
      #start port C anticlockwise
     packet = bytearray([  0x0D, 0x00, 0x80, 0x04, 0x02, 0x1E, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00])
     serout.write(packet)
     #stop port A
     packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
     serout.write(packet)
     #stop port B
     packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
     serout.write(packet)
   #if gesture 8 detected go down
 if   y_predg==8:
      #start port C clockwise
     packet = bytearray([  0x0D, 0x00, 0x80, 0x04, 0x02, 0xE1, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00, 0x00])
     serout.write(packet)
     #stop port A
     packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x00, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
     serout.write(packet)
     #stop port B
     packet = bytearray([ 0x0C, 0x00, 0x00, 0x04, 0x01, 0x00, 0x07, 0x00, 0x00, 0x20, 0x00, 0x00,
0x00, 0x00])
     serout.write(packet)
```