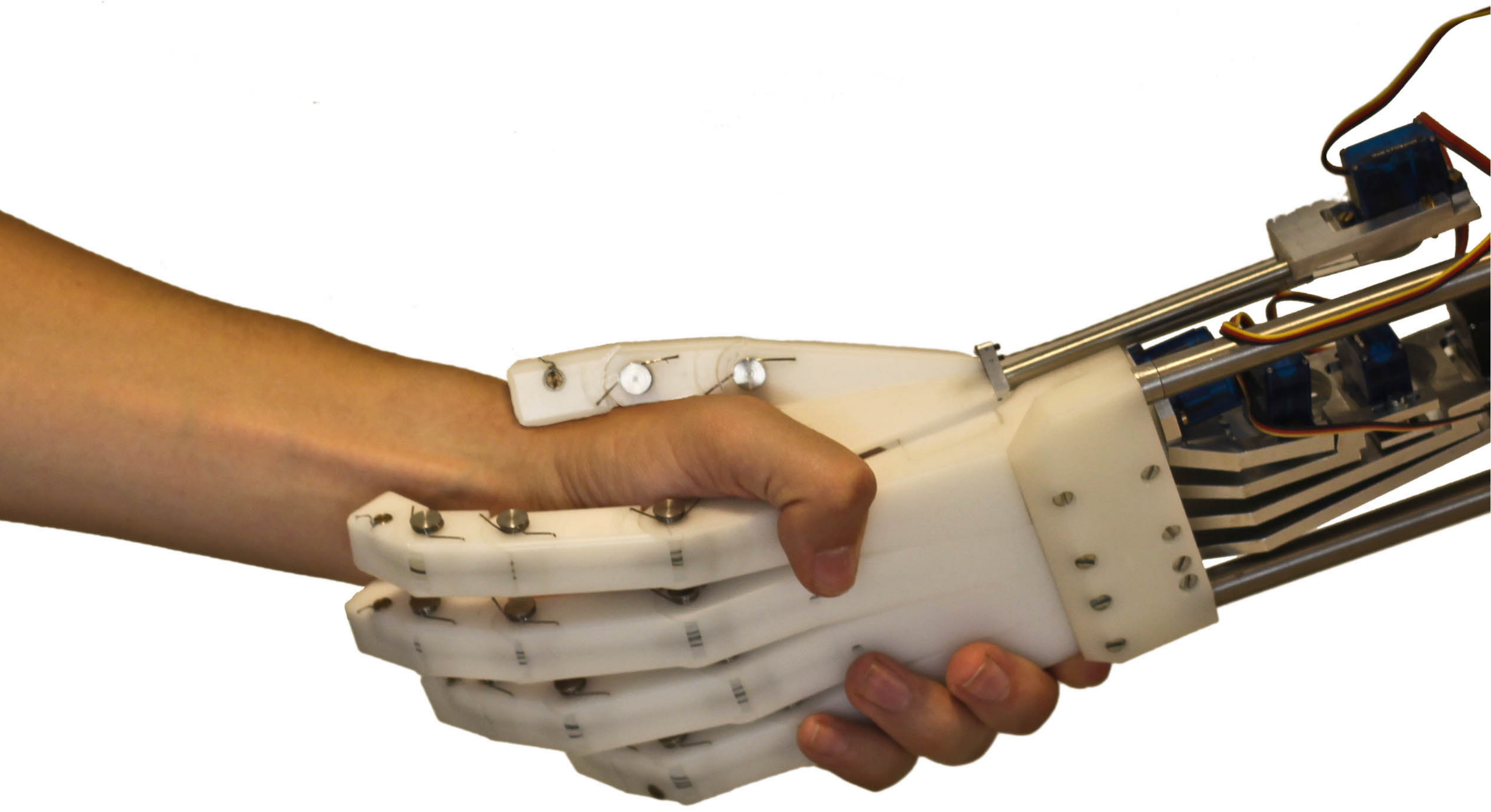

Classification of hand movements using multi-channel EMG

Master's Thesis in Bio-Medical Engineering

JOHAN BORGLIN

Department of Signals & Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2011
Master's Thesis 2011:1

Master of Science Thesis: Classification of hand movements using multi-channel EMG

Johan Borglin

Supervisors: Fredrik Sebelius and Christian Antfolk
Department of Measurement Technology
and Industrial Electrical Engineering
LTH, Lund, Sweden

Examiner: Professor Yngve Hamnerius
Department of Signals and Systems
Chalmers University of Technology

Could you give me a hand?

Abstract

Artificial neural networks (ANNs) were used to classify EMG signals from an arm. Using a amplifier card from the SmartHand project, 16-channel EMG signals were collected from the patients arm and filtered. After time-domain feature extraction, simple back-propagation training was used to train the networks. During the training the patient moved his fingers according to a predefined patter. After the training, the patient could move an artificial hand by duplicating the movements made during training.

One network was implemented for each finger, and good results were achieved when only one finger was contracted at any given time. The patient was able to use all the fingers in this way in a majority of the tests.

Additional tests were made with using several fingers at the same time. This was done successfully only in some of the tries.

The results show that ANNs are a possible candidate as a classifier for a project like the SmartHand project, but that great care has to be taken when training the networks, if advanced motions are desired.

The work was carried out at the Department for Electrical Measurements, the Faculty of Engineering (LTH), Lund University during the fall of 2010.

Acknowledgements

I would like to thank Christian Antfolk for all your help and encouragement during the whole project. Without your help this would have been an entirely different project, nowhere as good!

I would also like to thank Fredrik Sebelius for his help and support, as well as all the others who work at the Elmät institution. It was a pleasure to hang out in the lunch room with you.

Thanks to Yngve Hamnerius for being my examiner, and helping me with the bureaucracy!

A big thanks to Peter and Calle for offering me a great place to live, and to Krischansta nation for welcoming me with open arms when I had little to do in the evenings.

Johan Borglin, Gothenburg 11/9/11

Contents

List of abbreviations

EMG	ElectroMyoGraphy
KNN	K Nearest Neighbor (algorithm)
ANN	Artificial Neural Network
FIR	Finite Impulse Response
IIR	Infinite Impulse Response
FFT	Fast (discreet) Fourier Transform

1

Introduction

Many of the things we do in daily life, we do with our hands. It is therefore of greatest importance that we can replace a lost hand and restore as much functionality as possible, to ensure that the affected individual can continue to interact with the world. Several research projects around the world are in this moment working on a solution to this problem, and this report will describe how artificial neural networks (ANNs) can be used to help replacing a lost hand, by learning to recognize muscle signals that should have controlled the missing hand, and sending them to a new prosthesis.

Hopefully, this work will show that this approach to the problem of controlling a hand prosthesis is viable, and that it has benefits over other methods previously used.

Below follows a short overview of the project, to give the reader some orientation. Chapter 2 contains information about the background to the project as well as information on similar projects developed elsewhere. It also contains some information on the human hand. In chapter 5, the theory used in the project is presented and chapter 4 contains a description of how it was implemented. All results are presented in chapter 6 and discussed in 7, while the code written can be found in Appendix A.

1.1 Set-up and software overview

The project can be viewed as a chain of modules. Each module receives information from the previous, processes it in some way and sends it along to the next module. To make it easier to understand the rest of the report, an overview of this chain is given here. More information about each part of the chain can be found in the following chapters.

The first module consists of the arm of the patient. As will be discussed in chapter 2, most of the muscles controlling our fingers are located in the arm, and can remain functional even after a hand is lost. When they work they produce electrical signals which can be picked up by the 16 surface-mounted EMG sensors. The sensors are connected to amplifiers which in turn send the amplified signal via USB to the computer, where it is digitally filtered and features are extracted before the classification process begins.

The classification is divided into two distinct phases. In the training phase, the patient perform a predefined set of motions, and the data collected is tied to these known desired outputs. The network is then trained to match those outputs with the features of the collected data.

In the running phase, the trained network is used to classify any signal from the arm. This signal is then forwarded to an artificial hand, which mimics the patients movements.

A flowchart representation of the project can be found in figure 1.1.

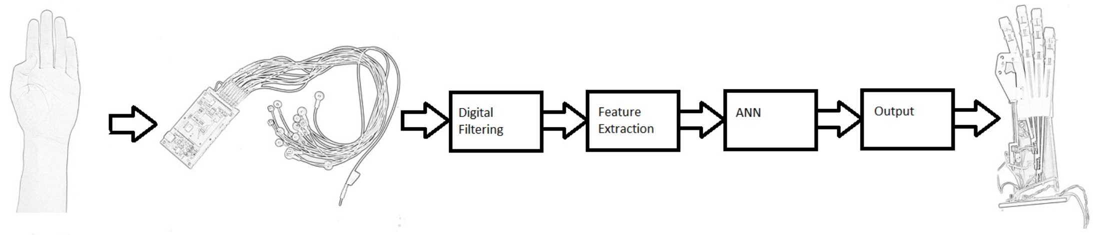

Figure 1.1: The project shown as a flowchart. A description of each element can be found here in the introduction or in chapter 5. Picture by the author.

2

Background

WHILE NOT STRICTLY PART OF THE PROJECT, the topics in this chapter should give a general idea about why projects such as this one are needed, and what is done today to help people who have lost their hand, or never had one.

2.1 The Human hand

Every year, several hundred thousand humans have one or more limbs amputated at a hospital, as a result of trauma or illness. Only a small portion of these are hand amputations, but for those affected this will of course completely change their lives. [1] This is because we use the hand to such a great extent in our daily lives, not only to interact with our surroundings, but also to gesticulate and in a way, to help us think better. [2]

Important for this project is that the muscles which enables the hand to move can be divided into two groups, intrinsic and extrinsic. The intrinsic muscles are located inside the hand and control some movements of the thumb and the little finger, while the extrinsic muscles are all located in the lower arm and connect to the rest of the fingers by tendons. They control most of the rest of the functions of the hand, and since they are not removed if only the hand and wrist are amputated, they will remain in the body. This is essential for prosthesis control, since the nerves controlling the hand are still connected to these muscles, which can be used to control the prosthesis instead. [3]

A picture of some of the muscles controlling the fingers can be found in figure 2.1.

2.2 Artificial Hands

Artificial hands are nothing new. One of the earliest mentions is of a roman general who fought with an iron arm back around the year 50 AD [5], and in the 16th century

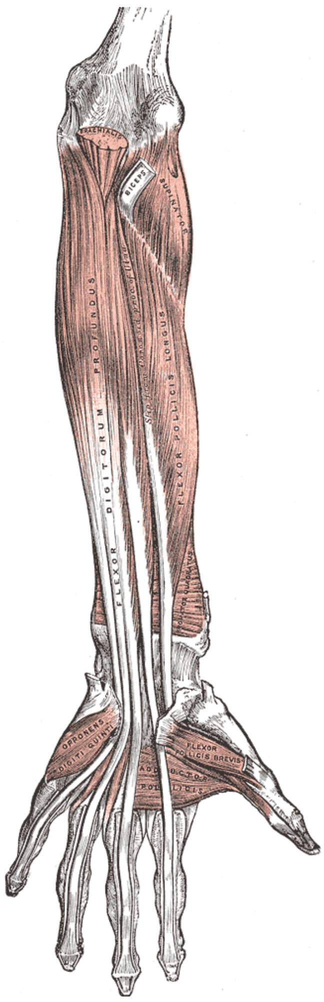

Figure 2.1: The muscles controlling the fingers are mostly located in the lower arm. This means that they can be present and functional even after an amputation.[4], used with permission.

the German imperial knight Götz von Berlichingen's iron arm could, according to contemporary sources, grip both a sword and feather pen thanks to an advanced system of springs and cogs. [6]

A lot has happened since then, and there are now several interesting projects running all over the world which aims at recreating a human hand. The SPRING-hand project in Italy, is much like the SmartHand project, a result of a desire to create a better and more natural hand prosthesis. It is a three-fingered prosthesis with two joints on each finger, but only uses one motor to control them, which limits the control one has over each finger, but reduces the weight. The feedback is provided by two strain gauges. They note that the task is complex but do not mention any control methods. It is interesting to note that several people who worked with the SPRING-hand went on to work with the SmartHand project. [7]

Bitzer et. al. use the DLR four-finger hand II, which is similar in construction to the previously mentioned hand, to try their control method. They note that the prosthesis must be very robust, yet intuitive, and that session independence is important. They use ten electrodes to detect muscle movements, and classify movements with ”support vector machines”, which is another kind of learning algorithms much like an artificiall neural network. They note that session independence is very hard to achieve, since the placement of the electrodes will change from session to session. They also note that their system is quite robust when it comes to arm movements. This is probably because they only train three fingers. (tumb, index and others) [8]

Constructing an artificial hand is not only about the hardware. Zollo et.al. uses a hand very similar to the ones mentioned above but focus more on the control process. By using a MatLAB model the hand was simulated and the feedback algorithms are tested. Two standard PD controllers were used for this. No feedback algorithms were written for my thesis, but it is something to keep in mind. [9]

A very important part of my work has been identifying features (see 4.3). Boostani and Moradi evaluated 19 different features for EMG classification in 2003, many of them more advanced than the ones covered in this work. They were evaluated based on their ability to classify, their noise tolerance and their calculation complexity. They conclude that energy wavelet coefficients in nine scales and cepstrum coefficients are the most efficient. None of these were used in my work due to their complexity. The features used in my work which were featured in the article received moderate scores. It was unfortunate that this article was found after the decision on what features to use was made. [10]

2.3 The SmartHand Project

The SmartHand project is a recently finished project in which a hand prosthesis was created which looked and operated like a real hand, and which also gave the wearer back the sensation of touch in the missing hand.

The hand is controlled via the signals from the patient’s brain to the remaining muscles in the arm which should have controlled the missing hand. When the patient tries to move the missing limb, the muscles will contract and an ElectroMyoGraphy (EMG) signal is generated. This signal is then measured and interpreted, the prosthesis can move to match the desired movement. [11]

A common problem in hand prostheses is the lack of feedback from the prosthesis to the user. This is often limited to visual signals where the user has to observe the hand to judge when to stop moving. This is neither optimal nor intuitive, and the SmartHand project tried a different approach. Using phantom limb sensations, the feeling that someone touches the missing limb when a specific area of the remaining arm is touched, the group was able to give feedback to the patients. The set-up consisted of sensors mounted on the SmartHand prosthesis. These sensors controlled motors which stimulated areas on the patients remaining arm where phantom limb sensations had previously been experienced. When the sensors were stimulated, the patient could now get the feeling that a finger from the missing hand was stimulated. In this way, some feedback from the hand could be given in a more intuitive way. A picture of the prosthesis can be found in figure 2.2 [11]

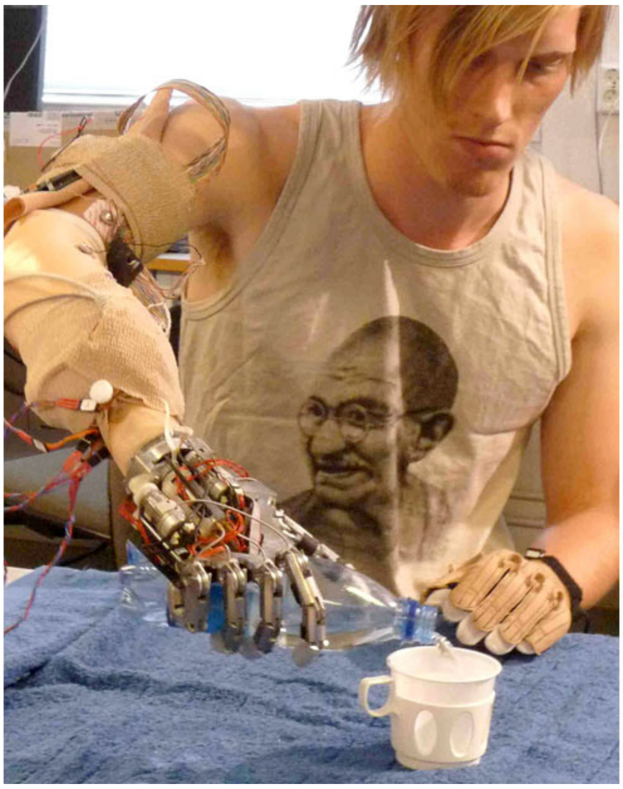

Figure 2.2: One of the patients from the SmartHand project testing the hand. Picture taken from [12], used with permission.

3

Problem definition and limitations

Below follows a short description of the problem that was being solved during this masters thesis project, along with a discussion about the limitations on the work that was decided upon beforehand. For more information about the classifiers and terms discussed here, please refer to section 4.1.1 and 4.1.2.

3.1 Problem definition

As discussed in chapter 2, losing one hand can seriously affect a persons daily life, and efforts are being made to lessen the impact this injury has. While it is true that the SmartHand project came a long way towards creating a functional hand prosthesis and give a solution to this problem, there were still ideas that they never tested. One of these were to substitute the ”K nearest neighbor”-method (KNN) used as a classifier with several ANNs running in parallel, and this is what was tested in this project.

In theory, this should lead to longer training times, but faster classifications while the hand is used. This is because an ANN requires a lot of computer power to converge, but once it has, the computations are fairly simple. A KNN is relatively fast to train, but needs a lot of data to operate. Using ANNs should make the hand more portable as the networks could be computed on a stationary computer and then downloaded into the hand.

In addition, the generalization abilities of the networks were tested by training one network for each finger, and then trying to get several of them to activate at the same time, based on the hypothesis that the muscle signal from several fingers can be seen as a superposition of the signal from each finger individually. This could allow for advanced movements of the hand without having to program all the special cases where a combination of fingers are used.

3.2 Limitations

Only simple back-propagation networks were tested during this project. ANNs can be modified in an almost infinite number of ways, which made it necessary to limit the tested designs to networks with one hidden layer. The number of neurons in this layer was varied as discussed in section 5.4, and the terms and definitions used are described in section 4.1.1 and 4.1.2.

No changes were made to either the amplifier card or the hand used to test the set-up. These limitations were set both to limit the size of the project to allow it to finish in time, and to make sure that all parts still were compatible with the other parts of the SmartHand project.

Data were collected from the author's arm, as actual amputees were not available.

4

Theory

IN THIS CHAPTER the general theory used in this master thesis project. This is intended to act as a reference for the later chapters where the theory is used. An overview of the software written for and used in this project can be found in chapter 1.1.

4.1 What is a classifier?

When working with statistics and other areas where large amounts of data are collected and analyzed, it is often necessary to sort the data points into sub-groups. This can be a very hard task for a human, who often aren't able to recognize which class a data point belongs to because of the large amounts of data contained in each data point. Instead, a digital classifier is used.

There are several different methods of creating a digital classifier, and two of the most common are described below. Common for all classifiers is that they work by supervised learning, where the classifier is trained on data with a known output, and then used on data of the same kind, allowing it to use its knowledge from the training data to classify the new data. It is important that the classifier can generalize and can sort data it has never encountered before, based on which sub-group it is most alike. [13]

4.1.1 Artificial neural networks

An artificial neural network is a classifier modeled after how the human brain works, which is very different from how one usually writes computer code.

A human brain contains an enormous amount of nerve cells, neurons. Each of these cells are connected to many other similar cells, creating a very complex network of signal transmission. Each cell collects inputs from all other neural cells it is connected to, and if it reaches a certain threshold, it signals to all the cells it is connected to.

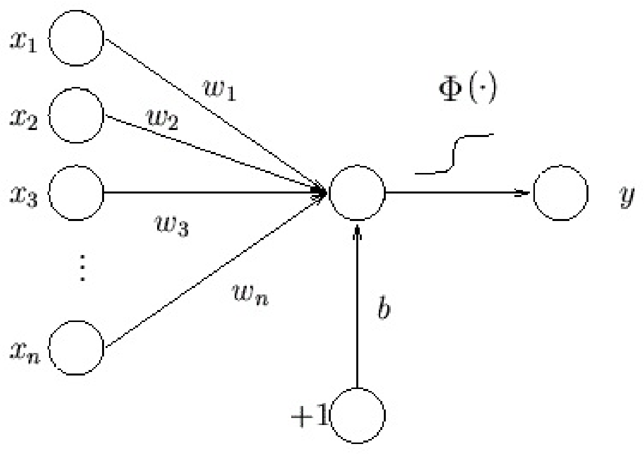

Figure 4.1: A graphical representation of a simple perceptron. Here y is the output signal, Φ is the activation function, n is the number of connections to the perceptron, w_i is the weight associated with the ith connection and x_i is the value of the ith connection. The b in the figure represents the threshold.[14]. Picture by the author.

When writing an ANN, this is mimicked by using a ”perceptron” as the basic unit instead of the neuron[1]. The perceptron can take several weighted inputs and summarize them, and if the combined input exceeds a threshold it will activate and send an output. Which output it sends is determined by the activation function and is often chosen to be between 0 and 1 or -1 and 1. Since the derivative of the activation function is often used in the training of the network, it is convenient if the derivative can be expressed in terms of the original function value, as few additional computations are needed to calculate the derivative in this case.

The equation for a perceptron can be written as

$$y = \Phi\left(\sum_{i=1}^{n} w_i x_i + b\right) \tag{4.1}$$

where y is the output signal, ϕ is the activation function, n is the number of connections to the perceptron, w_i is the weight associated with the ith connection and x_i is the value of the ith connection. b represents the threshold. A graphical representation can be found in figure 4.1. The threshold b is a neuron with a constant value of -1. By allowing the network to modify the weight associated with b, a dynamic threshold for when the perceptron activates is achieved.

This is a very simple design, and its strength can be shown when several perceptrons are combined and work together. The perceptrons are often organized in layers, where

[1]When working with ANNs, the perceptron is often referred to as a neuron. This is also true for this report

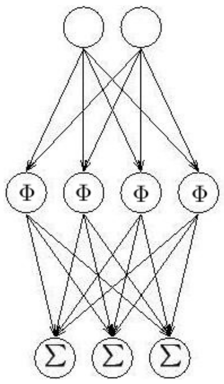

Figure 4.2: A graphical representation of an artificial neural network with one hidden layer. [14]. Picture by the author.

each layer takes input from the previous, applies weights and then signals to the next layer if appropriate. For a graphical representation, see figure 4.2.

As mentioned in 4.1, a classifier must be able to learn from examples and adapt. In an ANN, this is achieved by updating the weights associated with the connections between the layers. There are several ways of doing this, and most involve initializing the weights and fed the network an example. The error made by the network at the output is then calculated and feed backwards through a process called "back-propagation". This process is then used to update the weights, and by repeated use of this process, the network can learn to distinguish between several different classes. The exact equations involved vary from case to case, and the ones relevant to this project will be discussed in section 5.4.

To make the training more efficient, techniques such as momentum can be used. Momentum is used to find the right update step for the weights. If the step is too small the network will take too long to converge, while if it is to large the network might never converge and begin to oscillate instead. When using momentum, the step size is calculated dynamicaly during the run, on the basis that a weight which is changed often in the same direction most likely should be big, and can be changed faster. Another important techniqe is weight decay, which scales down all weights after every itteration. This means that large weights have to constantly be maintained to stay large, and avoids weights growing improportionally large.

One problem with the ANN approach is over-fitting of the data, which happens when the classifier becomes to good at recognizing the training examples, at the expense of not being able to recognize a general input. This can be avoided by cross-validation, where the network is trained on one set of data, and then evaluated on a separate one. When the error starts rising in the validation set, the network might be over-fitted. If previous networks are saved, the network can then be rolled back to the one which gave the smallest error. [15]

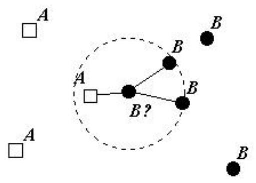

Figure 4.3: A KNN classifier with $K = 3$ compares the new sample with the 3 closest ones. The classifier classifies the new set as B, since 2/3 of the closest sets are labeled B. If k had been equal to 1, the set would have been classified as A. [17]. Picture by the author.

4.1.2 The K-nearest-neighbors algorithm

The K-nearest-neighbor algorithm is another classification method proven to be very useful in these kinds of problems [16]. Like the name implies, it is based on the idea that samples with similar parameter values should belong to similar classes. The classifier is trained simply by giving it access to a database containing pairs of parameter sets and classes, where each set of parameters belong to the class it is paired with.

When a new set of parameters is tested, the distance between the parameter set and all other sets is measured and the K sets in the training database which are closest are looked at. The classifier then assigns the set the class of the majority of the K nearest sets. This can be seen in figure 4.3.

A common variant of this method is to weight the sets by the distance from the new set. This way, sets closer to the new set are given more influence over the classification.

One downside of the method is that when the algorithm makes a new classification, the new set of parameters has to be compared to all the old sets in the training database. As the training database grows, this will make the method slower as more computations are needed. The database also has to be stored somewhere. This stands in contrast to an ANN, which was described in section 4.1.1, which takes longer to train, but does not get slower with more training, and which only requires the value of the current weights to be stored after the training. [18],[19]

4.2 Digital filtering

Every time a signal is gathered from somewhere there will be some amount of noise present. To remove this, one or more filters are usually needed. These can be hardware filters, built from for example amplifiers and resistors [20] or software filters built from equations in a computer. Both have their advantages, but in this project digital filters were chosen, to minimize the hardware modifications.

Just like with analog filters, digital filters come in several different types. The common ones are low-pass, high-pass, band-pass and band-stop. The names are rather self-explanatory. Low-pass filters stop frequencies over the cutoff frequency while high-pass stop the frequencies bellow the cutoff frequency. Band-pass and band-pass have two cutoff frequencies and stop the frequencies over and below, respectively between these cutoff frequencies.

A filter's cutoff frequency is defined as the frequency where the signal has dropped to $\sqrt{1/2}$ of the original signal.

Digital filters are usually represented as differential equations where the output is calculated from the current and previous inputs, in "finite impulse response"-filters (FIR), or from current and past inputs as well as past outputs in recursive, or "infinite impulse response"-filters (IIR). Some digital filters operating on data sets which are already available in full also use future inputs in the equation, but this is not possible when processing data live, like in this project.

A general digital filter can be written as;

$$y(n) = b_0x(n) + b_1x(n-1) + ... + b_Mx(n-M) - a_1y(n-1) - ... - a_Ny(n-N) \quad (4.2)$$

This equation is illustrated graphically in figure 4.4.

A filter where there exists an $a_n \neq 0$ is called a feedback filter, since it feeds back previous outputs to the equation. It is also known as an "infinite impulse response (IIR)"-filter, since it will give an often decaying, but in theory infinite response to an impulse input.

Filters where all $a_n = 0$ are called "finite impulse response (FIR)"-filters and only use previous and present inputs.

M and N denote the maximum delay used in the filter, and the maximum of these two is called the order of the filter. They could, in theory, be infinite but for natural reasons, they never are in practice.

A FIR generally needs a much higher order to achieve the same results as a IIR. [21]

These filters all operate in the time domain. It is possible to transform the signal into the frequency domain by applying a discreet Fourier-transform (FFT), but it is not practical in this case, as the amount of computations needed would be big.

4.3 Feature extraction

When dealing with large data sets, it is often impractical to work directly with the raw data. Not only because of the shear volume, but also because information can be hidden in a sequence of data-points which is not visible in a single data point. For this reason, feature extraction is often used.

In feature extraction, a large amount of data is described in terms of a few number of features. A good analogy is that instead of describing every small part of an object, we can describe it with its color, weight, height et.c. This will not give us all the information about the object, but enough to recognize it.

In statistical analysis, this is done with algorithms which are run on the collected data, returning a set of features describing it. A wide array of features exist, and which ones are the best depend heavily on the problem. As with digital filters, features can be calculated both in the time domain and in the frequency domain, but if the data

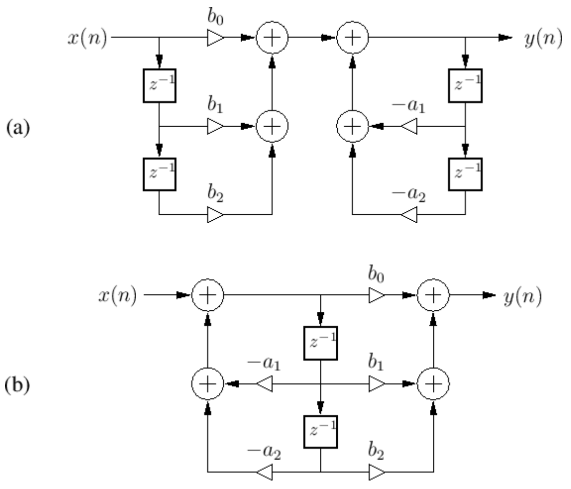

Figure 4.4: A possible signal flow chart of a digital filter with M,$N = 2$. z^{-1} denotes a one-sample delay. [21]. Picture by the author.

is collected in the time domain, a costly FFT is needed to transform the data into the frequency domain.

The features used in this project is described in section 5.2. [22] [23]

5

Method

THIS CHAPTER CONTAINS a thorough description of the experimental set-up used in this master thesis project, as well as the software written to process the collected data. For a more general description of the theory behind each part, please refer to chapter 4. The complete code can be found in appendix A.

5.1 Data collection and the amplifier

The data is collected from the arm with 16 electrodes connected to a special-built 8-channel bi-polar amplifier used in the SmartHand-project. "Red dot" monitoring electrodes from 3M^{tm} are placed according to a design developed previously in the SmartHand project. This design is illustrated in figure 5.1. The signal is then amplified 1000 times and sampled with a analog to digital converter from FTDI which connects to the host computer via USB. Pre-written methods supplied from FTDI was used to gather the data in C++. [16]

5.2 Digital filtering

The data collected is naturally noisy, and needs filtering to better representer the actual signal. In this project, three different types of filters were used. A band-pass filter was used to remove high and low frequency distortions. To remove the noise caused by the power grid, which operates at 50 Hz, a notch-filter was used twice. This is a band-stop filter which only stops frequencies at $50\,\text{Hz} \pm 2\,\text{Hz}$.

The filters where designed using MatLab's filter designer tool. Cutoff frequencies for the band-pass were chosen to 10 and 500 Hz as recommended in [24].

An example of the data filtration can be found in the results chapter, 6.1.

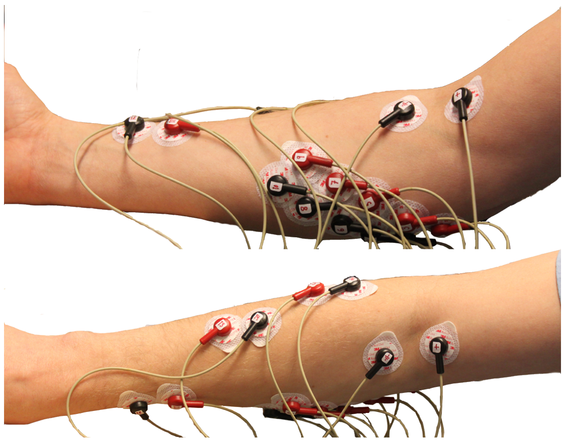

Figure 5.1: Electrode placement for collecting EMG data. 16 electrodes gather the signal, one acts as ground and is kept at $4V_{cc}$. Picture by the author.

5.3 Feature extraction

The data, read from the arm to the card and filtered, is now ready to undergo feature extraction. The feature extraction algorithms are run on groups of samples, epochs, each 300 samples long. They each overlap the previous by 200 samples. This means that a new epoch is ready whenever 100 new samples have been collected, giving the classifier an effective sample frequency of 16 Hz, which is only slightly lower than the 20 Hz proposed in [16].

In total, 7 features are used. As each of the 16 channels has its own set of features, this gives 112 features for the network to analyze. The features chosen were Time Domain features and are described below. [25]

5.3.1 Mean absolute value

This feature is an average over all the absolute values of the signal collected during the epoch.

$$y = \frac{1}{N}\sum_{i=1}^{N}|x_i| \tag{5.1}$$

5.3.2 Root mean square

Taken as the square root of the sum of all samples to the power of two, this feature is also a sort of average of the signal during the epoch.

$$y = \frac{1}{N}\sqrt{\sum_{i=1}^{N}x^2} \tag{5.2}$$

5.3.3 Slope sign change

This feature counts the number of times the sign of the slope of the signal change.

$$y = \sum_{i_1}^{N} f(x)$$

$$f(x) = \begin{cases} 1 \text{ if } (x_i < x_{i+1} \text{ AND } x_i < x_{i-1}) \text{ OR } (x_i > x_{i+1} \text{ AND } x_i > x_{k-1}) \\ 0 \text{ otherwise} \end{cases} \tag{5.3}$$

5.3.4 Waveform length

The waveform length is a measure of the variation of the EMG signal.

$$y = \sum_{i=1}^{N-1} (|x_{i+1} - x_i|) \tag{5.4}$$

5.3.5 Zero crossings

This feature measures the amount of times the signal crosses zero. To avoid interference from noise, a zero crossing is only registered if the difference between the two samples exceeds a pre-defined threshold (300 units).

$$y = \sum_{i_1}^{N-1} f(x)$$

$$f(x) = \begin{cases} 1 \text{ if } (x_i > 0\text{AND } x_{i+1} < 0) \text{ OR}(x_i < 0 \text{ AND}x_{i+1} > 0) \\ 0 \text{ otherwise} \end{cases} \tag{5.5}$$

5.3.6 Willson Amplitude

This feature counts the number of times the signal does a jump which exceeds a threshold. The threshold was chosen manually from inspection to give a signal when a muscle contracts and none when the muscle is at rest.

$$y = \sum_{i_1}^{N-1} f(|x_{i+1} - x_i|)$$

$$f(x) = \begin{cases} 1 \text{ if } x > 500) \\ 0 \text{ otherwise} \end{cases} \tag{5.6}$$

5.3.7 Variance

The variance of the signal gives a measure of how much the samples differ from each other. [26]

$$y = \frac{1}{N-1}\sum_{i=1}^{N}(x^2) \tag{5.7}$$

5.4 Artificial neural networks

As has been mentioned before in section 4.1.1, an artificial neural network can be varied in an almost infinite number of ways. It was decided early in the project to restrict the work to simpler layouts, using only a single layer of hidden neurons. The optimal number of neurons in this layer is not obvious, and a test algorithm was written which trains networks with different layouts and records the errors made. Results from this algorithm can be found in section 6.1 and the algorithm can be found in appendix A.

A simple $\tanh(x)$-function was used as activation function for the neurons. The $\tanh(x)$-function is well suited for this since it goes between -1 and 1 and it has a derivate of $1 - \tanh(x)^2$. This means that the functional value can be used to calculate the derivate without many additional calculations.

Simple back-propagation was employed to train the neurons, and to avoid unnecessary fluctuations of the weights, a momentum term was added as described in section 4.1.1.

Before each training session, all weights were set to a random number between 0.1 and -0.1 for the input layer and 1 and -1 for the hidden layer.

All the training patterns were shown to the network once in a random order before the network was evaluated with the patterns set aside as evaluation data. The error made by the network was then calculated as the sum the absolutes of the difference between what the network calculated and what was expected. If the error made was less than 0.5% the network was considered ready and the training was aborted early.

If no such network was found, the training continued until all training patterns had been shown 40 times. If the network was able to correctly classify 97% of the patterns at any point during the training, that configuration was used. If not, all weights were reset and the network was trained again in the same manner, but with new weights. If no network with 97% accuracy was found in 50 training repetitions, the best candidate so far was used.

To avoid over-fitting of the data only the network with the lowest evaluation error was saved and used. By saving the network with lowest evaluation error while still continuing the training, the problem with local error minima being taken for global error minima was made smaller.

5.4.1 Output smoothing

After the networks have classified the features, they will each output if they have identified a signal or not. Since the networks are no near perfect in recognizing the signals this output will be noisy. As a first step, a simple threshold filter is used, in which all outputs above 0 are set to 1 and the rest set to 0.

Even after this the signal might jump between “contract” and “extend” several times in a short interval, when the it is in fact “contract” or “extend” the whole time. To avoid having the hand moving every time a bit of noise makes the signal change, a technique called “majority vote” is used before the signal is sent to the hand.

This simply means that the last n outputs are examined, and if the number of “contract”-signals in that epoch exceeds a threshold, a “contract”-signal is sent to the hand. This adds a bit of lag to the set-up, since the out-put has to build up enough strength before it is sent to the hand, but avoids unnecessary and involuntary movements of the hand. In this case, n was set to 10, and the threshold to 5.

5.5 Robotic hand interface

For the last step of the project, a robotic hand was connected to the computer running the program using the serial port, and the software was modified to allow the hand to perform the movements identified by the ANN.

The hand was designed and constructed by Sebelius et.al. as a testing equipment, and is not the final “SmartHand”. It has 6 degrees of freedom, one for each finger and one extra to move the thumb into the hand. It is built from plastic and uses small motors and wires to move the fingers.

The output of the neural network is translated into a number which codes for the position of all 6 motors. The fingers can be set to any position between fully extended and contracted, but as the networks only return an ”contracted” or ”extended”-signal, only those two states are used.

6

Results and discussion

THE RESULTS FROM THE DIFFERENT STAGES of the process of translating an EMG-signal to hand movements are presented here. Wherever possible, the results are illustrated with graphs made in Matlab. For a detailed description of what is done in each step, please refer to chapter 5.

6.1 Filtering

Test code previously written for the SmartHand project was modified and used to test the filters. Real data was used for testing and the results can be seen in figure 6.1 and 6.2. As expected, the filters remove much of the interference from the power grid at 50 Hz and some of the high frequency distortions.

6.2 ANN design evaluation

To determine that the chosen network layout was good enough, several layouts were tested, with the number of neurons ranging from 10 to 110 in steps of 10 . All 11 configurations were tested 10 times each. During each test, the network was trained on the training data (740 patterns) 40 times. After training the network on all the patterns once, the network was evaluated against the validation data (240 patterns). The network which performed best on the validation data was saved and then ran again on previously unseen data (240 patterns). The error made in this data is shown in 6.3. As can be seen, the networks performed better with more neurons, but the increase in performance does not seem to be linear to the number of neurons. This is expected. No record of training times were made during the test runs, as this was not considered an important factor.

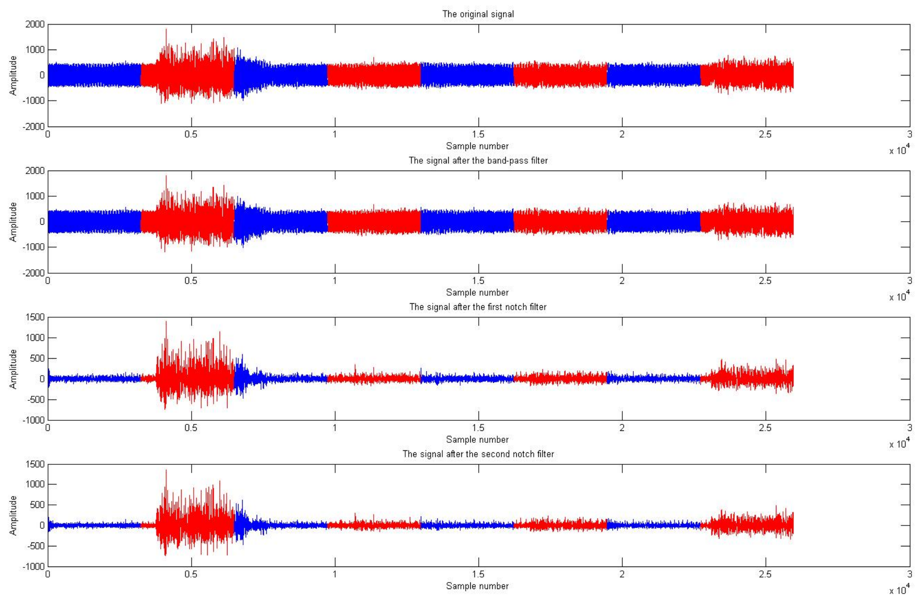

Figure 6.1: Results from filtering. These graphs show the signal before and after the digital filters were applied. The test was run on data collected from the experiment set-up. A blue signal indicates that the user was instructed to have all fingers extended, while a red signal indicates that the user was instructed to contract one finger, in order index finger to little finger. The delay between the instruction and the visible change in the graph is due to the users reaction time.

6.3 Final implementation

Figure 6.4 shows the results of the ANNs and the majority vote. The signal from the majority vote was then passed along to the hand which mimicked the movements of the user. The errors in the transitions between states are most likely a result of the reaction time of the user when the instructions on the screen change, and should not be seen as real classification errors but are instead a result of the user's reaction time during training. A thought is to discard all the data a certain time after a transition to avoid this problem.

A closer look at how each finger is reconstructed is shown in figure 6.5.

When only one finger was used at a time, the hand was quite accurate, and the user could quickly control it. However, only certain positions of the fingers led to a reaction in the artificial hand, indication that the ANNs are not perfect. Once these positions were found, control could be established.

While the results are no near perfect, it has been shown that artificial neural networks can be used to classify EMG signals and translate them to movements of an artificial hand. However, several problems remain to be fixed.

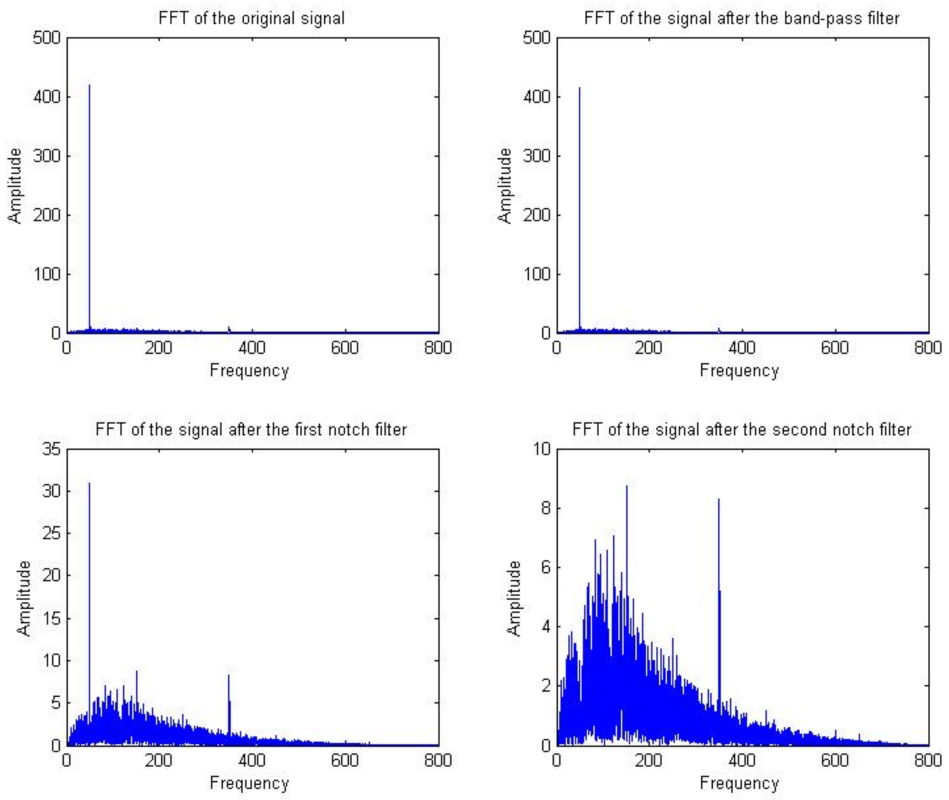

Figure 6.2: Results from filtering. These graphs show a FFT of the signal, before and after the filters. As can be seen, the 50 Hz peak dominates the signal before the notch filter is applied. The signal still contains some peaks after the filters, but nothing nearly as large. Please note that the scale is not constant throughout the graphs.

Firstly, the networks do not converge every time, and occasionally it will take quite a long time before a trained network is ready. This can hopefully be helped by a careful study of which network layout should be used, along with better update rules and better training sets.

The networks sometimes have problems identifying the movements of the user. This is a serious problem, since the usefulness of a prosthesis depends a lot on its ability to function reliably, as it does not matter if a hand works in 99% of the cases when you try to carry a cup of hot coffee and drop it in your lap. This could be avoided by rewriting the code to only open and close on certain signals instead of closing when a signal is detected and remaining open otherwise. This would make the hand more reliable but would make the interface less intuitive.

Another problem is that to get accurate control, the networks have to be retrained every time the sensors are remounted on the arm, since the positions of the sensors

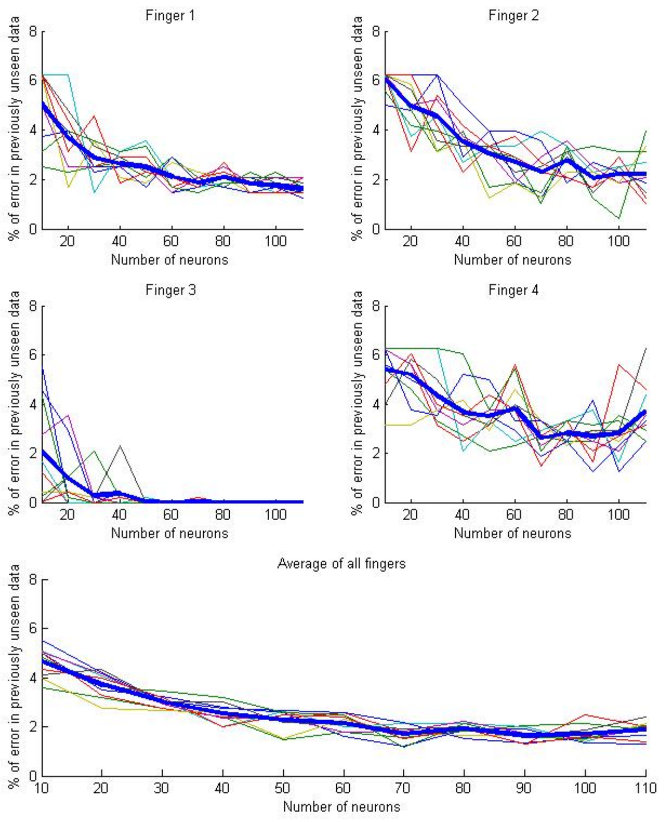

Figure 6.3: The best results achieved over several test training sessions with a varying number of neurons in the hidden layer. The error is calculated as the percentage of data points that was miss-classified. The thick line indicates the average for that number of neurons.

are bound to shift. The networks also drop in performance over time as the user gets more tired and sweaty, as this makes the signal from the arm change. If a user is to be fitted with a hand such as the SmartHand permanently, more accurate sensor placements can be made with plastic shells placed on the arm. This shell has fixed positions for the sensors. It was not tested how much the networks can detect after removing and replacing the sensors in the same general area.

6.3.1 Multi-finger movements

One goal of this project was to see if multi-finger movements was achievable using ANNs trained only on single finger movements. This was achieved several times with two fingers at once and occasionally with three fingers. However, the stochastic nature of

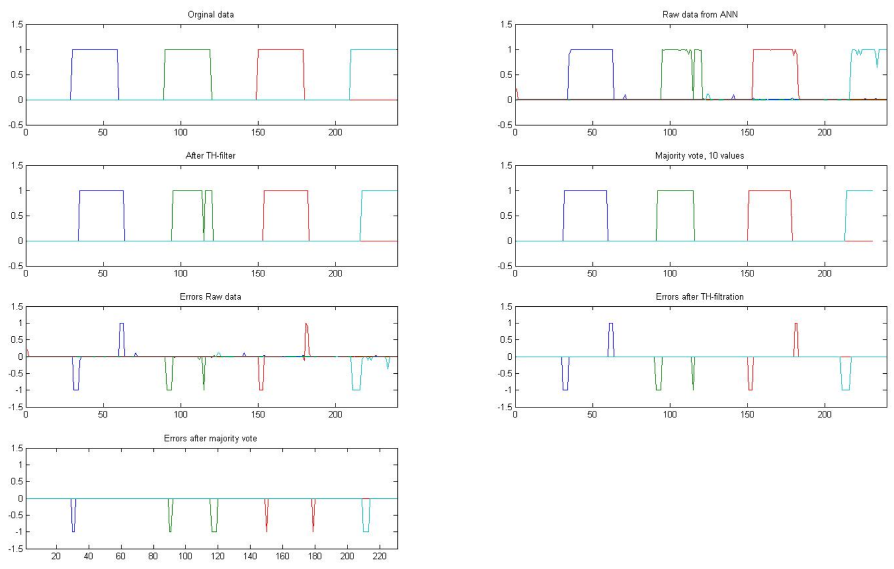

Figure 6.4: The first graph shows the signal as it was shown to the user, who mimicked it Each color represents one finger, and a high signal indicates that that finger is contracted. The second shows the raw output of the ANNs and the third the same output after threshold filtering. The forth one shows the final result that is being sent to the robotic hand after the majority vote. The signal is clearly recognizable, but the majority vote makes the "contract"-signals thinner, as the signal needs a few samples to be classified as "contract" before transitioning to a "contract"-state. The final three graphs show the errors made by the system in the raw data and after threshold and majority vote algorithms were applied. A "1" indicates that the system detected contraction when no contraction was expected and a -1 indicates that the system expected contraction but did not detect any. As can be seen, the system manages to reconstruct the data, but the signal is shorter due to the majority vote.

the ANNs and the placement of the sensors makes it very hard to determine what causes the network to be able to sometimes only identify one and sometimes up to three fingers at once.

6.4 Error sources

It is very hard to say exactly what causes the errors in classification, as there are several possibilities.

Perhaps the most obvious are measurement errors, where the sensors pick up signals from other muscles than they are placed on. This is certainly a problem in multi-finger movements, where several muscles are active at the same time. A more carefully designed

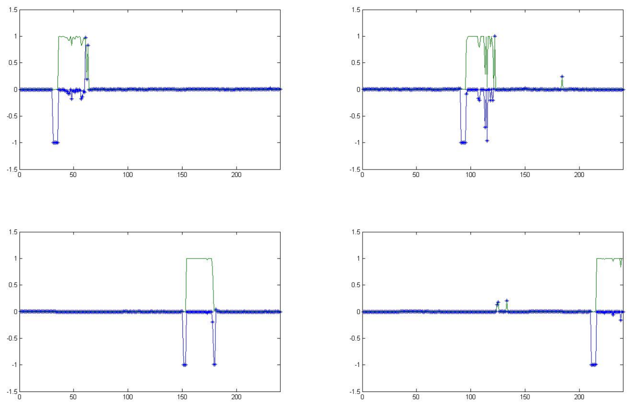

Figure 6.5: A closer look at how each finger was reconstructed by the best ANN. Each graph represent one finger. The green line is the reconstructed signal and the blue is the error made by the system.

training program could probably help with this problem. If more random muscles are active during the training and not only the ones performing the desired movement, perhaps the networks could get better at distinguishing the signal from the noise.

Errors could also show up as a result of changes in the positions of the sensors, changes in the skin-sensor interface as a result from sweating and the like. The best way to counter this would probably be to try to make the networks more general. This could probably be done by using better training data and being more careful not to over-train them. The user will eventually get tired from doing the same movements over and over again, resulting in a different signal. This could also be avoided with more careful training.

7

Conclusions

In this project, EMG-signals from an arm was used to control an artificial hand and make it mimic the movements of an arm. This was achieved using equipment developed for the SmartHand project. The data was digitally filtered and features were extracted, and these features were then classified by the networks after training. The end result was that the patient could move the muscles in his arm as if he was moving a finger, and the artificial hand would respond by mimicking that movement, as long as only one finger was moved at the given time.

One goal of the project was to evaluate how well ANNs perform in comparison to KNNs. Like expected, the ANNs take longer to prepare, but are relatively fast once the training is completed. Due to lack of time and access to software, no comparison in performance was made between the two systems, but they should perform on the same level once tested.

The other goal was to test if several ANNs could run in parallel and classify the movements of one finger each. Trials show that this is very possible for at least two fingers, but that it only works from time to time. This could probably be improved by better training routines and sensor placement.

Future work would also include a more careful evaluation of the network design, where different update rules and pre-processing of the data is used. It would also be very interesting to explore how the number of detectors influence the results. With only 16 detectors, a lot of information has to be gathered from each of these, and it is very hard to cover all the muscles involved. If there were many more, simpler detectors all muscles could be covered. This would probably allow us to significantly cut down on the number of features needed to correctly classify each movement. In the ideal case, the number of detectors are great enough to classify every movement simply by detecting the presence or absence of a strong EMG signal, since the distribution of EMG signal correspond to muscle activity and movements.

Bibliography

[1] National Limb Loss Information Center, Amputation statistics by cause, limb loss in the united states (2008).
URL http://www.amputee-coalition.org/fact_sheets/amp_stats_cause.html

[2] C. L. Taylor, R. J. Schwarz, The anatomy and mechanics of the human hand, Artificial Limbs 2 (1955) 22–35.

[3] B. J. Wilhelmi, Hand anatomy (2011).
URL http://emedicine.medscape.com/article/1285060-overview#a1

[4] H. Gray, Anatomy of the human body, Philadelphia: Lea and Febiger, 1918.

[5] Prosthesis.
URL http://en.wikipedia.org/wiki/Prosthesis

[6] Götz von berlichingen.
URL http://en.wikipedia.org/wiki/G%C3%B6tz_von_Berlichingen

[7] M. Carrozza, C. Suppo, F. Sebastiani, B. Massa, F. Vecchi, R. Lazzarini, M. Cutkosky, P. Dario, The spring hand: Development of a self-adaptive prosthesis for restoring natural grasping, Autonomous Robots 16 (2004) 125–141, 10.1023/B:AURO.0000016863.48502.98.
URL http://dx.doi.org/10.1023/B:AURO.0000016863.48502.98

[8] S. Bitzer, P. van der Smagt, Learning emg control of a robotic hand: towards active prostheses, in: Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on, 2006, pp. 2819 –2823.

[9] L. Zollo, S. Roccella, E. Guglielmelli, M. Carrozza, P. Dario, Biomechatronic design and control of an anthropomorphic artificial hand for prosthetic and robotic applications, Mechatronics, IEEE/ASME Transactions on 12 (4) (2007) 418 –429.

[10] R. Boostani, M. H. Moradi, Evaluation of the forearm emg signal features for the control of a prosthetic hand, Physiological Measurement 24 (2) (2003) 309.
URL `http://stacks.iop.org/0967-3334/24/i=2/a=307`

[11] C. Cipriani, C. Antfolk, C. Balkenius, B. Rosen, G. Lundborg, M. Carrozza, F. Sebelius, A novel concept for a prosthetic hand with a bidirectional interface: A feasibility study, Biomedical Engineering, IEEE Transactions on 56 (11) (2009) 2739–2743.

[12] The smart hand.
URL `http://www.zootpatrol.com/index.php/2009/11/the-smart-hand/`

[13] L. Crnkovic-Dodig, Classifier showdown (2006).
URL `http://blog.peltarion.com/2006/07/10/classifier-showdown/`

[14] A. Honkela, Nonlinear switching state-space models.
URL `http://www.hiit.fi/u/ahonkela/dippa/dippa.html`

[15] S. Haykin, Neural Networks and Learning machines, Pearson Education, 2009.

[16] F. Sebelius, M. Axelsson, N. Danielsen, J. Schouenborg, T. Laurell, Real-time control of a virtual hand, Technology and Disability SMC-6 (4) (1976) 325 –327.

[17] S. Savchenko, Editing nearest neighbour decision rules.
URL `http://cgm.cs.mcgill.ca/~godfried/teaching/projects.pr.98/sergei/project.html`

[18] T. Cover, P. Hart, Nearest neighbor pattern classification, Information Theory, IEEE Transactions on 13 (1) (1967) 21 – 27.

[19] S. A. Dudani, The distance-weighted k-nearest-neighbor rule, Systems, Man and Cybernetics, IEEE Transactions on 17 (3) (2005) 131 –141.

[20] T. R. Kuphalt, Fundamentals of electrical engineering and electronics (2011).
URL `http://www.vias.org/feee/filters_02.html`

[21] J. O. Smith, Introduction to Digital Filters with Audio Applications, W3K Publishing, 2007.

[22] I. Guyon, A. Elisseeff, An introduction to feature extraction (2004).

[23] M. Zecca, S. Micera, M. C. Carrozza, P. Dario, Control of multifunctional prosthetic hands by processing the electromyographic signal, Critical Reviews trade; in Biomedical Engineering 30 (4-6) (2002) 459–485.

[24] J.-U. Chu, I. Moon, Y.-J. Lee, S.-K. Kim, M.-S. Mun, A supervised feature-projection-based real-time emg pattern recognition for multifunction myoelectric hand control, Mechatronics, IEEE/ASME Transactions on 12 (3) (2007) 282 –290.

[25] B. Hudgins, P. Parker, R. Scott, A new strategy for multifunction myoelectric control, Biomedical Engineering, IEEE Transactions on 40 (1) (1993) 82–94.

[26] D. Joshi, K. Kandpal, S. Anand, Feature evaluation to reduce false triggering in threshold based emg prosthetic hand, in: R. Magjarevic, N. A. Abu Osman, F. Ibrahim, W. A. B. Wan Abas, H. S. Abdul Rahman, H.-N. Ting (Eds.), 4th Kuala Lumpur International Conference on Biomedical Engineering 2008, Vol. 21 of IFMBE Proceedings, Springer Berlin Heidelberg, 2008, pp. 769–772.

A

C++ Code

A.1 Main program

The main program used to train the ANNs and use the hand.

A.1.1 Declarations

```
#include "stdafx.h"
#include <windows.h>
#include <stdio.h>
#include "ftd2xx.h"
#include "iostream"
#include <mmsystem.h>
#include "fstream"
#include <cstdlib>
#include <math.h>
#include "time.h"
#include <string>
#include <winbase.h>
#include "atlstr.h"

using namespace std;

#pragma comment(lib, "winmm.lib")

#define AD_MCC_START_BUSYWAIT_MS 1
#define AD_MCC_START_TIMEOUT_MS 400
#define AD_MCC_RUN_BUSYWAIT_MS 1
#define AD_MCC_ADJ_MARGIN_MS 1
#define NR_FEAT 7

#define NRF 112
#define NRC 16
#define NRFEAT 7
#define MXPT 1500
#define RPS 40
```

```
#define NR_ANN 4

//////////////////////////////////////////////////////////////////////

#define USB_DESCRIPTOR "USB <-> Serial" // USB Descriptor
#define LATENCY_TIMER 2 // Latency timer for USB port milliseconds

#define MAX_TESTTIME_MS 5
#define MAX_TIMEOUT 0 // Was 50 before

#define COMBUFFERSIZE 8192

#define EMG_USB_SPEED 2000000

#define READBUFFSIZE 2880 // Should be 2880

#define BUFFSIZER 5760 //8192 16000 Should be much shorter maybe 2
#define BLOCKLENGTH 80          //50    //100   //200 // Should be 80
#define USB_BLOCK_SIZE  65536 // Multiple of 64, 16384, 65536
#define BLOCKSIZE       2880    //1700  //3400  //6800 //3600 Should be 2880
#define NBRCHANNELS 16

#define CLASSESTESTED 4


#define DEV_LENGTH 20  // Compute time deviation mean

#define BUFFERSIZE 4096
#define QUESIZE 8192
#define BUFFSIZEW 3600

HANDLE hComPort;
BYTE Buffer[BUFFERSIZE];
BYTE Data[BUFFERSIZE];

BYTE TestBuffer[BUFFERSIZE];

int ConvertedData[QUESIZE];
int QueueCounter;
int OldQueueCounter;
int LastStartByte;
int OldLastStartByte;
unsigned long NWrite;
BOOL DataReady;
BOOL DataCorrupt;

//int ConvertedData[BLOCKLENGTH*NBRCHANNELS];
//float RawData[BLOCKLENGTH*NBRCHANNELS];

int FirstData[NBRCHANNELS];

int ReceiveCounter;

unsigned long NRead;
int extraBytesRead;



// CRC constants
const int order = 16;
const unsigned long polynom = 0x8005;
const int direct = 1;
const unsigned long crcinit = 0x0000;
const unsigned long crcxor = 0x0000;
//const int refin = 0;
//const int refout = 0;

const unsigned long crcmask = 65535;
const unsigned long crchighbit = 32768;
const unsigned long crcinit_nondirect = 0;


// For FTDI chip dll
FT_STATUS ftStatus;
FT_DEVICE_LIST_INFO_NODE *devInfo;
FT_HANDLE ftHandle;

BYTE byteData[BUFFSIZEW];

bool openCard();

void readCardToFile(int length,int number);

int menu();

void trainFromFile2(int length, int overlap, int segment, int number, int typ);

void getFeatures(float *f,int segment, float *feat);

float ANN2(int target,float prev_error);

void filter3(float* c, float* f, int segment, int channel, bool firstrun);

void live(float nr_feat,int segment, int overlap);

void openComPort(int port);

void setCommsTimeOut();

void closeComPort();
```

A.1.2 Function: main

```
int main(int argc, char* argv[])
{
  srand(time(NULL));

  int length = 0,val,overlap=200,segment=300,rlength=102400,rnumber=1,nr_features=4,val_nr=2,val_length=25600;

  float errorANN,prev_error=10*rlength;


  do
  {
    val=menu();
    switch (val)
    {
    case 1:
      cout<<"Read how long for training? ";
      cin >> length;
      if(openCard())
      {
        readCardToFile(length,1);
      }
      cout<<"Read how long for validation? ";
      cin >> length;
      if(openCard())
      {
        readCardToFile(length,2);
      }
      break;
    case 2:
      cout<<"Calculating features"<<endl;
      trainFromFile2(rlength,overlap,segment,rnumber,1);
      trainFromFile2(val_length,overlap,segment,val_nr,0);
      cout<<"Features extracted"<<endl<<"------------------------\n";
      for (int i=1;i<NR_ANN+1;i++)
      {
        prev_error=10*rlength;
        for(int j=0;j<50;j++)
        {
          cout<<"ANN run "<<i<<" try "<<j+1<<endl;
          errorANN=ANN2(i,prev_error);

          if(errorANN<prev_error)
          {
            prev_error=errorANN;
          }
          if(errorANN<3)
          {
            cout<<"Match found after "<<j+1<<" tries"<<endl<<"------------------------\n";
            break;
          }
          cout<<"*** No match found ***"<<endl<<"------------------------\n";
        }

      }
      cout<<"Done! Enter to continue";
      getchar();
      getchar();
      break;
    case 3:
      if(openCard())
      {
        live(4,segment,overlap);
      }
      break;
    case 4:
      cout << "Read from what file? ";
      cin >> rnumber;
      cout << "Read how long? ";
      cin >> rlength;
      prev_error=10*rlength;
      cout << "Enter segment length: ";
      cin >> segment;
      cout << "Enter overlap: ";
      cin >> overlap;
      cout << "How many features? " ;
      cin >> nr_features;
      cout<<"Validate with what file? ";
      cin >> val_nr;
      cout<<"How long? ";
      cin >> val_length;
      break;
    case 0:
      break;
    default:
      cout << "Invalid\n";
      break;
    }
  }while(val!=0);

  FT_Close(&ftHandle);

  return 0;
}
```

A.1.3 Function: openCard

```
bool openCard()
{
  ftStatus = FT_Open(0,&ftHandle);
  if (ftStatus == FT_OK){printf("Ok 1");}
  else{printf("fail\n");}
  ftStatus = FT_ResetDevice(ftHandle);
  if (ftStatus == FT_OK){printf(", 2");}
  else{printf("fail\n");}
  ftStatus = FT_SetUSBParameters(ftHandle, USB_BLOCK_SIZE, 0); //BLOCKSIZE
  if (ftStatus == FT_OK){printf(", 3");}
  else{printf("fail\n");}
  ftStatus = FT_SetBaudRate(ftHandle, EMG_USB_SPEED); // Set baud rate to 300 aliased
  if (ftStatus == FT_OK){printf(", 4");}
  else{printf("fail\n");}
  ftStatus = FT_SetDataCharacteristics(ftHandle,FT_BITS_8,FT_STOP_BITS_1, FT_PARITY_NONE);
  if (ftStatus == FT_OK){printf(", 5");}
  else{printf("fail\n");}
  ftStatus = FT_SetFlowControl(ftHandle, FT_FLOW_NONE, 0, 0);
  if (ftStatus == FT_OK){printf(", 6");}
  else{printf("fail\n");}
  ftStatus = FT_SetTimeouts(ftHandle, MAX_TIMEOUT, 1000);
  if (ftStatus == FT_OK){printf(", 7");}
  else{printf("fail\n");}
  ftStatus = FT_SetLatencyTimer(ftHandle, LATENCY_TIMER);
  if (ftStatus == FT_OK){printf(", 8");}
  else{printf("fail\n");}
  ftStatus = FT_Purge(ftHandle, FT_PURGE_RX | FT_PURGE_TX); //Purge both Rx and Tx buffers
  if (ftStatus == FT_OK){printf(", 9");}
  else
  {
    printf("fail\n");
    return false;
  }
  Sleep(20); // According to LabView example ....

  ftStatus = FT_SetDtr(ftHandle);
  if (ftStatus == FT_OK){printf(", 10");}
  else{printf("fail\n");}
  ftStatus = FT_SetRts(ftHandle);
  if (ftStatus == FT_OK){printf(", 11");}
  else{printf("fail\n");}
  ftStatus = FT_Purge(ftHandle, FT_PURGE_RX | FT_PURGE_TX);
  if (ftStatus == FT_OK){printf(", 12 \n");}
  else{printf("fail\n");}
  return true;
}
```

A.1.4 Function: readCardToFile

```
void readCardToFile(int length, int number)
{
  DWORD BytesReceived;
    DWORD RxBytes;
  DWORD syncTime;
  DWORD TotalReadData = 0;
  DWORD T1;
  DWORD Timeout;

  string filename="datafil",ext=".txt";
  char buf[100];

  filename.append(itoa(number, buf, 10));
  filename+=ext;

  bool bSyncFound;
  int i,j,tempclass,oldclass=-1;

  ofstream outfile;
  outfile.open(filename);

  ifstream infile;
  infile.open("classes.txt");

  bSyncFound = false;
    syncTime = timeGetTime();
  Timeout = syncTime;

  do
  {
    ftStatus = FT_Read(ftHandle,byteData,1,&BytesReceived);
    TotalReadData += BytesReceived;
    if(BytesReceived == 1 && byteData[0] == 0)
    {
      ftStatus = FT_Read(ftHandle,byteData,35,&BytesReceived);
      TotalReadData += BytesReceived;
      T1 = syncTime;
      syncTime = timeGetTime();
      if(BytesReceived == 35)
      {
        ftStatus = FT_Read(ftHandle,byteData,36,&BytesReceived);
        TotalReadData += BytesReceived;
        if(BytesReceived == 36)
        {
          ReceiveCounter = byteData[1] + 1;
          if(ReceiveCounter == 256)
            ReceiveCounter = 1;
          bSyncFound = true;
          printf("Score!\n");
        }
        else
        bSyncFound = false;
      }
      else
      bSyncFound = false;
    }
    else
    bSyncFound = false;
  }while((bSyncFound == false || syncTime-T1 > AD_MCC_ADJ_MARGIN_MS) && (syncTime-Timeout < AD_MCC_START_TIMEOUT_MS));
  unsigned int in;

  for(i = 0; i<length; i++)
  {
    ftStatus = FT_GetQueueStatus(ftHandle, &RxBytes);

    else{printf("fail and ");}
        ftStatus = FT_Read(ftHandle,byteData,36,&BytesReceived);
    if (ftStatus == FT_OK)
    {
      outfile << (unsigned int) byteData[0] << " " << (unsigned int) byteData[1] << " ";
      TotalReadData += BytesReceived;
      ReceiveCounter = byteData[1] + 1;
      if((unsigned int) byteData[0]!=0)
        cout<<"Sync Error"<<endl;
      for (j=0;j<16;j++)
      {
        in =  (unsigned int) byteData[2*j+2] * 256 +  (unsigned int) byteData[2*j+3];



        if (in<32767)
          in = in + (unsigned int) 32768;
        else
          in = in - (unsigned int) 32769;

        outfile << in << " ";
      }
      infile >> tempclass;
      if (oldclass!=tempclass)
      {
        cout << tempclass << endl;
        oldclass=tempclass;
      }
      outfile << tempclass;
      outfile << endl;

    }
    else{printf("fail\n");}
  }
  cout << "And done!\n";
  outfile.close();
}
```

A.1.5 Function: menu

```
int menu()
{
  system("cls");
  int val;
  cout << "=======================\n";
  cout << "   1 to read           \n";
  cout << "   2 to train ANN      \n";
  cout << "   3 to test live      \n";
  cout << "   4 to set parameters  \n";
  cout << "   0 to exit           \n";
  cout << "=======================\n";
  cin >> val;
  return val;
}
```

A.1.6 Function: trainFromFile2

```
void trainFromFile2(int length, int overlap, int segment, int number, int typ)
{
  ifstream infile,classes;
  ofstream features;
  int sync[5000],counter[5000],tracker=0,classtest,classsegment;
  string filename="datafil",ext=".txt",featnamn="features";
  char buf[100];
  float c[NBRCHANNELS][5000],f[NBRCHANNELS][5000],feat[NR_FEAT],classarray[5000];

  switch(typ)
  {
    case 0:
      featnamn+="_val.txt";
      break;
    case 1:
      featnamn+="_train.txt";
      break;
    default:
      exit(1);
  }


  filename.append(itoa(number, buf, 10));
  filename+=ext;

  features.open(featnamn);
  infile.open(filename);

  //----------------------------------

  for(int i=0;i<segment;i++)
  {
    infile>>sync[i]>>counter[i];
    if(sync[i]!=0)
    {
      cout<<"Error when reading from file"<<endl;
      getchar();
      getchar();
    }
    for(int j=0;j<NBRCHANNELS;j++)
    {
      infile>>c[j][i];
    }
    infile>>classarray[i];
  }

  classtest=classarray[0];
  classsegment=classarray[0];

  for(int i=0;i<segment;i++)
  {
    if(classtest!=classarray[i])
    {
      classsegment=-1;
      break;
    }
  }
  for (int i=0;i<NBRCHANNELS;i++)
    filter3(&c[i][0],&f[i][0],segment,i,true);
  if(classsegment!=-1)
```

```
{
  for (int i=0;i<NBRCHANNELS;i++)
  {
    getFeatures(&f[i][0],segment,&feat[0]);

    features<<i<<" ";

    for(int j=0;j<NR_FEAT;j++)
    {
      features<<feat[j]<<" ";
    }
    features<<classsegment<<endl;
  }
}


tracker+=segment;

//---------------------------------

while(tracker+segment-overlap<=length)
{
  for(int i=0;i<overlap;i++)
  {
    sync[i]=sync[segment-overlap+i];
    counter[i]=counter[segment-overlap+i];
    for(int j=0;j<NBRCHANNELS;j++)
    {
      c[j][i]=c[j][segment-overlap+i];
      f[j][i]=f[j][segment-overlap+i];
    }

    classarray[i]=classarray[segment-overlap+i];
  }
  for(int i=overlap;i<segment;i++)
  {
    infile>>sync[i]>>counter[i];
    if(sync[i]!=0)
    {
      cout<<"Error when reading from file"<<endl;
      getchar();
      getchar();
      return;
    }
    for(int j=0;j<NBRCHANNELS;j++)
    {
      infile>>c[j][i];
    }

    infile>>classarray[i];

  }


  classtest=classarray[0];
  classsegment=classarray[0];
  for(int i=0;i<segment;i++)
  {
    if(classtest!=classarray[i])
    {
      classsegment=-1;
      break;
      }
    }
    for (int i=0;i<NBRCHANNELS;i++)
      filter3(&c[i][overlap],&f[i][overlap],segment-overlap,i,false);

    if(classsegment!=-1)
    {
      for (int i=0;i<NBRCHANNELS;i++)
      {

        getFeatures(&f[i][0],segment,&feat[0]);

        features<<i<<" ";

        for(int j=0;j<NR_FEAT;j++)
        {
          features<<feat[j]<<" ";
        }
        features<<classsegment<<endl;
      }
    }
    tracker=tracker+segment-overlap;
  }
  infile.close();
  features<<-1;
  features.close();

}
```

A.1.7 Function: getFeatures

```
void getFeatures(float *f,int segment, float *feat)
{
  //MAV


  float mav=0;

  for(int i=0;i<segment;i++)
  {
    mav+=abs(f[i]);
  }
  feat[0]=mav/segment;


  //RMS

  float rms=0;

  for(int i=0;i<segment;i++)
  {
    rms+=f[i]*f[i];
  }
  feat[1]=sqrt(rms/segment);


  //WAMP

  float wamp=0,th=500;

  for (int i=0;i<segment-1;i++)
  {
    if(abs(f[i]-f[i+1])>th)
      wamp++;
  }
  feat[2]=wamp;

  //ZC
  float zc=0;
  th=300;

  for (int i=0;i<segment-1;i++)
  {
    if( ( (f[i]<0&&f[i+i]>0) || (f[i]>0&&f[i+i]<0) ) && (abs(f[i]-f[i+1])>th) )
      zc++;
  }
  feat[3]=zc;


  //WL

  float wl=0;

  for(int i=0;i<segment-1;i++)
  {
    wl+=abs(f[i]-f[i+1]);
  }
  feat[4]=wl/segment;

  //SSC

  float ssc=0;

  for (int i=1;i<segment-1;i++)
  {
    if ((f[i-1]-f[i])*(f[i]-f[i+1])<th)
      ssc++;
  }
  feat[5]=ssc;

  //Var

  float var=0,mean=0;

  for (int i=0;i<segment;i++)
    mean+=f[i];
  mean=mean/segment;



  for (int i=0;i<segment;i++)
    var+=(f[i]-mean)*(f[i]-mean);
  feat[6]=var/segment;

}
```

A.1.8 Function: filter3

```
void filter3(float* c, float* f, int segment,int channel, bool firstrun)
{
  float m[5000],b[3]={0.5893,0,-0.5893},a[3]={1,-0.7745,-0.1786},n[5000],o[5000],bb[3]={0.9807,-1.9242,0.9807},aa[3]={1,

  static float xn[NBRCHANNELS],xn_1[NBRCHANNELS],xn_2[NBRCHANNELS],yn[NBRCHANNELS],yn_1[NBRCHANNELS],yn_2[NBRCHANNELS],x
```

```
string filename="filtrerat",ext=".txt";
char buf[100];

filename.append(itoa(channel, buf, 10));
filename+=ext;

ofstream utfil;
if (firstrun)
{
  xn[channel]=0;
  xn_1[channel]=0;
  xn_2[channel]=0;
  yn[channel]=0;
  yn_1[channel]=0;
  yn_2[channel]=0;
  xxn[channel]=0;
  xxn_1[channel]=0;
  xxn_2[channel]=0;
  yyn[channel]=0;
  yyn_1[channel]=0;
  yyn_2[channel]=0;
  xxxn[channel]=0;
  xxxn_1[channel]=0;
  xxxn_2[channel]=0;
  yyyn[channel]=0;
  yyyn_1[channel]=0;
  yyyn_2[channel]=0;

  utfil.open(filename);
  utfil.close();

}
utfil.open(filename, ios::app);






for (int i=0;i<segment;i++)
{

  m[i]=c[i]-32768;
}



for (int i=0;i<segment;i++)
{
  xn[channel] = m[i];

  yn[channel] =  b[0]* xn[channel] + b[1] * xn_1[channel] + b[2] * xn_2[channel] - a[1] * yn_1[channel] - a[2] * yn_2[

  xn_2[channel] = xn_1[channel];
  xn_1[channel] = xn[channel];
  yn_2[channel] = yn_1[channel];
  yn_1[channel] = yn[channel];

  n[i] = a[0]*yn[channel];
}

for (int i=0;i<segment;i++)
```

```
  {
    xxn[channel] = n[i];

    yyn[channel] =  bb[0]* xxn[channel] + bb[1] * xxn_1[channel] + bb[2] * xxn_2[channel] - aa[1] * yyn_1[channel] - aa[

    xxn_2[channel] = xxn_1[channel];
    xxn_1[channel] = xxn[channel];
    yyn_2[channel] = yyn_1[channel];
    yyn_1[channel] = yyn[channel];

    o[i] = aa[0]*yyn[channel];
  }

  for (int i=0;i<segment;i++)
  {
    xxxn[channel] = o[i];

    yyyn[channel] =  bb[0]* xxxn[channel] + bb[1] * xxxn_1[channel] + bb[2] * xxxn_2[channel] - aa[1] * yyyn_1[channel]

    xxxn_2[channel] = xxxn_1[channel];
    xxxn_1[channel] = xxxn[channel];
    yyyn_2[channel] = yyyn_1[channel];
    yyyn_1[channel] = yyyn[channel];

    f[i] = aa[0]*yyyn[channel];
    utfil << f[i] << " ";
  }
  utfil.close();

}
```

A.1.9 Function: ANN2

```
float ANN2(int target, float prev_error)
{


  float layer1[NRF+1],wl12[NRF+1][NRF],pd_wl12[NRF+1][NRF],layer2[NRF],pd_wl23[NRF],wl23[NRF],layer3[MXPT],val_layer3[MX
  bool done=false;
  char buf[100];
  float absfel=0,val_absfel=0;


  string filename3="val_results",ext=".txt";
  filename3.append(itoa(target, buf, 10));
  filename3+=ext;

  string filename4="val_errors";
  filename4.append(itoa(target, buf, 10));
  filename4+=ext;

  string filename="results";
  filename.append(itoa(target, buf, 10));
  filename+=ext;
  string filename2="errors";
  filename2.append(itoa(target, buf, 10));
  filename2+=ext;

  string slayer1="layer1net";
  slayer1.append(itoa(target, buf, 10));
  slayer1+=ext;
```

```
string slayer2="layer2net";
slayer2.append(itoa(target, buf, 10));
slayer2+=ext;

ofstream results,errorfil,val_results,val_errorfil,flayer1,flayer2;


int kontroll,klass,villha[MXPT];

ifstream lasin;

lasin.open("features_train.txt");


for (int i=0;i<NRF+1;i++)
{
  for (int j=0;j<NRF;j++)
  {
    wl12[i][j]=((rand()%10000)/50000.0)-0.1;
    pd_wl12[i][j]=0;
  }
}
for (int i=0;i<NRF;i++)
{
  layer2[i]=0;
  wl23[i]=((rand()%10000)/5000.0)-1;
  pd_wl23[i]=0;
}
layer1[NRF]=-1;


int pat=-1,val_pat=-1,val_villha[MXPT];
float val_data[MXPT][NRF];

fstream valin;
valin.open("features_val.txt");


while(!done||pat<MXPT)
{
  pat++;

  for(int i=0;i<NRC;i++)
  {

    lasin>>kontroll;
    if(kontroll==-1)
      break;
    if (kontroll!=i)
    {
      cout<<"Out of sync! ANN reading "<<kontroll<<" "<<i<<endl;
      done=true;
      break;
    }

    for (int j=0;j<NRFEAT;j++)
    {
      lasin>>data[pat][i*7+j];

    }
    lasin>>klass;
```

```
      if(klass==target)
        villha[pat]=1;
      else
        villha[pat]=-1;
    }
    if(kontroll==-1)
      break;


  }
  while(!done||val_pat<MXPT)
  {
    val_pat++;

    for(int i=0;i<NRC;i++)
    {

      valin>>kontroll;
      if(kontroll==-1)
        break;
      if (kontroll!=i)
      {
        cout<<"Out of sync! ANN reading "<<kontroll<<" "<<i<<endl;
        done=true;
        break;
      }

      for (int j=0;j<NRFEAT;j++)
      {
        valin>>val_data[val_pat][i*NRFEAT+j];

      }

      valin>>klass;

      if(klass==target)
        val_villha[val_pat]=1;
      else
        val_villha[val_pat]=-1;
    }
    if(kontroll==-1)
      break;


  }
  int lista[MXPT],maxp=pat,p,slump;
  float error3,error2[112],error1[113],errortot=0,wdecay=0.0001,val_errortot;



  for(int rep=0;rep<RPS;rep++)
  {

    for(int i=0;i<pat;i++)
    {
      lista[i]=i;
    }
    maxp=pat;
    errortot=0;
    for(int i=0;i<pat;i++)
    {
```

```
slump=rand()%maxp;

p=lista[slump];

lista[slump]=lista[maxp-1];
maxp--;


for(int j=0;j<NRF;j++)
{
  layer1[j]=data[p][j];
}
layer1[112]=-1;
for(int j=0;j<NRF;j++)
{
  layer2[j]=0;
  for(int k=0;k<NRF+1;k++)
  {
    layer2[j]+=wl12[k][j]*layer1[k];
  }
  layer2[j]=tanh(layer2[j]);

}
layer3[p]=0;
for(int j=0;j<NRF;j++)
{
  layer3[p]+=layer2[j]*wl23[j];
}
layer3[p]=tanh(layer3[p]);


errortot+=abs(layer3[p]-villha[p]);

//-----------------------------------

error3=(1-layer3[p]*layer3[p])*(villha[p]-layer3[p]);

for(int j=0;j<NRF;j++)
{
  error2[j]=(1-layer2[j]*layer2[j])*error3*wl23[j];
}

for(int j=0;j<NRF+1;j++)
{
  error1[j]=0;
  for (int k=0;k<NRF;k++)
  {
    error1[j]+=error2[k]*wl12[j][k]*(1-layer1[j]*layer1[j]);
  }
}
for(int j=0;j<NRF;j++)
{
  dw=alpha*error3*layer2[j];
  wl23[j]+=dw;//+momentum*wl23[j];
  //wl23[j]=(1-wdecay)*wl23[j];
}
for(int j=0;j<NRF+1;j++)
{
  for(int k=0;k<NRF;k++)
  {
    dw=alpha*error2[k]*layer1[j];
    wl12[j][k]+=dw;//+momentum*wl12[j][k];//pd_
```

```
        //wl12[j][k]=(1-wdecay)*wl12[j][k];
      }
    }
  }

  val_errortot=0;
  for(int i=0;i<val_pat;i++)
  {
    for(int j=0;j<NRF;j++)
    {
      layer1[j]=val_data[i][j];
    }
    layer1[112]=-1;
    for(int j=0;j<NRF;j++)
    {
      layer2[j]=0;
      for(int k=0;k<NRF+1;k++)
      {
        layer2[j]+=wl12[k][j]*layer1[k];
      }
      layer2[j]=tanh(layer2[j]);

    }
    val_layer3[i]=0;
    for(int j=0;j<NRF;j++)
    {
      val_layer3[i]+=layer2[j]*wl23[j];
    }
    val_layer3[i]=tanh(val_layer3[i]);

    val_errortot+=abs(val_layer3[i]-val_villha[i]);
  }

  val_absfel=0;
  absfel=0;
  for(int i=0;i<val_pat;i++)
  {
    if(abs(val_layer3[i]-val_villha[i])>0.5)
      val_absfel+=1;
  }

  for(int i=0;i<pat;i++)
  {
    if(abs(layer3[i]-villha[i])>0.5)
      absfel+=1;
  }
  if ((prev_error>val_absfel/val_pat*100)&&(int_error>val_absfel/val_pat*100))
  {

    prev_error=val_absfel/val_pat*100;
    int_error=prev_error;
    cout<<"New best = "<<prev_error<<endl;
    results.open(filename);
    errorfil.open(filename2);

    val_results.open(filename3);
    val_errorfil.open(filename4);

    flayer1.open(slayer1);
    flayer2.open(slayer2);
    for(int i=0;i<val_pat;i++)
    {
        val_results<<val_layer3[i]<<" ";
        val_errorfil<<val_layer3[i]-val_villha[i]<<" ";

      }
      for(int i=0;i<pat;i++)
      {
        results<<layer3[i]<<" ";
        errorfil<<layer3[i]-villha[i]<<" ";
      }
      for(int i=0;i<NRF;i++)
      {
        for(int j=0;j<NRF+1;j++)
        {
          flayer1<<wl12[j][i]<<" ";
        }
        flayer2<<wl23[i]<<" ";
        flayer1<<endl;
      }
      results.close();
      errorfil.close();

      val_results.close();
      val_errorfil.close();

      flayer1.close();
      flayer2.close();
    }




    if (val_errortot<0.5)
    {
      cout<<endl<<"Early stop reached after "<<rep<<" epocs"<<endl;
      break;

    }

  }

  if(prev_error<1000)
    cout<<"Best so far = "<<prev_error<<" Fel nu = "<<val_absfel/val_pat*100<<endl;



  cout<<"Training error = "<<absfel<<" Validation error = "<<val_absfel<<endl;


  return prev_error;
}
```

A.1.10 Function: live

```
void live(float nr_feat,int segment, int overlap)
{
  float xnl[NBRCHANNELS],xnl_1[NBRCHANNELS],xnl_2[NBRCHANNELS],ynl[NBRCHANNELS],ynl_1[NBRCHANNELS],ynl_2[NBRCHANNELS],xx

  float layer1[113],layer2[NR_ANN][112],layer3[NR_ANN],wl12[NR_ANN][113][112],wl23[NR_ANN][112];

  int major[NR_ANN][10];
  int rp,trashy;
```

```
openComPort(1);
cout<<"Starting.."<<endl;
setCommsTimeOut();
cout<<"Any key to start!"<<endl;
cin>>trashy;
for(int i=0;i<6;i++)
{
  Data[i*3]=255;
  Data[i*3+1]=i;

}
Data[2]=160;
Data[5]=140;
Data[8]=140;
Data[11]=140;
Data[14]=140;
Data[17]=25;
WriteFile(hComPort, &Data[0], 18, &NWrite, NULL);

DWORD BytesReceived;
  DWORD RxBytes;
DWORD syncTime;
DWORD TotalReadData = 0;
DWORD T1;
DWORD Timeout;


syncTime = timeGetTime();
Timeout = syncTime;

for(int i=0;i<NR_ANN;i++)
  for(int j=0;j<10;j++)
    major[i][j]=0;

ifstream in_wl12,in_wl23;
char buf[100];
string slayer1;
string slayer2;

bool bSyncFound;



cout<<"loading ANN"<<endl;

for(int featnr=0;featnr<NR_ANN;featnr++)
{
  slayer1="layer1net";
  slayer1.append(itoa(featnr+1, buf, 10));
  slayer1+=".txt";

  slayer2="layer2net";
  slayer2.append(itoa(featnr+1, buf, 10));
  slayer2+=".txt";

  in_wl12.open(slayer1);
  in_wl23.open(slayer2);

  for(int i=0;i<112;i++)
  {
    for(int j=0;j<113;j++)
    {
      in_wl12>>wl12[featnr][j][i];
    }
    in_wl23>>wl23[featnr][i];
  }
  in_wl12.close();
  in_wl23.close();

}


cout<<"Going live!"<<endl;
float* data[NRC];

for (int i=0;i<NRC;i++)
  data[i] = new float [25600];

for (int i=0;i<NRC;i++)
{
  xnl[i]=0;
  xnl_1[i]=0;
  xnl_2[i]=0;
  ynl[i]=0;
  ynl_1[i]=0;
  ynl_2[i]=0;
  xxnl[i]=0;
  xxnl_1[i]=0;
  xxnl_2[i]=0;
  yynl[i]=0;
  yynl_1[i]=0;
  yynl_2[i]=0;
  xxxnl[i]=0;
  xxxnl_1[i]=0;
  xxxnl_2[i]=0;
  yyynl[i]=0;
  yyynl_1[i]=0;
  yyynl_2[i]=0;
}


unsigned int in;
float feat[NR_FEAT],b[3]={0.5893,0,-0.5893},a[3]={1,-0.7745,-0.1786},bb[3]={0.9807,-1.9242,0.9807},aa[3]={1,-1.9242,0.


int inint;
do
{
  ftStatus = FT_Read(ftHandle,byteData,1,&BytesReceived);
  TotalReadData += BytesReceived;
  if(BytesReceived == 1 && byteData[0] == 0)
  {
    ftStatus = FT_Read(ftHandle,byteData,35,&BytesReceived);
    TotalReadData += BytesReceived;
    T1 = syncTime;
    syncTime = timeGetTime();
    if(BytesReceived == 35)
    {
      ftStatus = FT_Read(ftHandle,byteData,36,&BytesReceived);
      TotalReadData += BytesReceived;
      if(BytesReceived == 36)
      {
        ReceiveCounter = byteData[1] + 1;
        if(ReceiveCounter == 256)
          ReceiveCounter = 1;
        bSyncFound = true;
        printf("Sync found!\n");
      }
      else
      bSyncFound = false;
    }
    else
    bSyncFound = false;
  }
  else
  bSyncFound = false;
}while((bSyncFound == false || syncTime-T1 > AD_MCC_ADJ_MARGIN_MS) && (syncTime-Timeout < AD_MCC_START_TIMEOUT_MS));
cout<<"Going live!"<<endl;

for(int i = 0; i<segment; i++)
{
  ftStatus = FT_GetQueueStatus(ftHandle, &RxBytes);

  else{printf("fail and ");}
  ftStatus = FT_Read(ftHandle,byteData,36,&BytesReceived);
  if (ftStatus == FT_OK)
  {


    if((unsigned int) byteData[0]!=0)
      cout<<"--------------------------------------------------------------------------------"<<endl;
    for (int j=0;j<16;j++)
    {

      in =  (unsigned int) byteData[2*j+2] * 256 +  (unsigned int) byteData[2*j+3];
      inint=(int)in;

      inint-=32768;


      //-----filter-------------
      xnl[j] = inint;

      ynl[j] =  b[0]* xnl[j] + b[1] * xnl_1[j] + b[2] * xnl_2[j] - a[1] * ynl_1[j] - a[2] * ynl_2[j];

      xnl_2[j] = xnl_1[j];
      xnl_1[j] = xnl[j];
      ynl_2[j] = ynl_1[j];
      ynl_1[j] = ynl[j];

      xxnl[j] = a[0]*ynl[j];



      yynl[j] =  bb[0]* xxnl[j] + bb[1] * xxnl_1[j] + bb[2] * xxnl_2[j] - aa[1] * yynl_1[j] - aa[2] * yynl_2[j];

      xxnl_2[j] = xxnl_1[j];
      xxnl_1[j] = xxnl[j];
      yynl_2[j] = yynl_1[j];
      yynl_1[j] = yynl[j];

      xxxnl[j] = aa[0]*yynl[j];


      yyynl[j] =  bb[0]* xxxnl[j] + bb[1] * xxxnl_1[j] + bb[2] * xxxnl_2[j] - aa[1] * yyynl_1[j] - aa[2] * yyynl_2[j];
      xxxnl_2[j] = xxxnl_1[j];
      xxxnl_1[j] = xxxnl[j];
      yyynl_2[j] = yyynl_1[j];
      yyynl_1[j] = yyynl[j];

      data[j][i] = aa[0]*yyynl[j];


    }


  }
  else{printf("fail\n");}


}
for (int j=0;j<NBRCHANNELS;j++)
{
  getFeatures(&data[j][0],segment,&feat[0]);



  for(int k=0;k<NR_FEAT;k++)
  {

    layer1[j*NR_FEAT+k]=feat[k];

  }

}

int m_counter=0;

for(int ep=0;ep<1000;ep++)
{
  cout<<ep;
  for(int i=0;i<overlap;i++)
  {
    for(int j=0;j<NRC;j++)
    {
      data[j][i]=data[j][segment-overlap+i];
    }
  }
  for(int i=overlap;i<segment;i++)
  {
    ftStatus = FT_GetQueueStatus(ftHandle, &RxBytes);

    else{printf("fail and ");}
    ftStatus = FT_Read(ftHandle,byteData,36,&BytesReceived);
    if (ftStatus == FT_OK)
    {

      for(int j=0;j<NRC;j++)
      {




        in =  (unsigned int) byteData[2*j+2] * 256 +  (unsigned int) byteData[2*j+3];
        inint=(int)in;
        inint-=32768;


        //-----filter-------------
        xnl[j] = inint;

        ynl[j] =  b[0]* xnl[j] + b[1] * xnl_1[j] + b[2] * xnl_2[j] - a[1] * ynl_1[j] - a[2] * ynl_2[j];

        xnl_2[j] = xnl_1[j];
        xnl_1[j] = xnl[j];
        ynl_2[j] = ynl_1[j];
        ynl_1[j] = ynl[j];

        xxnl[j] = a[0]*ynl[j];


        yynl[j] =  bb[0]* xxnl[j] + bb[1] * xxnl_1[j] + bb[2] * xxnl_2[j] - aa[1] * yynl_1[j] - aa[2] * yynl_2[j];

        xxnl_2[j] = xxnl_1[j];
        xxnl_1[j] = xxnl[j];
        yynl_2[j] = yynl_1[j];
        yynl_1[j] = yynl[j];

        xxxnl[j] = aa[0]*yynl[j];

        yyynl[j] =  bb[0]* xxxnl[j] + bb[1] * xxxnl_1[j] + bb[2] * xxxnl_2[j] - aa[1] * yyynl_1[j] - aa[2] * yyynl_2[j

        xxxnl_2[j] = xxxnl_1[j];
        xxxnl_1[j] = xxxnl[j];
        yyynl_2[j] = yyynl_1[j];
        yyynl_1[j] = yyynl[j];

        data[j][i] = aa[0]*yyynl[j];


      }
    }
    else{printf("fail\n");}

  }
  for (int j=0;j<NBRCHANNELS;j++)
  {
    getFeatures(&data[j][0],segment,&feat[0]);



    for(int k=0;k<NR_FEAT;k++)
    {
      layer1[j*NR_FEAT+k]=feat[k];

    }

  }


  for(int p=0;p<NR_ANN;p++)
  {
    layer1[112]=-1;
    for(int i=0;i<112;i++)
    {
      layer2[p][i]=0;
      for(int j=0;j<113;j++)
      {
        layer2[p][i]+=layer1[j]*wl12[p][j][i];

      }
      layer2[p][i]=tanh(layer2[p][i]);

    }
    layer3[p]=0;
    for(int i=0;i<112;i++)
    {
      layer3[p]+=layer2[p][i]*wl23[p][i];
    }
    layer3[p]=tanh(layer3[p]);

    if (layer3[p]>0)
      major[p][m_counter]=1;
    else
      major[p][m_counter]=0;
    rp=3*p+5;
    if((major[p][0]+major[p][1]+major[p][2]+major[p][3]+major[p][4]+major[p][5]+major[p][6]+major[p][7]+major[p][8]+ma
    {

      cout<<"++++++++++++++";
      //-----Fyra fingrar-----
      Data[p*3+5]=40;
      if(p==2)
        Data[11]=240;



      //-----Kombo-----
      /*if(p==0)
        Data[5]=40;
      if(p==1)
      {
        Data[8]=40;
        Data[11]=240;
        Data[14]=40;
      }
      if(p==2)
        Data[2]=110;
      if(p==3)
        Data[17]=60;*/
      //----------


    }
    else
    {

      cout<<". . . . . . . ";

      //-----Fyra fingrar-----
      Data[p*3+5]=150;
      if(p==2)
        Data[11]=140;
      if(p==4)
        Data[14]=160;

      //-----Kombo-----
      /*if(p==0)
        Data[5]=140;
```

```
        if(p==1)
        {
          Data[8]=140;
          Data[11]=130;
          Data[14]=140;
        }
        if(p==2)
          Data[2]=130;
        if(p==3)
          Data[17]=25;*/
        //----------
      }


    }
    WriteFile(hComPort, &Data[0], 18, &NWrite, NULL);
    m_counter=(m_counter+1)%8;

    cout<<endl;
  }
  closeComPort();
  for (int i=0; i<NRC; i++) delete [] data[i];

}
```

A.1.11 Function: openComPort

```
void openComPort(int port) {

    DWORD dw;
    dw = GetLastError();
    printf("LastError : %d\n", dw);


    DCB dcb;
    dw = GetLastError();
    printf("LastError : %d\n", dw);

    hComPort = CreateFile(_T("COM6"),
           GENERIC_WRITE,
           0,                    // exklusiv åtkomst
           NULL,                 // no security attributes
           OPEN_EXISTING,
           NULL,  //FILE_FLAG_OVERLAPPED, // overlapped I/O
           NULL);

    dw = GetLastError();
    printf("LastError after CreateFile : %d\n", dw);

    if (hComPort == INVALID_HANDLE_VALUE)
    printf("Couldnt open comport\n");

    dcb.DCBlength = sizeof(DCB);
    GetCommState(hComPort, &dcb);

    // communication properties
    dcb.BaudRate    = CBR_9600;// m_nBaudRate
    dcb.ByteSize    = 8;
    dcb.Parity     = NOPARITY;
    dcb.StopBits    = ONESTOPBIT;
    // setup hardware flow control
    dcb.fOutxDsrFlow  = FALSE;
    dcb.fDtrControl = DTR_CONTROL_DISABLE;
    dcb.fOutxCtsFlow  = FALSE;
    dcb.fRtsControl = RTS_CONTROL_DISABLE;
    // setup software flow control
    dcb.fInX      = FALSE;
    dcb.fOutX     = FALSE;
    dcb.XonChar   = 0;
    dcb.XoffChar    = 0;
    dcb.XonLim    = 0; //Cannot be bigger than 4096
    dcb.XoffLim   = 0; //Cannot be bigger than 4096
    // other various settings
    dcb.fParity   = TRUE;
    dcb.fBinary   = TRUE; // The Win32 API does not support nonbinary mode
    // transfers, so this member should be TRUE.
    // Trying to use FALSE will not work.
    dcb.fTXContinueOnXoff = FALSE; // CA addition
    dcb.fAbortOnError = FALSE;  // CA addition


    SetCommState(hComPort,&dcb); // todo: check error
    dw = GetLastError();
    printf("LastError after SetCommState : %d\n", dw);
    PurgeComm(hComPort,PURGE_TXABORT|PURGE_RXABORT|PURGE_TXCLEAR|PURGE_RXCLEAR); // CA addition
    dw = GetLastError();
    printf("LastError after Purge: %d\n", dw);
}
```

A.1.12 Function: setCommsTimeOut

```
void setCommsTimeOut()
{
  DWORD dw;
  COMMTIMEOUTS CommTimeouts;
  GetCommTimeouts (hComPort, &CommTimeouts);

  // Change the COMMTIMEOUTS structure settings.
  CommTimeouts.ReadIntervalTimeout = MAXDWORD; //MAXDWORD
  CommTimeouts.ReadTotalTimeoutMultiplier = MAXDWORD;
  CommTimeouts.ReadTotalTimeoutConstant = 0;
  CommTimeouts.WriteTotalTimeoutMultiplier = 10;
  CommTimeouts.WriteTotalTimeoutConstant = 1000;

  // Set the timeout parameters for all read and write operations
  // on the port.
  if (!SetCommTimeouts (hComPort, &CommTimeouts))
  {
    // Could not set the timeout parameters.
    printf("Unable to set timeout parameters...\n");
    dw = GetLastError ();
    printf("LastError : %d\n", dw);
  }
}
```

A.1.13 Function: closeComPort

```
void closeComPort()
{
  if(hComPort) {
    CloseHandle(hComPort);
    hComPort=0;
  }
}
```

A.2 ANN tester

```
#include "stdafx.h"
#include "iostream"
#include "fstream"
#include <math.h>
#include "time.h"

using namespace std;

#define NRCH 16
#define NRFEAT 112
#define NROUT 4
#define RPS 40

float data_train[960][112],data_val[240][112];
float* goal_train= new float [960];
float* goal_val = new float [240];

void ANN(int target, int nr_neu, int rep_b);

int main(int argc, _TCHAR* argv[])
{

  srand(time(NULL));

  ifstream features_train,features_val;

  features_train.open("features_train.txt");
  features_val.open("features_val.txt");

  int syncbit=0,pat_t=0;

  while (syncbit!=-1)
  {
    for (int c=0;c<16;c++)
    {
      features_train >> syncbit;
      if (syncbit == -1)
        break;


      for (int i=0;i<7;i++)
      {
        features_train >> data_train[pat_t][syncbit*7+i];
      }
      features_train >> goal_train[pat_t];

    }
    if (syncbit == -1)
      break;
    pat_t++;
  }
  cout<<pat_t<<" patterns loaded for training"<<endl;

  syncbit=0;
  int pat_v=0;

  while (syncbit!=-1)
  {
    for (int c=0;c<16;c++)
    {
      features_val >> syncbit;
      if (syncbit == -1)
        break;


      for (int i=0;i<7;i++)
      {
        features_val >> data_val[pat_v][syncbit*7+i];
      }
      features_val >> goal_val[pat_v];

    }
    if (syncbit == -1)
      break;
    pat_v++;
  }

  cout<<pat_v<<" patterns loaded for validation"<<endl;


  //------------------------------------------------------------------------


  for(int target=4;target<5;target++)
  {

    for (int nr_neu=10;nr_neu<120;nr_neu=nr_neu+10)
    {
      for(int rep_b=1;rep_b<11;rep_b++)
      {
        cout<<"Target = "<<target<<" #Neurons = "<<nr_neu<<" #Try = "<<rep_b<<endl;

        ANN(target,nr_neu,rep_b);
      }
    }

  }
  cout<<"KLAR!"<<endl;
  getchar();
  getchar();

  return 0;
}

void ANN(int target, int nr_neu,int rep_b)
{
  char buf[100];
  string filename1="error_",ext=".txt",score="_";
  filename1.append(itoa(target, buf, 10));
  filename1+=score;
  filename1.append(itoa(nr_neu, buf, 10));
  filename1+=score;
  filename1.append(itoa(rep_b, buf, 10));
  filename1+=ext;

  ofstream utfil;

  utfil.open(filename1);

  float* layer[3];

  layer[0] = new float [113];
```

```
layer[1] = new float [nr_neu];
layer[2] = new float;

float* w23 = new float [nr_neu];
float* error2 = new float [nr_neu];

float** w12 = new float* [113];

float error3,error1[113],errortot=0,wdecay=0.0001,val_errortot,dw,alpha=0.05,abs_error,abs_valerror,live_errortot,abs_

for (int i=0;i<113;i++)
  w12[i] = new float [nr_neu];

int list[720],maxp,slump,p,vill_ha;

layer[0][112]=-1;

for (int i=0;i<113;i++)
{
  for (int j=0;j<nr_neu;j++)
    w12[i][j]=((rand()%10000)/50000.0)-0.1;
}

for (int i=0;i<nr_neu;i++)
  w23[i]=((rand()%10000)/5000.0)-1;


for(int rep=0;rep<RPS;rep++)
{
  utfil<<rep+1<<" ";
  for (int i=0;i<720;i++)
    list[i]=i;
  maxp=720;
  errortot=0;
  abs_error=0;
  for(int pat_nr=0;pat_nr<720;pat_nr++)
  {
    slump=rand()%maxp;
    p=list[slump];
    list[slump]=list[maxp-1];
    maxp--;

    for (int i=0;i<112;i++)
    {
      layer[0][i]=data_train[p][i];

    }

    if (target==goal_train[p])
      vill_ha=1;
    else
      vill_ha=-1;

    for (int i=0;i<nr_neu;i++)
    {
      layer[1][i]=0;
      for (int j=0;j<113;j++)
        layer[1][i]+=(w12[j][i]*layer[0][j]);
      layer[1][i]=tanh(layer[1][i]);

    }
    layer[2][0]=0;
    for (int i=0;i<nr_neu;i++)
    {
      layer[2][0]+=(w23[i]*layer[1][i]);
    }
    layer[2][0]=tanh(layer[2][0]);

    errortot+=abs(layer[2][0]-vill_ha);
    if(abs(layer[2][0]-vill_ha)>0.5)
      abs_error++;

    //-----------------------------------

    error3=(1-layer[2][0]*layer[2][0])*(vill_ha-layer[2][0]);

    for(int j=0;j<nr_neu;j++)
    {
      error2[j]=(1-layer[1][j]*layer[1][j])*error3*w23[j];
    }

    for(int j=0;j<NRFEAT+1;j++)
    {
      error1[j]=0;
      for (int k=0;k<nr_neu;k++)
      {
        error1[j]+=error2[k]*w12[j][k]*(1-layer[0][j]*layer[0][j]);
      }
    }
    for(int j=0;j<nr_neu;j++)
    {
      dw=alpha*error3*layer[1][j];
      w23[j]+=dw;
      w23[j]=(1-wdecay)*w23[j];
    }
    for(int j=0;j<NRFEAT+1;j++)
    {
      for(int k=0;k<nr_neu;k++)
      {
        dw=alpha*error2[k]*layer[0][j];
        w12[j][k]+=dw;
        //w12[j][k]=(1-wdecay)*w12[1][2];
      }
    }
  }
  utfil<<errortot/14.40<<" "<<abs_error/14.4<<" ";

  val_errortot=0;
  abs_valerror=0;
  for(int i=0;i<240;i++)
  {
    if (target==goal_val[i])
      vill_ha=1;
    else
      vill_ha=-1;

    for(int j=0;j<NRFEAT;j++)
    {
      layer[0][j]=data_val[i][j];
    }
    layer[0][112]=-1;
    for(int j=0;j<nr_neu;j++)
    {
        layer[1][j]=0;
        for(int k=0;k<NRFEAT+1;k++)
        {
          layer[1][j]+=w12[k][j]*layer[0][k];
        }
        layer[1][j]=tanh(layer[1][j]);
      }
      layer[2][0]=0;
      for(int j=0;j<nr_neu;j++)
      {
        layer[2][0]+=layer[1][j]*w23[j];
      }
      layer[2][0]=tanh(layer[2][0]);
      val_errortot+=abs(layer[2][0]-vill_ha);
      if (abs(layer[2][0]-vill_ha)>0.5)
        abs_valerror++;
    }
    utfil<<val_errortot/4.80<<" "<<abs_valerror/4.8<<" ";



    live_errortot=0;
    abs_liveerror=0;
    for(int i=721;i<960;i++)
    {
      if (target==goal_train[i])
        vill_ha=1;
      else
        vill_ha=-1;

      for(int j=0;j<NRFEAT;j++)
      {
        layer[0][j]=data_train[i][j];
      }
      layer[0][112]=-1;
      for(int j=0;j<nr_neu;j++)
      {
        layer[1][j]=0;
        for(int k=0;k<NRFEAT+1;k++)
        {
          layer[1][j]+=w12[k][j]*layer[0][k];
        }
        layer[1][j]=tanh(layer[1][j]);
      }
      layer[2][0]=0;
      for(int j=0;j<nr_neu;j++)
      {
        layer[2][0]+=layer[1][j]*w23[j];
      }
      layer[2][0]=tanh(layer[2][0]);

      live_errortot+=abs(layer[2][0]-vill_ha);
      if (abs(layer[2][0]-vill_ha)>0.5)
        abs_liveerror++;
    }
    utfil<<live_errortot/4.80<<" "<<abs_liveerror/4.8<<endl;


  }
}
```