LEONARDO MARIANO GOMES

HANDHELD SOUND GENERATOR

Trabalho de Conclusão de Curso apresentado à
Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase em
Eletrônica

ORIENTADOR: Evandro Luís Linhari Rodrigues

São Carlos
2013

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO, POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

Gomes, Leonardo Mariano
G633h Handheld Sound Generator / Leonardo Mariano Gomes; orientador Evandro Luís Linhari Rodrigues. São Carlos, 2013.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2013.

1. Accelerometer. 2. Electromyograph. 3. Wireless Communication. 4. Sound. 5. Embedded System. I. Título.

FOLHA DE APROVAÇÃO

Nome: Leonardo Mariano Gomes

Título: "Handheld sound generator"

Trabalho de Conclusão de Curso defendido e aprovado em 19 / 11 / 2013,

com NOTA 10,0 (dez, zero), **pela Comissão Julgadora:**

Prof. Associado Evandro Luís Linhari Rodrigues - (Orientador - SEL/EESC/USP)

Prof. Dr. Marcelo Basílio Joaquim - (SEL/EESC/USP)

Profa. Associada Liliane Ventura Schiabel - (SEL/EESC/USP)

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Homero Schiabel

This work is dedicated to my parents Roberto and Vera, and to my brother Roberto.

ACKNOWLEDGEMENT

The development of this work overcame my expectations and took lot of efforts for doing it. However, it would not have been possible without the support and help of many individuals and organizations. I would like to extend my sincere thanks to all of them.

First, I'd like to thank my parents for the attention and care during all my life, supporting my decisions and always giving me advices about all aspects of this complicated life. Special thanks to my father, who introduced me to the world of electronics and machines and made me to love the smell of soldering and, above all, made me love the idea of building stuff and solving problems. I'd like also to thank my brother, for all the moments, from childhood until now, for the chats and funny moments.

I'm really grateful to all my friends, especially the undergraduate ones. All of you contributed somehow for my professional and mostly personal formation. During the course, we faced different situations together, from scary exams to funny (sometimes weird) parties and pleasant talks. We've all learned together and I hope seeing all of you in the next years.

I'd like to thank the University of São Paulo and the teachers I had, the opportunities given by being on this rich in knowledge environment and the knowledge and skills I obtained here. Many thanks to CAPES and to the opportunity I've received of studying in the University of Glasgow. In Scotland I had the opportunity to grow more and more as a professional and learn things from a different perspective. Thanks also to the technicians of the University of Glasgow who helped me a lot on prototyping the equipment I've designed. Special thanks to both supervisors I had: Calum Cossar and Evandro, both contributed to the final result of my project and my personal development.

At last but very far from least, I'd like to thank my girlfriend Fernanda, for all support given, attention and dedication during the period we are together. You helped me when I was sad, when my project was not working and I was almost kicking the electronic boards but mainly when I was missing you and my heart freezing in Scotland. It's not easy dating thousands miles away but you are being strong and making me strong every day, making me wish to love you every moment of my life.

Abstract

This work was aimed at designing new electronic equipment which was used to generate sound according to the movements of the user. Two different sensors were used to measure different signals: an accelerometer detected the angular position of the user's arm and Electromyography (EMG) electrodes correctly positioned detected the muscular activity in the same arm where the accelerometer was placed. The information obtained by the sensors was processed using an embedded system and electronic circuits, and a wireless communication system was responsible for transferring this information to a remote computer. The software present in this computer acquired the information transmitted from the equipment and converted it into musical sounds. The characteristics of the generated sound could change dynamically, following the movements of the user. Simulations and experimental procedures were used to evaluate the results of the developed electronic equipment and software. After prototyping the equipment, the EMG signals could be amplified and measured using a portable oscilloscope, the angles of the user's arm were measured and compared with the calculated expected angles. The software on computer was able to connect to the equipment, processing the information received and generating musical sounds. The work accomplished its goals because a fully operational equipment was designed fulfilling the requirements of generating sound, as the parameters of the generated sound changed in function of the motion of the person using the equipment.

Keywords: Accelerometer, Electromyograph (EMG), wireless communication, sound, embedded system.

Resumo

Este trabalho teve como objetivo o projeto de um novo equipamento eletrônico o qual foi usado para gerar sons de acordo com os movimentos do usuário. Dois sensores diferentes foram usados para medir diferentes sinais: um acelerômetro mediu a posição angular do braço do usuário e eletrodos de Eletromiografia (EMG) corretamente posicionados detectaram a atividade muscular, no mesmo braço onde o acelerômetro estava localizado. A informação obtida pelos sensores foi processada usando um sistema embarcado e circuitos eletrônicos, e comunicação sem-fio é responsável por transmitir a informação para um computador remoto. O software presente neste computador recebeu a informação transmitida do equipamento e converteu-a em sons musicais. As características do som gerado pode variar dinamicamente, de acordo com os movimentos do usuário. Simulações e procedimentos experimentais foram utilizados para validar os resultados do equipamento eletrônico e software desenvolvidos. Após a prototipagem do equipamento, os sinais de EMG puderam ser amplificados, e medidos através de um osciloscópio portátil, as posições angulares do braço do utilizador foram medidos e comparados com os angulos calculados esperados. O software do computador foi capaz de conectar-se com o equipamento, processar a informação recebida e gerar sons musicais. O trabalho cumpriu seus objetivos, pois um equipamento totalmente operacional foi desenvolvido preenchendo os requisitos,onde os parâmetros do som gerado alteravam em função dos movimentos do indivíduo ao usar o equipamento.

Palavras-chave: Acelerômetro, Eletromiógrafo (EMG), comunicação sem-fio, sons, sistema embarcado.

Contents

List of Figures

CHAPTER 1 – INTRODUCTION

I – FOREWORD

The application of new technologies in different fields of knowledge is growing fast over the years. The technology used today for doing a task is much more complex but helpful for people in comparison with the technology used during the last decade. Smartphones are the first example that comes in mind in terms of evolution of technology, but medical equipment, video-games, electronic musical instruments and even toys are other examples in which technology is changing constantly.

Considering the usages of technology mentioned above, it can be highlighted two of them, which will be important for the ideas and concepts of the work: the medical equipment, specifically biomedical instrumentation, and electronic musical instruments. These areas will be interconnected in the project, with the medical instrumentation being used to control a musical instrument, not necessarily a well-known instrument, but a set of hardware and software which will work like it, as will be explained in the following.

The biomedical engineering involves the application of engineering techniques to improve health care and health services to enhance the quality of human life. Different fields of knowledge are covered by the biomedical engineering, for example medical devices, biomechanics, medical imaging and biological signal processing [1]. The biomedical engineering begun after the second world war, primarily studying complex biological systems and the rehabilitation of soldiers. The evolution of technology made possible the biomedical engineering to develop instruments for medical usage [2]. Many diseases can be diagnosed through the medical devices and medical monitors. A classical example is the ECG (electrocardiograph), which is the equipment responsible for monitoring the electrical activity of heart and helps detecting cardiac diseases. Another medical device which works similarly with the ECG is the EMG, or Electromyograph.

Electromyography is a technique for evaluating and recording the electrical activity produced by skeletal muscles [3]. EMG signals are used in many clinical and biomedical applications as a diagnostics tool for identifying neuromuscular diseases, for example problems with the transmission of electrical signals from brain to muscles, but also the study of human kinects (Kinesiology). The EMG signals can also be used to control prosthetic devices and help the life of persons who have limited conditions of moving or have lost their members for any reason.

As any medical instrumentation device, the EMG has the purpose of measure or determining the presence of a physical quantity, in its case the muscular activity, and assists the medical personnel to make better diagnosis and treatment. Certain characteristic features, which are common to most instrumentation systems, are also applicable to medical instrumentation systems and any medical

instrument would comprise of the following basic functional components: measurand, transducer/sensor, signal conditioner and display system [4]. This work will explain what the four basic functional components are and how they are used in this project.

The second area of interest or usage of technology, which deserves special attention in this work, is the musical instruments area. As mentioned above, the area of medical instrumentation and music generation will be connected to produce new equipment, which can be classified as a musical instrument or more specifically, an electronic musical instrument.

Music is present every day, every moment in the life of all the people. Whether in personal music players, on the gym as an ambient sound, on a jingle of an advertisement or in a concert hall, people are listening to music in different ways, different styles and being influenced by it, without even realizing it. Each person has an involvement with the music depending on their personal interests, but certainly music influences and is also influenced by society.

With advanced electronic circuits, different software resources and open-source platforms, even for electronic and software, it is becoming easier for people to transform their ideas into practical works and machines, and thereby easily integrate specific knowledge, needing to solve a problem and creativity into applications which can be used for everyone and everywhere. The importance of music can be verified on society but also on individuals by the influences and the effects caused by music on people. Music is considered a universal language because it can be understood the same way by different cultures that not necessarily spoke the same language. Music also inspires emotion and influences mood of people mostly in a healthy way. Music also contributes for education of adults but more of children because it enhances learning and makes it more enjoyable. The act of playing music sparks the imagination and makes children develop and improve skills such as motor control, hearing, sight, memory and teaches us self-discipline and time management skills [5][6].

Musical instruments are an essential part on process of music creation. They have been developed since the ancient ages and on different parts of the world. From 18th- century, an emergency of electric musical instruments and after that electronic started to being explored by numerous scientists and musicians. With the fast development of technology through years, those instruments became more improved and popular, making possible the creation of new music genres.

A musical instrument can be defined as a device created or adapted to make musical sounds. The characteristics and physical properties of an instrument are used to classify it as a musical instrument and in its sub-genres. Musical instruments are able to control characteristics of the generated sound, for example timbre, volume, sound duration and intensity and the way sound is produced can be used to categorize instruments by its user interfaces. Some examples of user interfaces are keys on a keyboard, reeds on a saxophone and strings on a guitar. Musical instruments can also be classified as acoustic, when the sound is produced by physical properties, for example the shape of the body, or as electronic, when sound is produced by electronic circuits and a loudspeaker.

An important component responsible but also a result of the fast evolution of technology is the embedded system. These systems are defined as systems which have at least one microprocessor and this microprocessor is responsible for a specific task. With the constant size reduction of the electronic components and also their manufacturing cost reduction, it became profitable to include these systems inside general equipment. Different usages of embedded system can be noticed nowadays in most of applications mentioned on the beginning of this section. Embedded systems comprises to main parts: the hardware and the software. The hardware consists of electronic components and circuits, and the software is a written code, which is recorded in the memory of the microcontroller or externally in a memory component. The software, also called firmware, is responsible for making the electronic circuits functioning and developing the specific task required by the equipment.

II – AIMS

This project aims at developing electronic equipment which will generate sound and the characteristics of sound will change dynamically following the motion of the person using the equipment. The idea is to integrate different subjects of the Electronics Engineering and develop the equipment from the simulations to the final working project and evaluate the knowledge acquired during the undergraduate course.

III – TEXT ORGANIZATION

This work will be organized into 7 chapters. Chapter 1 discuss the basic areas of knowledge whose have some relation to the project presented in this work and also exposes the aims of the project. Chapter 2 presents the theory and concepts used in this project and familiarizes the reader of this work with some terms used in the other chapters. In chapter 3, can be found information about the working principles of the equipment design and how its components are important to the system. Chapter 4 present the experimental techniques used for designing circuits and build the prototype, but also programming techniques are explained. Chapter 5 is about the results from circuit simulation and measurements and Chapter 6 is a discussion about the results obtained. Finally, chapter 7 will present conclusions about the project and a brief discussion about future works that can be developed considering the hardware and software of equipment. In the end of this work, appendixes containing electronic circuit projects and program codes, can be found.

CHAPTER 2 – THEORY & CONCEPTS

The theory involved with the project is related to digital data acquisition, sampling and quantization, which were used on data acquisition and conversion from analog sensors by the microcontroller and its peripherals. The processing of accelerometer data involves math theory, considering the conversion of voltage into angle of orientation information. To generate sounds that can be considered as musical sounds, it was necessary to calculate the frequency of notes following concepts of music theory. The principle of acquiring the muscular signal will be discussed, as well the way this signal is generated in human body. Concepts of electronics such as instrumentation amplifiers and analog filters will be briefly explained. The concept of serial communication will also be explained in the context of this work.

2.1 - SAMPLING AND QUANTIZATION

A signal can be defined as physical quantity, varying in time, which carries information about the behaviour of a system. In general it can be represented mathematically by a function of one or more independent variables. The signals present in this project only depend on time, for example voltage in function of time.

The independent variable is an important quantity used to classify the signals as continuous (analog) or discrete in time (digital). On digital signals, the independent variable assumes discrete values, which can be represented as a sequence of numbers. The amplitude of a signal can be also discrete or continuous. On discrete case, it is quantized; this means the amplitude value can be approximated to a value present in a finite set of possible amplitudes that can be digitally codified.

To obtain a digital signal from an analog signal, the amplitude values must be recorded at regular time-intervals and this process is called sampling. The sampling interval is denoted T_s and its reciprocal, the sampling frequency or sample-rate, is denoted as $f_s = \frac{1}{T_s}$. To reconstruct an analog signal from a digital version of it, an important requirement about the time between samples must be considered and this condition is stated by the *Nyquist Sampling Theorem.* The sampling theorem states, that a band-limited signal, whose components larger than W Hz, can be reconstructed or recovered from its samples if the signal was sampled at least with a rate equal or larger than 2W samples per second (Hz)[7].

$$W < \frac{f_s}{2} \tag{1}$$

Expression (1) represents the condition stated by Nyquist theorem, in which f_h represents the highest frequency present on a signal and $\frac{f_s}{2}$ is known as Nyquist frequency.

After sampling, a second process must be applied and it is called quantization. In signal processing context, quantization is the process of assign discrete values to a signal which its amplitude varies in an infinite values range. A physical quantity is considered quantized and it can't assume vales between two possible ones. On quantization, values are assigned for the maximum and minimum possible continuous input and the continuous values of amplitude are assigned to the intermediate discrete numbers between the possible ranges of inputs. In other words, amplitude of the input signal will be interpreted as a finite set of numbers on a pre-determined sequence.

2.2 - PITCH NOTE/ FREQUENCY CALCULATION

Sound is the propagation of a mechanical longitudinal wave, which propagates through different materials, but normally air. Sounds are a composition of sine signals and they have a speed of oscillation called frequency, which can be measured in Hz (hertz) unit. The human auditory system is capable of detecting sounds in the range of 20Hz to 20 kHz. A musical tone is a steady periodic sound which can be characterized by its duration, pitch, intensity and timbre. A simple tone has a sinusoidal waveform and a complex musical sound can be represented by a sum of sine waves of different frequencies and amplitudes.

Musical sounds can be performed by following a musical scale. A musical scale is a discrete set of pitches used in making music and each pitch corresponds to a particular frequency of sound. A scale has an interval of repetition, normally the octave, in which the frequency of an octave has exactly twice the value of the previous one. Several melodic scales have been created during centuries and the equal-tempered scale will be used to calculate the playable musical notes on the project.

In equal-tempered scale, the frequency value of musical notes is separated by a ratio of rational numbers, in this case $2^{\frac{1}{12}} = a$, which is equal to about 1.059463094359. The basic formula for the frequencies of the notes is given by:

$$f_n = f_0 * (a)^n \quad (2)$$

In equation 2, f_0 is the frequency of one fixed note which must be defined. A common choice is setting the *A* musical note above middle *C* (A_4) at f_0 = 440 Hz. The variable *n* is the number of half steps away from the fixed chosen note. For higher notes, n is positive and for lower notes, n is

negative. The constant *a* is the twelfth root of 2, a number which when multiplied by itself 12 times is equals to 2 [8].

2.3 - ACCELEROMETER – TILT CALCULATIONS

In this project, the accelerometer will be used to convert the angular displacement from user's arm, compare it with a reference position and the information of the angle will be used to modify the frequency of output sound. To accomplish this task, the microcontroller should be able to convert the voltage signal from the accelerometer to a value in angle and for that reason some mathematical considerations about axis orientation and respective angles has to be done.

The accelerometer is a sensor that measures the physical acceleration experienced by an object due to inertial forces or due to mechanical excitation. Conceptually, an accelerometer behaves as a damped mass on a spring. When the accelerometer experiences acceleration, the mass is displaced and the displacement is then measured to give the acceleration [9].

In this project, the accelerometer can be classified also as a transducer, because its input (acceleration) is converted into a voltage signal. In some applications, where the net acceleration or force on a system over the time is gravity, an accelerometer can be used to measure static angle of tilt or inclination. Inclination sensing uses the gravity vector and its projection on the axes of the accelerometer to determine the tilt angle [10].

The equations used for angle calculation follows the instructions presented in the application note [10], written by the manufacturer of the sensor used in project. As the angular position of the sensor changes, a variation of voltage output can be sensed in the corresponding pin for each tri-dimensional axis.

The angles that will be determined by using the formulas are show in figure 1. These angles are named *theta(θ), psi(Ψ)* and *phi (Φ)* and correspond respectively to the variation of x, y and z axis in respect with a fixed reference position.

The formulas used for calculating the angles are presented in figure 2 in which $A_{n,out,}$ represents the value in volts of the current output minus the voltage value for the reference position. In these formulas, *n* represents the axis being measured. A full explanation about how the formulas for calculating the angles are derived can be found in the reference document [10].

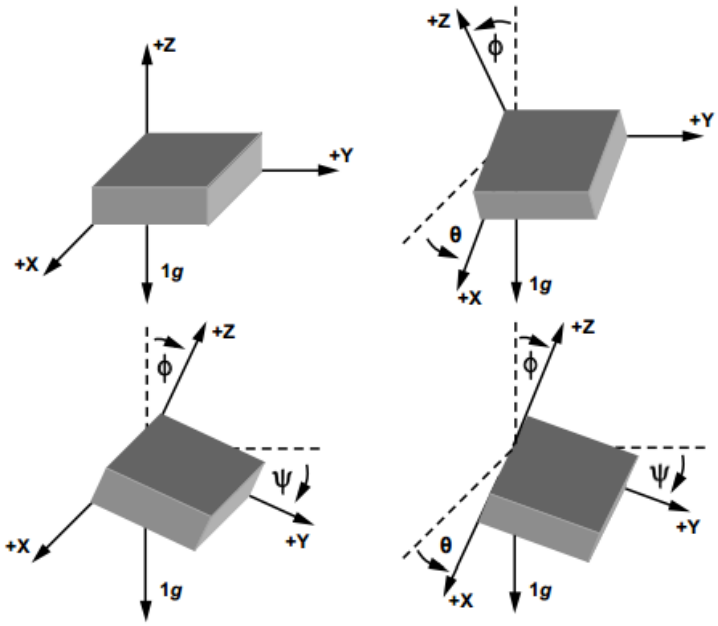

Figure 1 - Accelerometer triple axis inclination calculation

$$\theta = \tan^{-1}\left(\frac{A_{X,OUT}}{\sqrt{A^2{}_{Y,OUT} + A^2{}_{Z,OUT}}}\right)$$

$$\psi = \tan^{-1}\left(\frac{A_{Y,OUT}}{\sqrt{A^2{}_{X,OUT} + A^2{}_{Z,OUT}}}\right)$$

$$\phi = \tan^{-1}\left(\frac{\sqrt{A^2{}_{X,OUT} + A^2{}_{Y,OUT}}}{A_{Z,OUT}}\right)$$

Figure 2 - angles for independent inclination sensing

The inclination angle is given in radians and to convert its value into degrees, equation present in figure 2 has to be multiplied by $(180/\pi)$.

2.4 - PRINCIPLE OF MUSCLE CONTRACTION

The muscle consists of a large number of a structure called muscle fibres. Each muscular fibre is composed of microfibrils and they consist of two filaments called actin and myosin. Contractions on muscles can be classified as voluntary and involuntary and for voluntary muscles are stimulated by signals sent by the brain. These signals are in the form of action potentials and are sent through the nervous system by the motor neuron. Contractions can also occur as result of nervous reflexes of the

spinal cord. Involuntary muscles such as the heart contract as a result of non-conscious brain activity [11].

Muscles can be classified by the type of muscles tissues: Skeletal, responsible for movements; Cardiac, responsible for pumping blood and Smooth, responsible for sustained contractions in the blood vessels, and other areas of the body.

The sliding filament model is used to describe generation of active tension or action potential. Action microfibril consists of two negatively charged molecules in spiral shape and the myosin is much ticker filaments with globular heads and Adenosine triphosphate (ATP) molecule attached.

The action potential originated on brain comes through the nerves and causes a release of a neurotransmitter *Acetycholyne*. This neurotransmitter is sent by structures called transverse tubules that surround the actin and myosin microfilaments. The acetylcholine diffuses and activates receptors on the neuromuscular junction. The action potential also causes the opening of pores that release Ca++ ions. The diffusion of acetylcholine causes the depolarization of inner portion of muscle fibers. The Ca++ ions are bind to the actin filaments making it positively charged. Because depolarization, negatively charged ATP from myosin are attracted to positively charged actin filaments. After the contact of myosin head and actin, a power stroke occurs and causes filaments to slide. The head of myosin now contains Adenosine diphosphate (ADP) and binds to the newly uncovered binding sites, receiving a new ATP and after the stimulus of the brain, the process repeats. Through successive cross-bridging the muscle twitches whiles shortening and work is done. The cross bridge is the process of transferring ATP charge to positively charged actin [12].

The process of contraction in smooth muscles is the same as skeletal muscles, based on the sliding actin and myosin filaments. The difference is because some proteins involved in contraction are not similar.

2.5 - ELECTROMYOGRAPH SIGNAL

An electromyogram (EMG) is a procedure used to record electrical activity of muscles during its activity. When muscles are active, they produce an electrical current, which is proportional to the level of the muscle activity. There are two types of electromyography, one which uses internal electrodes, in needle shapes, and another that uses external sensors, characterizing this method as surface electromyography, or (SEMG). In this report, SEMG will be called by EMG.

The EMG signal can be measured by the difference of potential between two points and has an order of few mV. This signal has low amplitude because the energy generated by muscles is small. The EMG records asynchronous activation of many motor units, which are made up of a motor neuron and skeletal muscle fibers. The motor units close to the surface contribute the most for generating signal. As the signal goes through fat and skin, its amplitude is reduced and an amplifier must be used for making the signal detectable for electronic devices and computers.

Another factor that contributes for the quality of EMG signal is the impedance of the skin. This impedance can vary due to moisture, skin oil and thickness of dead cell layer. To reduce the impedance, it is recommended to use an abrasive cream and alcohol and an electrolytic medium to improve conductivity.

EMGs can be used to detect abnormal electrical activity of muscle that can occur in many diseases and conditions, including muscular dystrophy, inflammation of muscles, pinched nerves, peripheral nerve damage and others [13].

There are two types of EMG recordings; one is called monopolar and the other bipolar. The monopolar one is characterized by measuring the signal between on point over the muscle and the other point somewhere on the body. This last point is called reference and normally is chosen in a place where there is no activity of the analysed muscle. The other type of recording is the bipolar, which is the voltage signal measured between two points over the muscle and a reference point. The reference will act as a ground for the body and device of recording EMG.

2.6 - ELECTRODES

For recording EMG signals, electrodes are placed over the muscle and the reference point. An electrode is an electrical conductor used to make contact with a non-metallic part, for example skin, and a circuit. EMG electrodes can be found in different types and are used for different contexts of measuring muscular activity. The most common types are direct contact electrode, which can be directly placed on a skin and is gold or silver plated, has a layer of Ag/AgCl to stabilise electrical potential of skin and is reusable; Floating electrodes, present on figure 3, which are housed within a cup filled with an electrolytic medium, are suitable to dynamic movements and as single use. This model will be used in the project; Hydrogel electrodes, made by Ag/AgCl and covered with dry sticky layer of gel, as high impedance and can be reused [14].

The electrodes can be classified as passive or active electrodes. The passive ones where described as having the necessity of using a conductive gel to improve its performance. Most of times, this type of electrode can't be reused and this can be considered a problem. The active electrodes do not require the same preparation of the passive ones and can be reused. This feature is available because this type of electrode has an electronic buffer with high input impedance. The signal transmitted is less noisy than the signal transmitted by the passive electrode but as disadvantage, each electrode needs to be supplied with voltage [14].

Figure 3 - floating electrode

2.7 - INSTRUMENTATION AMPLIFIER

An instrumentation amplifier is a type of differential amplifier that has been outfitted with input buffer amplifiers, which eliminate the need of input impedance matching and thus make the amplifier particularly suitable for use in measurement and test equipment [14].

This electronic device is responsible for increasing the amplitude of the EMG signal and making this signal suitable for being analysed by electronic circuits and computers. As the impedance of the amplifier has to be 10 – 100 times larger than the impedance of the skin, this amplifier can be used because among its characteristics, it presents very high input impedance, with a low current consumption. This characteristic allows the signal to be amplified without being distorted. Another important feature of this amplifier is the high common mode rejection. This feature is responsible for cancelling signals which are common to the measured points, for example noise, that is unwanted and causes distortion in the information represented by the EMG signal.

The gain of the amplifier can be adjusted and allowing the correct amplification of the EMG signal. As mentioned in the last section, the EMG signal originally has a very low amplitude value, which it impossible for being measured and processed. It also makes the signal very suitable for influence of different sources of noise.

The circuit present in figure 4 represents the instrumentation amplifier and its basic blocks. The output voltage is shown as equation 3.

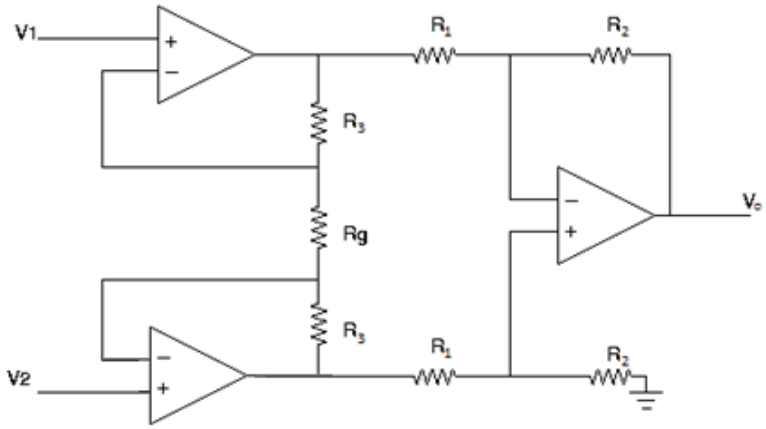

Figure 4 - Instrumentation Amplifier

$$V_o = \frac{R_2}{R_1} \cdot \left(1 + \frac{2R_3}{R_g}\right) \cdot (V_2 - V_1) \quad (3)$$

2.8 - FILTERS

As mentioned previously, different sources of noise can distort the EMG signal, making the information presented in it lost or mistaken. The sources of noise which the EMG recording can deal are the skin, the quality and correct placement of electrodes, the leads that can receive interferences and electrical noise in the amplifier circuit. Noise can classified as instrumental, for example 60Hz from mains, computers, fluorescent lamp, radio frequency (picked up in leads) and electromagnetic force from moving the cables; Biological noise, which consists of ECG (electrocardiogram) artefacts, movement artefact (seen as a DC current shift), respiration and cross talk from distant muscles.

To reduce the effect of these sources of noise, filters are used in conjunction with the amplifier circuit. Filters are the basic building block of signal processing and are much used in electronics. Filters can be designed in different ways following numerous approaches. They can be classified as Analog and Digital. Digital filters are implemented in computers by an algorithm that performs mathematical operations in the sampled signal. Analog filters are designed with electronic components and acts in the continuous signal. Both filters are responsible for separating or rejecting the desired information of the components present in the filtered signal. In this project, analog filters are used due to its simplicity of design and calculation. Analog filters can be classified as active and passive filters. Active filters use active electronic components, for example operational amplifiers, allowing the filtered signal to have a gain in respect with the input original signal. Each active electronic component needs a power supply. Passive filters are easier to build and use passive components, as resistor, capacitors and inductors. The trade-off because the simplicity of circuits is

that the filtered signal will always has a decrease in its amplitude in comparison with the original signal.

There are various types of filter considering the transfer function response in the frequency domain. The main types of filter are: low-pass, high-pass, band rejection or notch, band pass and all pass. The ideal response of each filter is presented in figure 5.

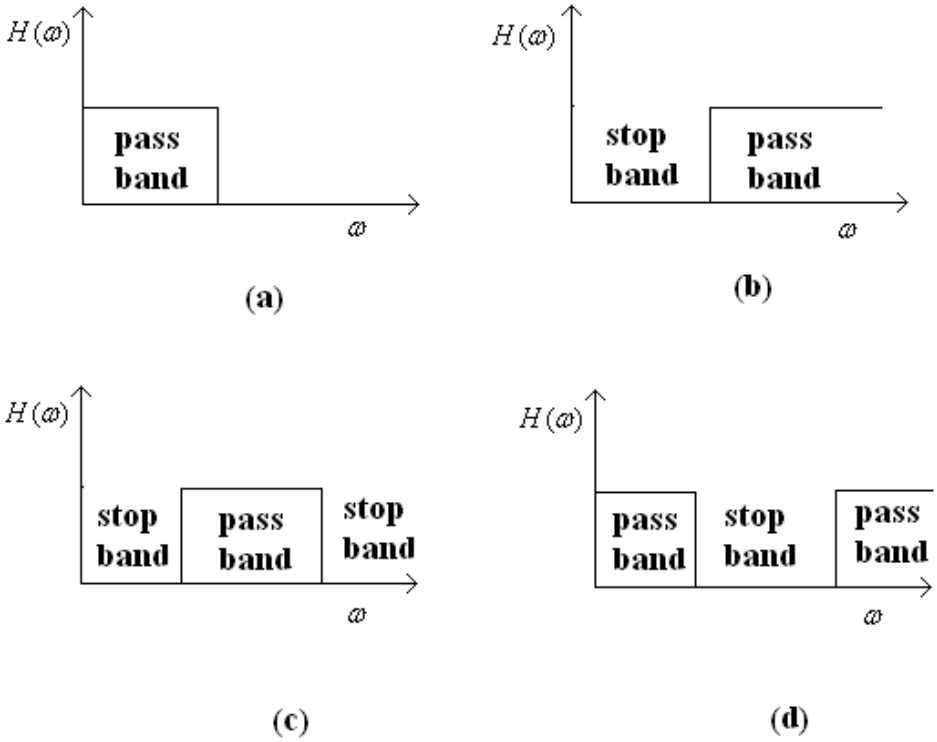

Figure 5 - Ideal frequency response of (a)low-pass filter (b) high-pass filter (c)band pass filter (d) band rejection filter

Each filter has an important parameter called cut-off frequency. The values of electronic components are calculated to reach the desired cut-off frequency, which is the frequency "where the filter acts". The low-pass filter is characterised by allowing signals with frequency lower than the cut-off to be transmitted and signals with higher frequency to be attenuated and not be transmitted. The high pass filter acts as the low-pass, but transmit only signals with frequency higher than the cut-off frequency. The notch filter is responsible to filter only a desired frequency, and in the project will be designed to reject 60Hz, and reduce the noise caused by mains and other electrical activity.

Another important specification parameter for filters is the Quality Factor, or Q. The parameter represents the relation between the cut-off frequency (central frequency) and the pass band with attenuation of -3dB. The higher the Q value, more selective the filter is. The value of Q will depend on the topology chosen and the type of filter used (active or passive). Generally, passive filters can't reach the Q value larger than 0.3 but active filters can have Q larger than 50. It is possible to

increase the Q factor of filter by increasing its order, for example adding more filter blocks in the output of the first filter block. This method is called increasing the order of a filter.

2.9 - SERIAL COMMUNICATION - UART

Asynchronous serial communication in its most primitive form is implemented over a symmetric pair of wires connecting two devices. Data is transmitted one bit a time, sequentially, over a communication channel. One of the devices is called host and the other is target. Whenever the host has data to send to target, it does so by sending an encoded bit stream over its transmit (TX) wire; this data is received by the target over its receive (RX) wire. Transmission of data can also be from the target to host. This mode of communication is called "asynchronous" because the host and target share no time reference. Instead, temporal properties are encoded in the bit stream by the transmitter and must be decoded by the receiver [15].

For accomplish serial connection, part of the microcontroller hardware used is dedicated in translate data from the internal bus from parallel to serial. The UART (Universal asynchronous receiver/transmitter) is a peripheral, part of microcontroller's integrated circuit and has a dedicated function.

The UART is present in both sides of communication channel (host and target), the transmitter side and the receiver side. Bytes of data are transmitted by bits sequentially and a group of bytes is called frame. Each frame is formed by a fixed number of bits and this number depends on the configuration of the serial communication. An example of a frame is show in figure 6.

Bit number	1	2	3	4	5	6	7	8	9	10	11
	Start bit	**5–8 data bits**								**Stop bit(s)**	
	Start	Data 0	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Stop	

Figure 6 - data frame

The bits are represented as changes in voltage logical level. When the level is high, the bit is 1 and when low is 0. "Each character is sent as a logic low start bit, a configurable number of data bits (usually 8, but users can choose 5 to 8 or 9 bits depending on which UART is in use), an optional parity bit if the number of bits per character chosen is not 9 bits, and one or more logic high stop bits".

The receiver tests the incoming data to detect the start and the stop bits and synchronize with the transmitter. All the operations of UART hardware are controlled by a clock signal. To result successful connection and transmission of data, both UART must be configured with the same

settings. These settings include speed, or data rate, which is the number of bits per second and is given in *baud (bit/s)*. Commercial devices operates in standard baud rate values, for example 75, 110, 300, 1200, 2400, 4800, 9600, 19200, 38400, 57600 and 115200 bit/s. Number of *data bits* is another setting that must match between both sides of communication channel. This setting corresponds to the number of bits in each transmitted character and can be 5, 6, 7, 8 and 9, with 8 bits the commonly used number. *Parity* is a method of detecting errors in transmission, and is a setting that can be activated or not. When activated, an extra data bit is sent making the data rate decrease. *Stop Bits* are used to allow the receiver detect the end of a character transmission. This setting has the value of 1 or 2 bits, with 1 normally used.

CHAPTER 3 – PROJECT DESCRIPTION

3.1 – THE SYSTEM

The equipment developed in this project consists in a set of integrated sub-systems, including sensors, hardware and software. These sub-systems work together to accomplish the main goal of project: convert the motion of the body into musical sounds. The technology applied in the equipment, which nickname is *Handheld Sound Generator*, involves different concepts of electrical engineering, biomedical engineering and a small part of music theory.

The system present in figure 7 represents the main parts of the equipment developed. The basic functioning can be described as following: the sensors (electrodes and Accelerometer) are responsible for converting the motion of body into electrical signals, each one in its own manner. The accelerometer is used with a wristlet and positioned in the underside of arm. Two electrodes are placed in the same arm that the wristlet is being used, more specifically the underside of the forearm, two finger away from the bend of the elbow and two centimetres apart each other. A third electrode must be placed elsewhere in the body, but far from first two electrodes, for example in thigh or abdomen. This last electrode is called reference electrode. The correct positioning of both accelerometer and electrodes can be visualized in figures 8 and 9 respectively. The *measurand* of the system are both the angle of arm measured by the accelerometer and the muscular activity electrical signals measured by the electrodes.

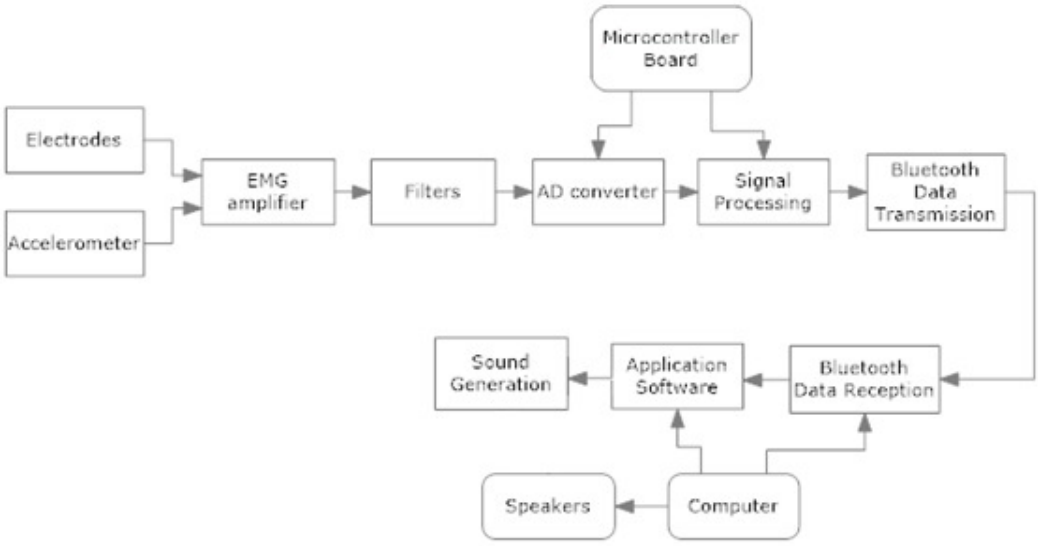

Figure 7 - block diagram of equipment system

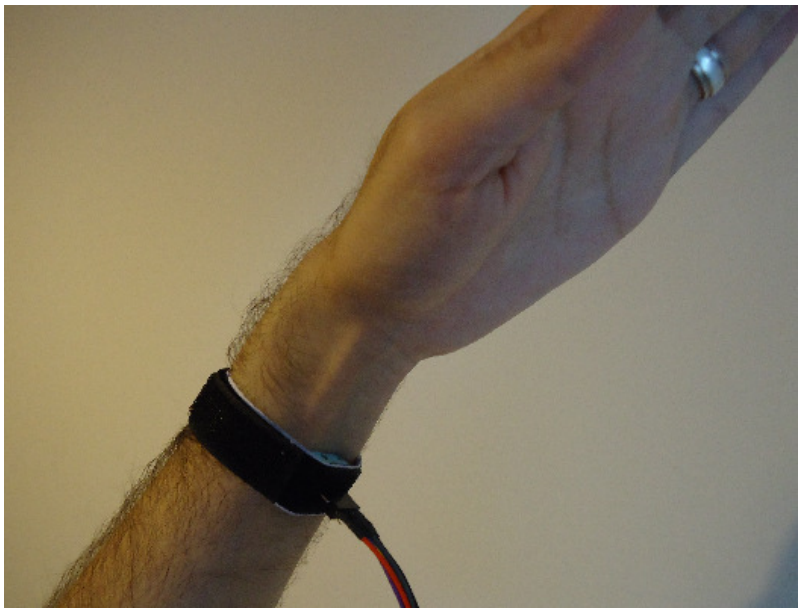

Figure 8 - Accelerometer with arm band

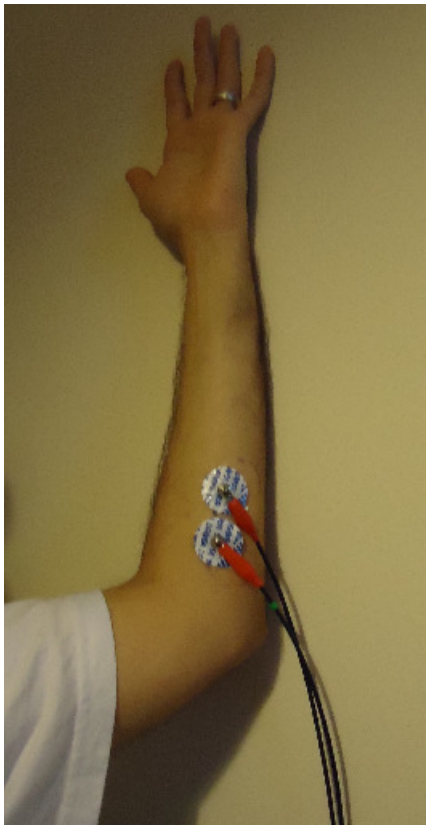

Figure 9 - Electrodes positioning and connection

The accelerometer is connected to the equipment by a cable containing supply voltage and signal transmission wires. The bipolar electrodes (placed on the forearm) are connected with a cable using alligator terminals and jack to one of the four connectors present in the front panel of the equipment. Each connector corresponds to one channel of the EMG amplifier. The reference electrode is also connected to the equipment using a cable and an alligator terminal, but its cable is separated from the bipolar electrodes cable and has its own connector on front panel. The images of the cables described are shown below, on figures 10 (accelerometer) and 11 (electrodes).

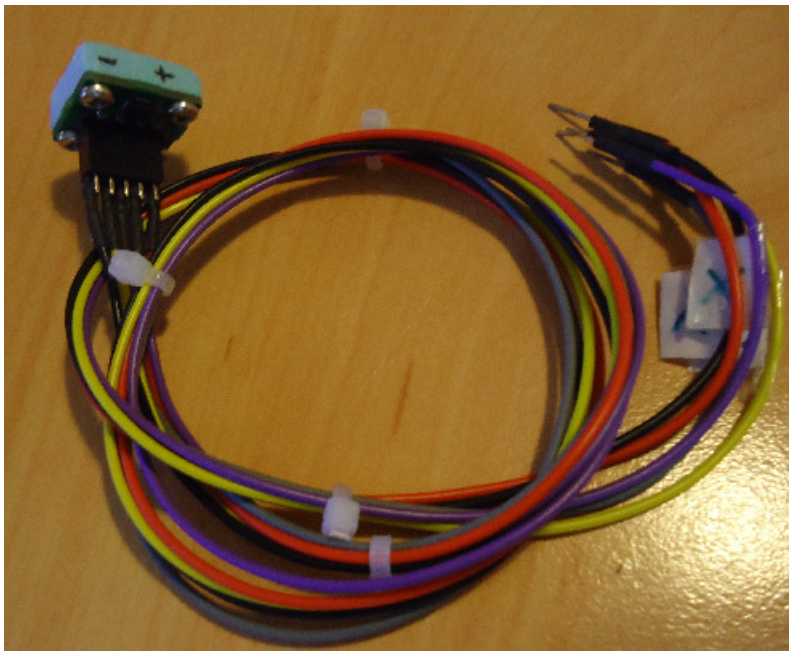

Figure 10 - Accelerometer, cable and connectors

Figure 11 - Electrodes connectors

Inside the equipment, electronic circuits are present for *conditioning* and processing signals provided by the sensors through cables. The main circuits are the 4 channel EMG amplifier, the filters, the microcontroller board and the Bluetooth module. The EMG amplifier increases the amplitude of the measured signals about 1000 times. The filters are responsible for removing the noise and make the measurements reliable for being processed. The microcontroller board is an evaluation kit and has several functions, for example Analog to Digital conversion and serial communication interface. This board converts the filtered signal into useful information, for example the voltage from the accelerometer is converted into angle information. A Bluetooth receiver/transmitter module is connected to the microcontroller board and communicates with it using serial interface (UART). The Bluetooth module is responsible for transmitting the information of muscular activity and angular position of the person using the equipment, to a remote computer.

The computer that will receive the information must have a built-in Bluetooth receiver or an external dongle capable of doing this kind of communication. In this computer, software is responsible for receiving the information transmitted by the equipment and processing it. The result of this processing will be generation of musical sound, which will change its characteristics dynamically depending on the way the user are moving. Every time the user closes its hand or only the middle finger, a sound is triggered and generated by the speakers connected to the computer. Depending on the position of the arm when the sound is triggered, characteristics like timbre and note duration will change. When user lifts his arm, the pitch of the musical note is raised and when it is lowered, the pitch note decreases. By rotating the arm to the sides, the duration of note played can be extended or reduced, with a short note played when there is no rotation and a long note played when the arm is 90 degrees rotated in relation to the relaxed position.

3.2 – HARDWARE

The hardware of an electronic device can be defined as the physical parts or what can be seen or touched on equipment. On the equipment developed in this project, the hardware consists in the electronic circuits and its components: the microcontroller board; the EMG amplifier and filters board (one for each of the 4 channels); the power supply unit comprising the AC-DC board and the DC-DC converter; the Bluetooth module and the sensors, electrodes and accelerometer.

3.2.1 – Microcontroller Board

The Equipment main electronics will be based on a commercial development board called STM32F4 Discovery, from STMicroelectronics (figure 12). A microprocessor development board is a

printed circuit board containing a microprocessor and basic electronic components to make the board work with minimal requirements and be used by the engineer to learning about that specific microcontroller, its characteristics, how to programming and to explore about the possibilities of using that technology on its favour, creating prototype applications for use in future commercial products.

A microcontroller is a small computer on a single integrated circuit containing a processor core, memory, and programmable input and output peripherals. Microcontrollers are designed to execute only a single specific task and are much smaller and simplified than microprocessors, which are used on personal computers. Most of current electronic equipment has a microcontroller, from micro-ovens and toys to modern cars and military jet planes, and the ideas involved from the conception of circuit and the programming logic are about the same, with a difference on the complexity and the way the technology is used. Microcontrollers work based on programs which are written on a personal computer and are saved on the Flash memory of microcontroller. By this reason the microcontroller can work without being connected to a computer and can be used inside devices, controlling other circuits and functionalities, as an embedded system.

The chosen board has the STM32F407VGT6 microcontroller featuring a 32-bit ARM Cortex-M4F core, with 1 MB capacity on its flash memory and 192 KB on the RAM memory. ARM is architecture for microprocessors and this means the way that relations and parts of internal components are integrated and communicate. This microcontroller can be also classified as a DSP (digital signal processor) and this means that it is a specialized microcontroller with its architecture is optimized for the operational needs of digital signal processing, for example to work with filters and floating point numbers (real number). This feature will be important for the necessary functionality of the equipment.

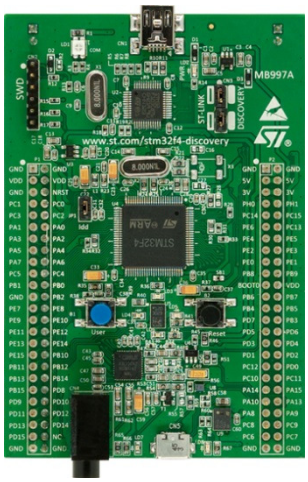

Figure 12 - STM32F4 Discovery development board

The chosen board includes an ST-LINK/V2 embedded debug tool, a digital accelerometer and digital microphone, one audio DAC (digital to analog converter), LEDs, push buttons and an USB OTG micro-AB connector. The board communicates with computer using USB communication protocol by a mini-USB cable and is able to receive computer programs, called firmware, created by engineers, to accomplishing certain functionalities and work as an embedded system.

3.2.2 – EMG Amplifier

As it was necessary to use an instrumentation amplifier to amplify the EMG signal and take advantage of its features, a model of this electronic component was chosen. After some research on internet and analysis of the most popular amplifiers, the model chosen was the LT1167 from Linear Technology. It is a single resistor gain programmable, precision instrumentation amplifier and has as main features the possibility of adjusting gain between 1 to 10,000, very low gain error (0.08%), low input bias current (350 pA maximum) which characterizes high impedance (about 200GΩ). Its CMRR (common mode rejection rate) has a value about 125dB. Full specification can be found on datasheet of component [16]. In datasheet, it also can be found some examples of application circuits including a nerve impulse amplifier (page 17).

3.2.3 – Filters

Because of the characteristics of the signal measured, it was necessary using three different filters: a low-pass with f_c = 300Hz, a high-pass with f_c = 1Hz and a band rejection filter adjusted to remove the 60Hz component, with a high Q value. The low-pass filter has the function of removing high-frequency components, much of it noise and interference but also unnecessary information from muscular activity (unnecessary for this application, but these components can be used for other purposes). The high-pass filter is responsible for removing the low frequency components which may cause error for the signal processing algorithm. These low frequency components are also considered noise and occur because of movements of the body part being measured by the electrodes. For example, when an arm being measured moves up and down, a drift or low frequency modulation can be verified in the EMG signal.

Two approaches were considered when designing the filters needed, one using passive filters and the other using active components. The filter type used on the prototype is the passive one. The topologies used and the components values will be detailed on next chapter.

3.2.4 – Power Supply Unit

The power supply unit consist of two parts: one is the AC/DC voltage converter and the other is the DC/DC voltage converter. The first is necessary to convert the voltage from mains to a DC value, which can be used by electronic circuits. The second is responsible to convert the output from the AC/DC supply to voltage values calculated for the EMG amplifiers and filters work.

The AC/DC power supply was decided to be bought because the complexity of its project and also because the focus of the project was not to design this type of circuit. The model chosen is the ECL15UT02-S, illustrated on figure 13. The main features are 15W of power, outputs of 12V, -12V and 5V and input voltage VAC from 85V to 264V. Full specification can be found on it technical datasheet [17].

Figure 13 - Xppower - ac/dc power supply

The DC/DC voltage converter is responsible to transform the +12V and the -12 V into +3.3V and -3.3V respectively. This is accomplished by using low dropout voltage regulators, as the LM317 for positive voltage and the LM337 for negative voltage. These are adjustable regulators which the output voltage depends on the values of connected resistors.

3.2.5 – Bluetooth Module

The Bluetooth module is responsible for transmitting the data acquired and processed by the microcontroller's board, from the equipment to a computer with capable of receiving Bluetooth data. The chosen module is the HC-05, represented in figure 14. This module is a Bluetooth serial module used for converting serial port to Bluetooth protocol. It was chosen due its simplicity of programming and connection but also because it's low price. The specification of this module is shown below:

Specifications:

- Bluetooth protocol: Bluetooth Specification v2.0+EDR
- Frequency: 2.4GHz ISM band
- Modulation: GFSK(Gaussian Frequency Shift Keying)
- Emission power: ≤4dBm, Class 2
- Sensitivity: ≤-84dBm at 0.1% BER
- Speed: Asynchronous: 2.1Mbps(Max) / 160 kbps, Synchronous: 1Mbps/1Mbps
- Size: 28mm x 15 mm x 2.35mm
- Security: Authentication and encryption
- Profiles: Bluetooth serial port
- Power supply: +3.3VDC 50mA
- Working temperature: -20 ~ +75 Centigrade
- Dimensions: 15.2x35.7x5.6mm

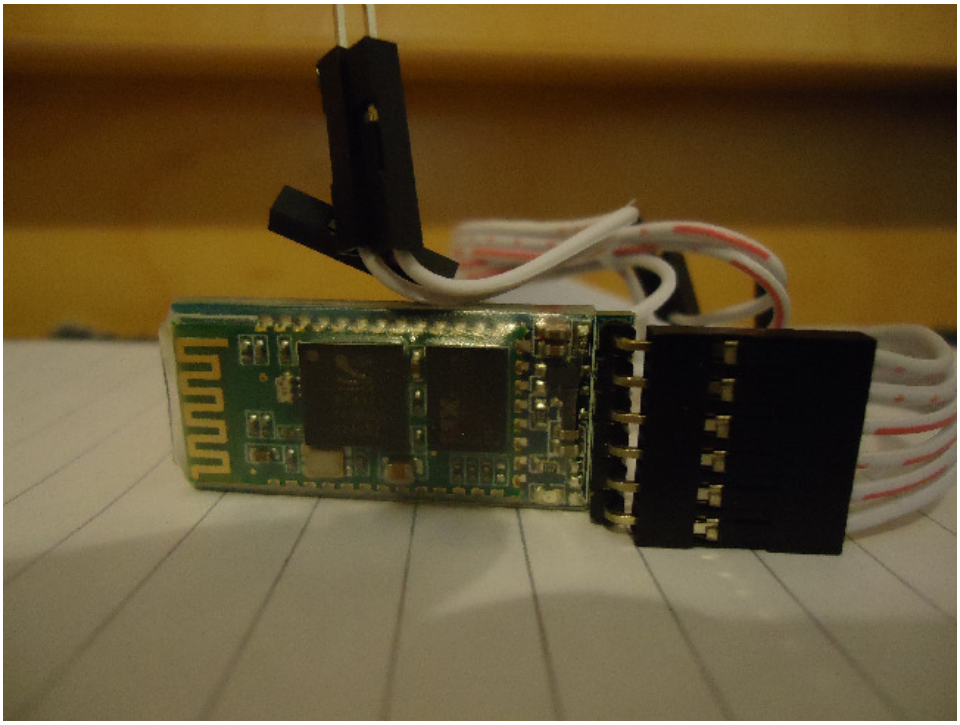

Figure 14 - Bluetooth module HC-05

The dimensions of this module were suitable for being used inside the equipment alongside with the other circuit boards and this module could also be supplied by the same power supply used for the microcontroller's board. The ordinary maximum distance for a reliable connection and data transmission between two Bluetooth devices is about 10 meters, which is suitable for the application in this project.

3.2.6 – Accelerometer

To accomplish the task of converting motion to sound, a special sensor called accelerometer can be used. Accelerometer is device used to measure acceleration, i.e. how fast something is increasing or decreasing its speed. This sensor can be used to detected the orientation of the surface it is placed in, giving information of orientation, inclination and position over the time because its characteristic of being affected by the acceleration of gravity. Some important features of accelerometers that need to be taken into account when choosing a model to be used in a project is the interface of the electronic component and the number of axis measured.

The interface can be basically analog or digital. When analog, the sensor will convert the physical quantity, in this case the orientation of motion, into an analog voltage signal, proportional to the variation of the first quantity. The voltage output needs to be converted into digital by the microcontroller or other external component and be processed by the computer program, so the information about acceleration can be interpreted and used for a task. In digital accelerometers, the output of sensor is already a digital signal or a computer data, for example a digital word, removing the need of using an additional ADC. The digital information goes to microcontroller and can be directly processed by it. Relative to axes of measurement, 3 axes can be measured (X, Y and Z) and accelerometer with different number of total axes measured can be found commercially. The number of axes measured is related to the quantity of information the microcontroller and the application will need about the physical variation, for example the angular position and displacement of the object being measured.

The tilting sensor will be the ADXL-335 accelerometer (figure 15). This sensor is presented in a breakout board and this device is an analog output interface accelerometer, which can measure up to 3 axes and its power can be supplied by STM32 board. The choice of using this accelerometer instead of the digital one occurred by the simplicity present of reading an analog signal using microcontroller peripherals.

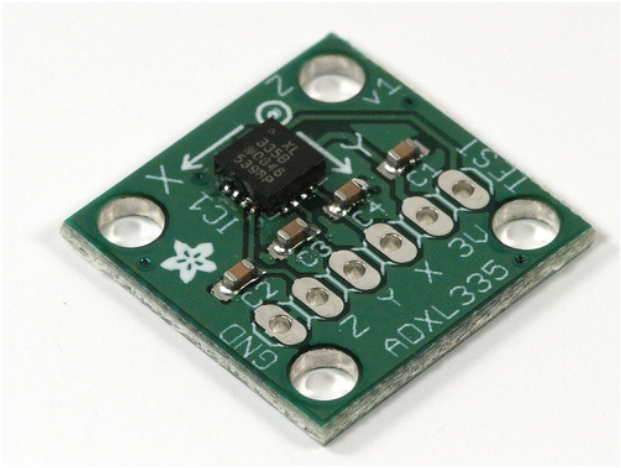

Figure 15 - ADXL 335 accelerometer board

The main features [18] of this accelerometer are listed ahead:

- 3-axis sensing;
- Small, low profile package;
- 4 mm × 4 mm × 1.45 mm LFCSP;
- Low power : 350 µA (typical);
- Single-supply operation: 1.8 V to 3.6 V;
- 10,000 g shock survival;
- Excellent temperature stability;
- BW adjustment with a single capacitor per axis;
- RoHS/WEEE lead-free compliant;

3.3 – SOFTWARE

Software is the part of a device which is untouchable and can't be seen without using a computer screen. It's a set of machine-readable instructions that directs the microprocessor to perform specific operations. The engineer who creates a device is responsible to project and write a computer program that must accomplish certain tasks and control the hardware part of device. For microcontrollers, the program (firmware) is written in a normal computer using specific software called toolchain. The program codes are usually written using a programming language, which consist of a set of particular codes with defined syntax (form) and semantics (meaning). The codes are used to control the behaviour of a machine and express algorithms precisely. Algorithm is a step-by-step procedure used for calculation data processing and automated reasoning.

3.3.1 – Firmware

In this project, the firmware is programmed in C language. This programming language was adopted because it is a high level language and this reduces the interaction of the engineering with hardware, for example adjustment of register banks, memory allocation and initialization sequences. The focus while programming in C is the logic applied to solve certain problem and not the intern microcontroller details.

As mentioned previously, the microcontroller has the function of sampling and quantizing the filtered EMG signal, which comes from the EMG Electrodes through the amplifier and filters but also the 3 voltages from the accelerometer (one for each axis being measured). The microcontroller will also calculate the angular information using the voltage values measured by the accelerometer and create the data package which will be sent by Bluetooth to a computer. The microcontroller is responsible to configure the Bluetooth module using serial transmission parameters and also command the communication.

Next in figure 16, a flowchart is shown illustrating the operation of the firmware.

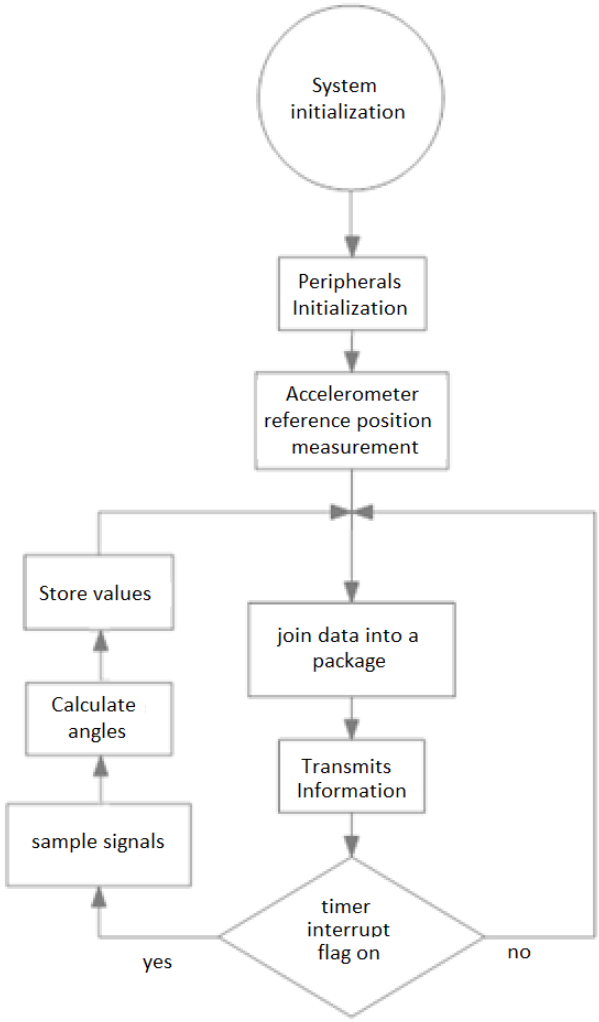

Figure 16 - firmware

3.3.2 – Toolchain

A toolchain is the set of programming tools that are used to create a product, which usually is another computer program or system of programs. A simple software development toolchain consist of a compiler and a linker to transform the source code into an executable program, libraries to provide interfaces to the operating system and a debugger. Library is a set of commands describing a behaviour in terms of a language and are made for be used in different programs, independent of the program's purpose. A debugger is a computer program that is used to test other programs. The code to be examined run on a simulator and changes on states of microcontroller peripherals and memory are shown. Possible errors on syntax of code or odd results can be verified by debugger and are shown to programmer that can correct the errors and make the program work as expected.

In this project, the firmware will be developed using KEIL MDK ARM toolchain. This program is a complete software development environment for the microcontroller of the chosen board. By this toolchain is possible to write a program or a system of programs in C language and store them in memory of microcontroller. The debug tool also is very important because makes possible to analyse the code in real time and compare simulated results with the expected working of the machine. The free version used has the limitation of a maximum 32kb size firmware can be developed. The user interface of toolchain is presented in figure 17.

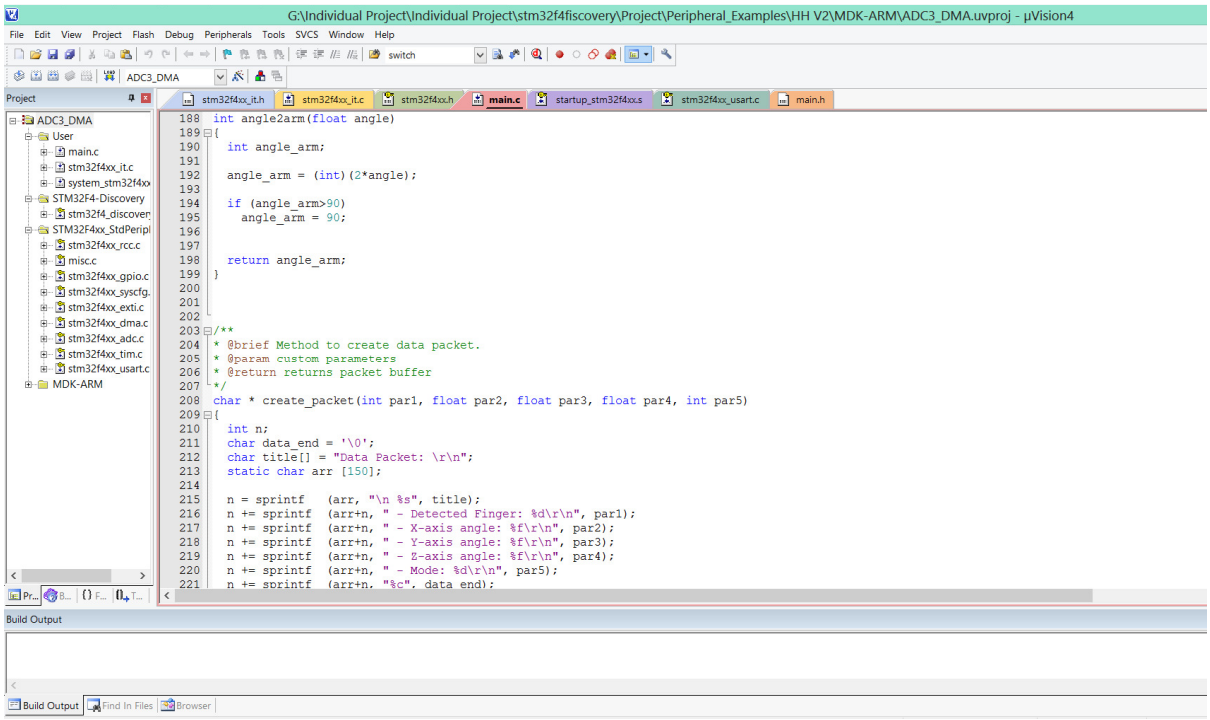

Figure 17 - Keil MDK ARM user interace

3.3.3 – Application Software

The application software is responsible for converting the received data, containing values of voltage and angles, into sound. For accomplishing that, a Python program was written. Python language was chosen because its simplicity of programming and because useful modules can be easily found on internet, providing functionalities needed in complex projects. Python is an object-oriented programming language and high-level, providing features as dynamic type system and automatic memory management.

The Python program can be divided in three main blocks: serial communication which is responsible for connecting the program with the Bluetooth module, handle exceptions, read incoming data and close connection when necessary; Data handling block is responsible for receiving the information from the first block, split the data correctly and assign values for variables and Sound The third, generation block, which contains functions to convert angles in appropriate ways and generating the sound. It also detects when a finger moves and trigger a function, playing sound instantly.

Python scripts can be found in second part of Appendix B, with detailed comments explaining why certain functions were chosen. The block diagram representing the application program is shown in figure 18.

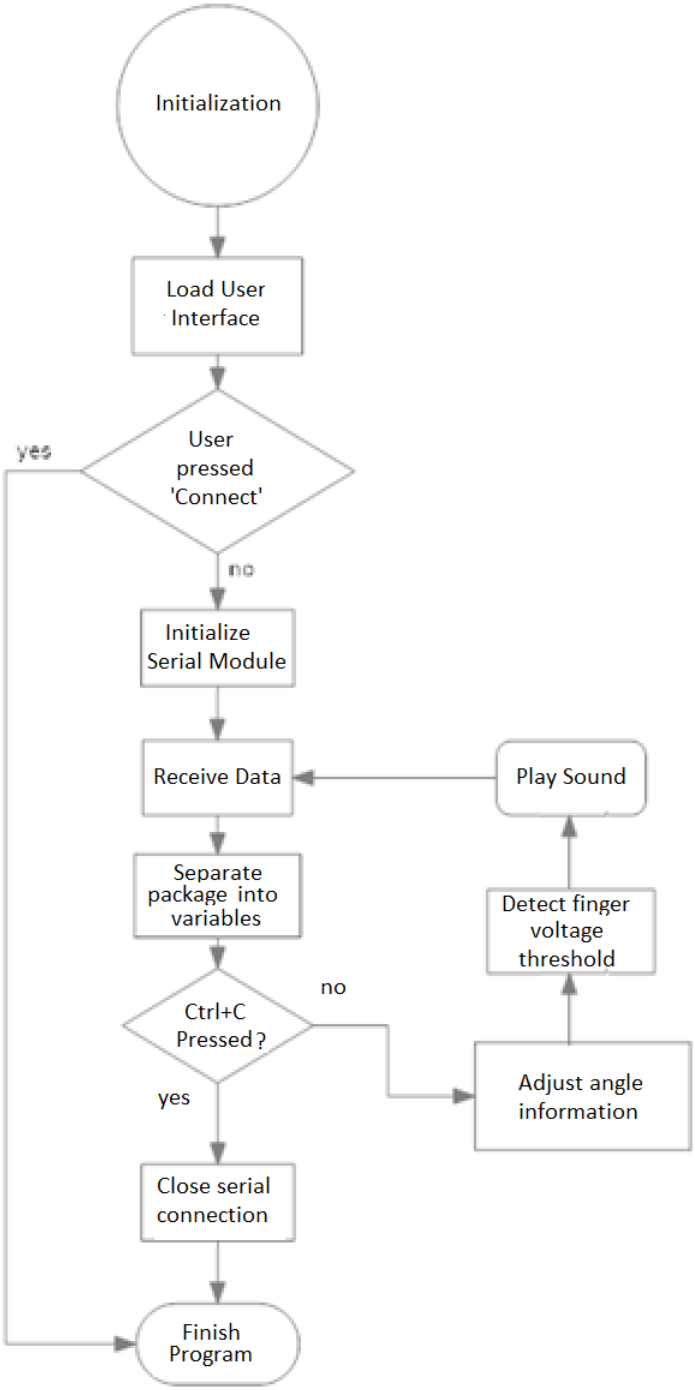

Figure 18 - Python Application Program

CHAPTER 4 – EXPERIMENTAL TECHNIQUES

In the experimental techniques chapter, all the aspects related to the construction of the equipment's prototype will be presented. The study of electronic circuits, projects and simulations will be detailed and prototyping procedures will be explained. The measurement procedures, from electronic circuits and signals, to angular position of accelerometer, will be presented. The processes of configuring the Bluetooth module, programming and setting up the communication between it and computer will also be explained, the programming sequential steps and evolution will be briefly shown as well.

4.1 – CIRCUIT SIMULATIONS

The EMG amplifier circuit was based on the 'nerve impulse amplifier' example present in the datasheet of the instrumentation amplifier chosen [16].

The circuit presented consists in the instrumentation amplifier LT1167 working alongside with another operational amplifier, the LT1112. The resistors connected to the instrumentation amplifier makes this stage has a gain of 10. One of the LT1112 has the function of increasing the gain from first stage, by adding a gain of 100, making the whole circuit amplify the input signal 1000 times. The second operation amplifier has the function of creating a virtual ground for the common mode signal and maintains the stability of the patient ground.

The characteristics of circuit present on figure 19 attend the requisites of the EMG amplifier. The main characteristics of this circuit are the high CMRR (115dB) and high gain (1000 times) to amplify the low amplitude EMG signals.

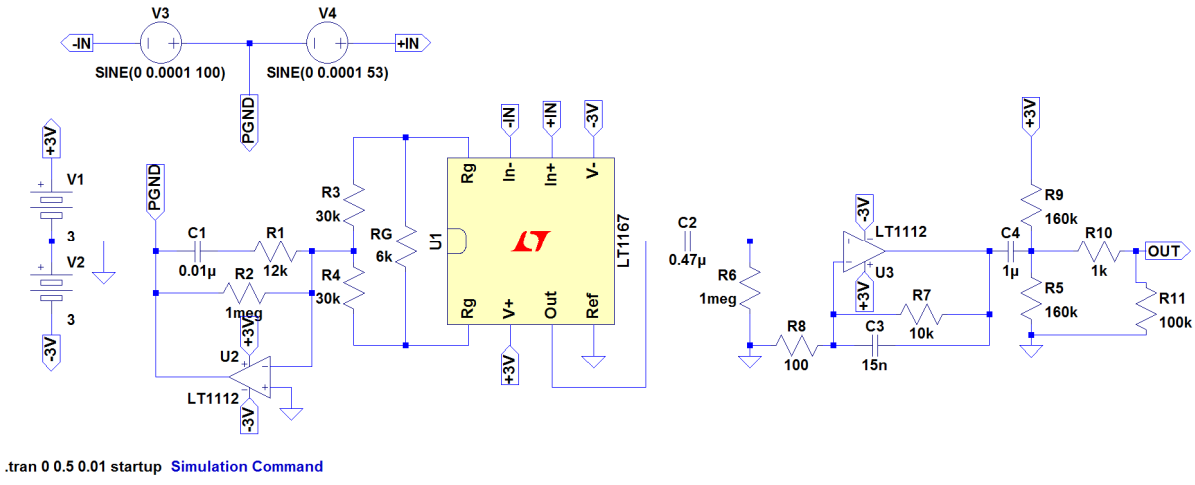

Figure 19 - nerve impulse amplifier

The results of simulation can be found in next chapter. The components C4, R5, R9, R10 and R11 correspond to the ADC interface. The function of this part of circuit is to add a DC offset to the AC signal from amplifier output, allowing the signal to be measured by the AD converter present on microcontroller board.

The next step was to design the filters for rejecting unwanted noise. As mentioned in previous chapter, both approaches (active and passive) filters were used at designing circuits but the passive filters were prototyped and included in the EMG amplifier board.

The passive filters use only resistors and capacitors as components. All filters will be from 1st order, i.e. will have only on stage of capacitor and resistor. To calculate the values of components and reach the necessary specification, online tools were used, whose can be found in [19][20][21]. The suggested design for filters is widely used in different applications because its simplicity and functionality. In those websites, it was necessary to input the value of cut-off frequency or a suggestion of electronic component value. By clicking *calculate* button, the other calculated components are given, in addition to a frequency response diagram and the transfer function. The chosen cut-off frequencies for the LP, HP and notch are 300Hz, 1Hz and 60Hz respectively.

The circuit diagram was designed and simulated again on LTSPICE IV software. Filtering circuit is shown on figure 20.

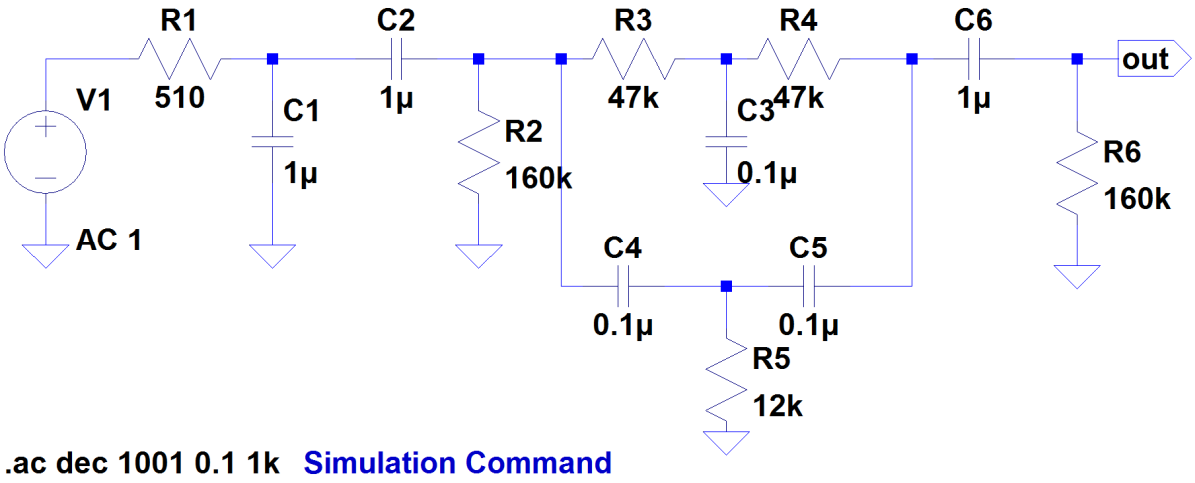

Figure 20 - Passive Filters

In figure 20 it is represented the low pass filter (R1 and C1), the High pass filter (C2 and R2) and the notch filter (R3, R4, R5, C3, C4, C5). C6 and R6 correspond to another high pass and is part as the ADC interface circuit.

The second approach was designing an active filters board. As mentioned above, this circuit was only designed, but the board wasn't manufactured until the moment this work was written. The active filters should have the cut-off frequencies in comparison with the passive filters, but two other parameters were changed. The first parameter was the gain of each filter. As in the active filter, the

gain is a parameter which can be chosen, the value components were calculated to provide a unitary gain or G = 1. The second changed parameter is the order of filter and consequently its quality factor. The topology chosen for each filter will provide a second order filter, with a sharper frequency response.

The topology of the low-pass and high-pass filters was chosen taking as reference the options presented in [22]. For the mentioned filters, Sallen-Key topology of second order and unitary gain was chosen. The values of components were calculated following the recommended procedures for this topology. The value of constants used can be found in Butterworth table, last chapter of [22].

For the notch filters, it was necessary to use a topology different from the options present in [22] because these topologies couldn't provide a quality factor as a parameter while using unitary gain. The solution was found on an online tool [23]. The components used on this circuit provide an central frequency of 60 Hz and Q factor equal 2.5. Figure 21 represents the circuit for the three active filters.

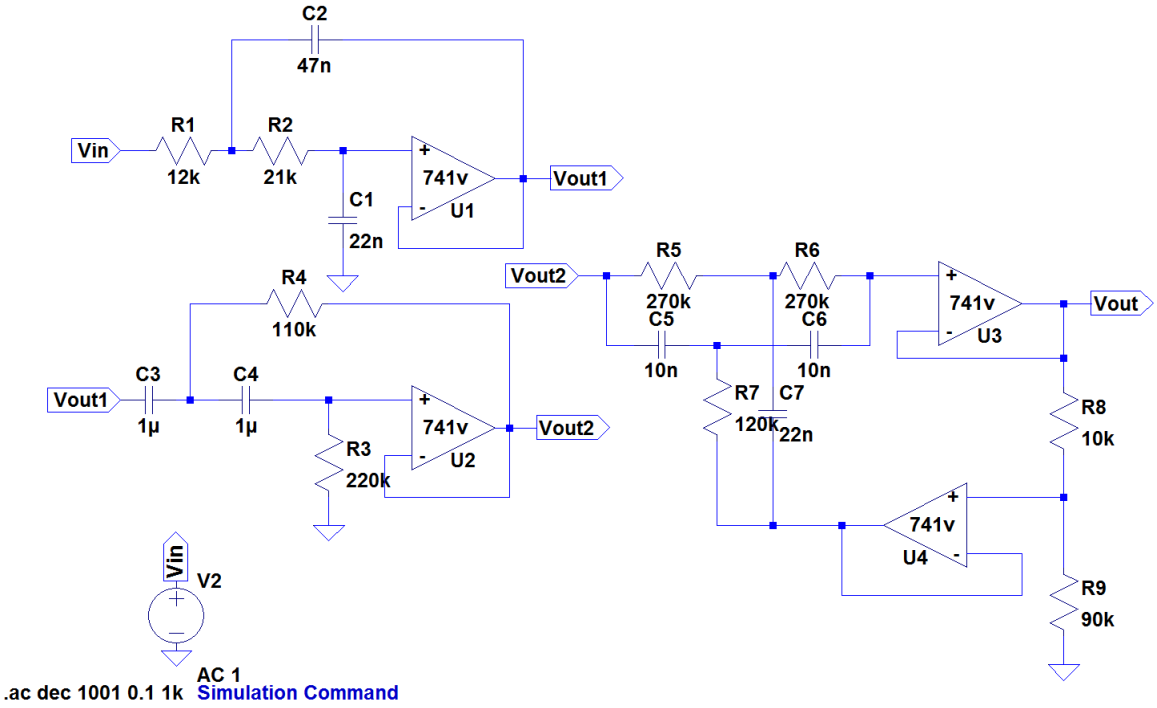

Figure 21 - Active Filters (a) LP (b) HP (c) Notch

The DC/DC voltage converter is responsible to transform the +12V and the -12 V into +3.3V and -3.3V respectively. As mentioned in previous chapter, the main component of this circuit is the voltage regulator LM317. The circuit was based on the examples found in datasheet [24] and resistor were calculated following instructions on it. The requirement for the voltage regulator to work is the input voltage is at least 1.5V larger than the desired output voltage. Power dissipated must not be high to eliminate the necessity of using heat sinks. Figure 22 illustrate the DC-DC converter circuit schematics. Two LEDs (Light Emitting Diodes) were used to indicate the status circuit voltages. The

voltage supply +12V and – 12V comes from the XPPower power supply unit, which is connected to the mains and works as a AC/DC converter.

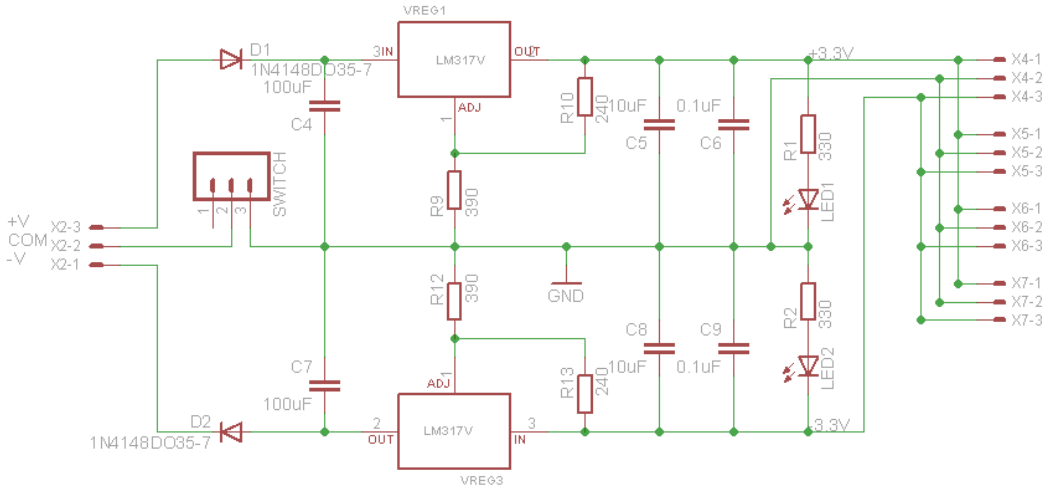

Figure 22 - voltage regulators and LED indicators

4.2 – PROTOTYPING

After simulation the circuit diagrams were transposed to software called *Cadsoft Eagle version 6.4.0.* This software is responsible to convert the electronic diagram to printed circuit board layout. Each component has its model containing the fingerprint and pads in real size. The project of the board consists in allocate the components in the dimension of the board in an optimized way. After that, the software can link the components using a tool called *autorouter*. The software tries to link all points but hardly can it be done. For example, if 90% of the connections are made automatically, the engineer needs to link the unrouted pads manually. Two layers of connections (top and bottom) were used in this board. Figures 23, 24 and 25 represent the allocation of components in real size PCI (printed circuit board) and also the trace connections between component terminals.

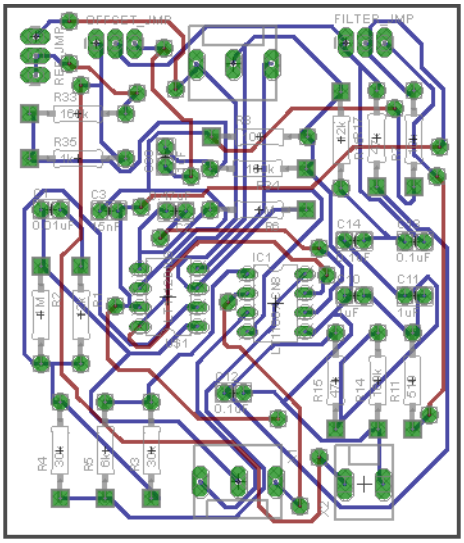

Figure 23 - EMG amplifiers board (original size), red is top layer and blue is bottom layer

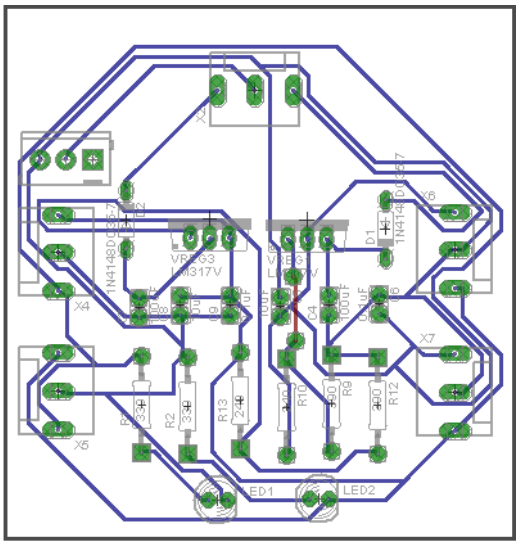

Figure 24 - DC-DC voltage converter (original size), red is top layer and blue is bottom layer

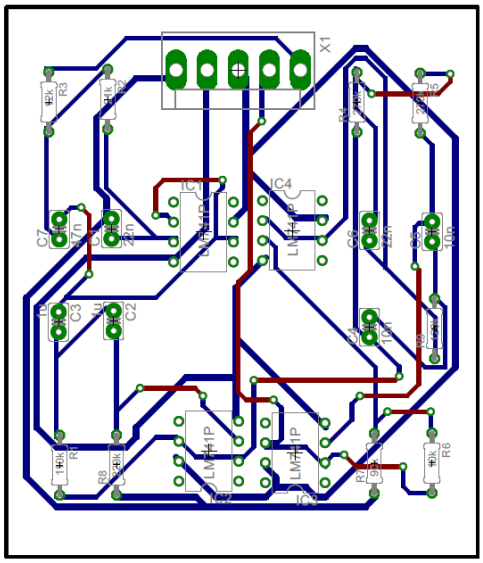

Figure 25 - Active Filters Board (original size), red is top layer and blue is bottom layer

In the project of boards, it was used PCB connectors to connect the boards with wires. The wire must be placed inside one of the connector's input and a screw is rotated, pressing a metallic piece inside connector. This will fix the wire connected to the connector and consequently to the PCB. The dimension of the EMG amplifier board is 60x70mm, the same dimensions of the Active electrodes Board. The DC/DC converter board has 70x70mm.

The boards were manufactured by the technicians of the Electronics Engineering department, from University of Glasgow. They transferred the printed mask to the copper board using UV radiation. After that, the board is submitted to an etching process, which removes the unwanted copper. Only the trace routes between pads remains with copper. The technicians also drilled the holes of component pads, making the process of soldering electronic components the only remaining.

After soldering components to the PCB, connectors were placed and wires connected to them. To accommodate the electronic boards, connectors and wires, a plastic black box were used. The external dimensions of this box are: 58.0mm (height), 102.1mm (width) and 148.0 mm (length). Four drills were made in front panel for placing the EMG amplifiers input jack, and a fifth hole for the reference electrode connector. Two additional smaller holes were made for voltage indicator LEDs. On the cover of the box, two rectangular holes were made for placing the connectors of the STM32F4 discovery board. A smaller box was used for placing the AC/DC power supply. This second box contains a LED for indication of power status and a cable is connected between the power supply and the main equipment box. This cable contains +12V, -12V, GND and + 5V voltages. A second cable is present in the smaller box and contains a plug for the AC voltage.

In the left side of box, a small hole is placed and through it is possible to see the LED status indicator from the Bluetooth module.

Figures 26 and 28 represent the PCBs manufactured. The power status LED indicators are shown in figure 27 with their respective connectors. Figure 29 shows how the connections between amplifiers, DC/DC converter and microcontroller's board are made internally. Figures 30 to 34 show the equipment's final with different views.

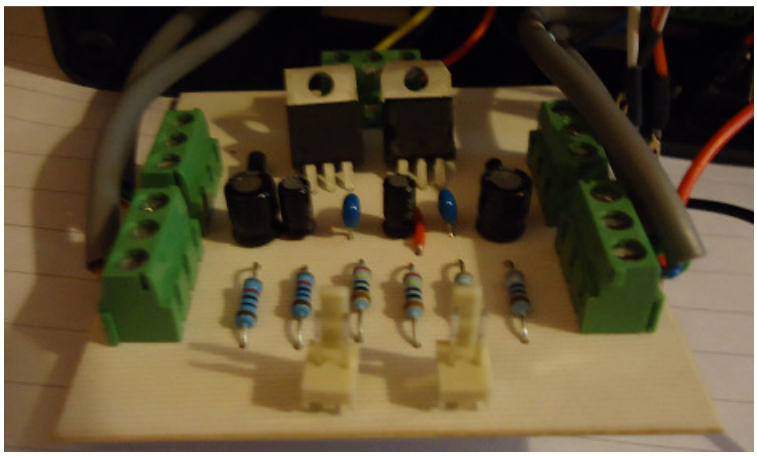

Figure 26 - DC-DC voltage converter board

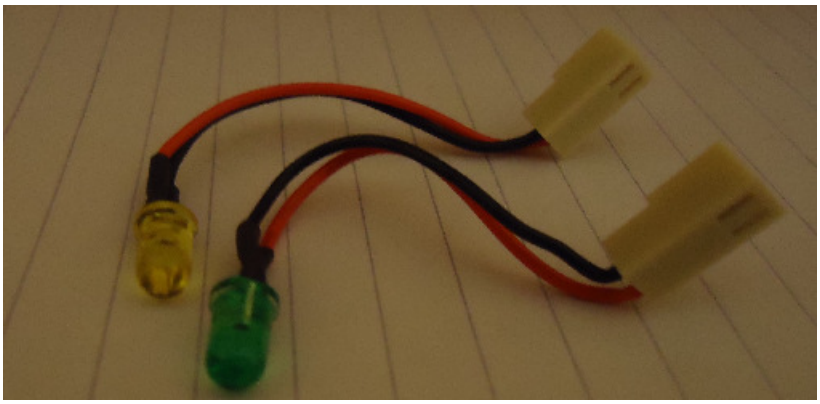

Figure 27 - LED voltage status indicators with connectors

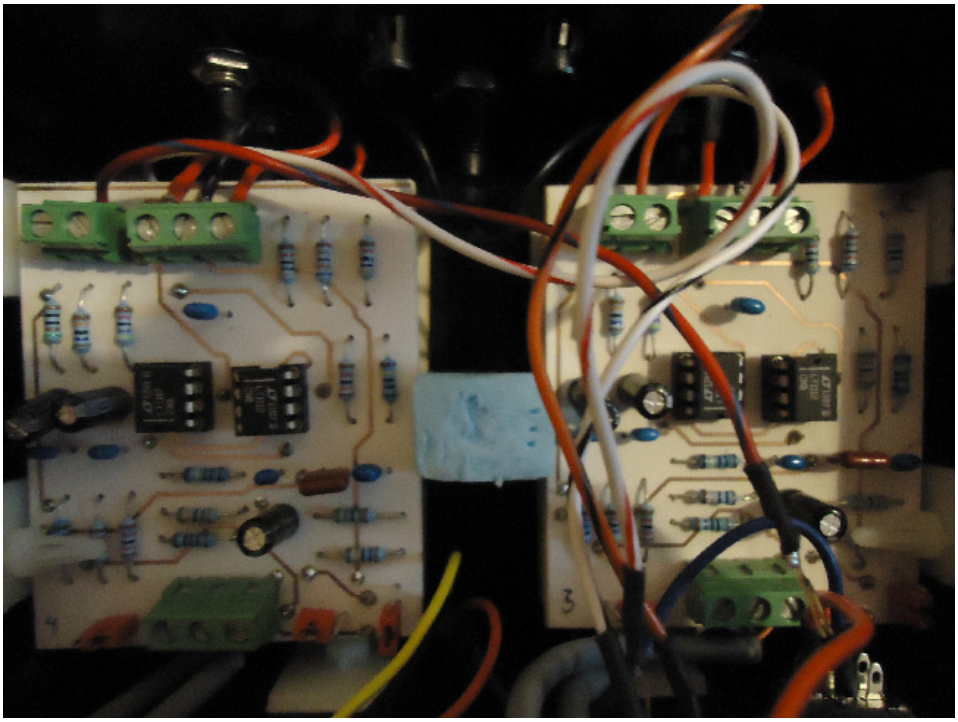

Figure 28 - EMG amplifier boards disposition and connections inside box

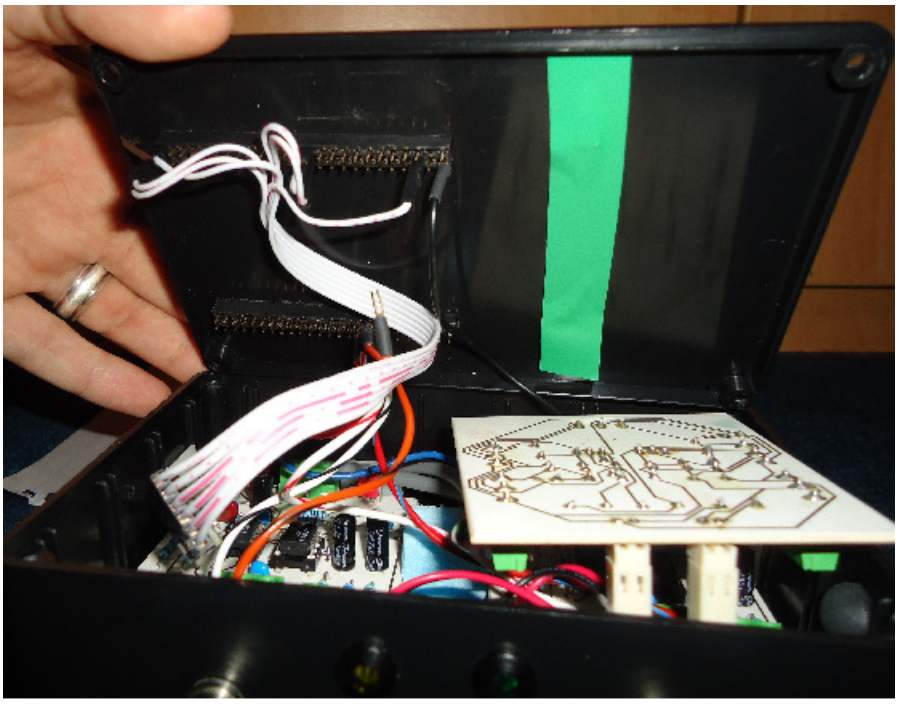

Figure 29 - Intern connections between amplifiers, Bluetooth module and microcontroller board

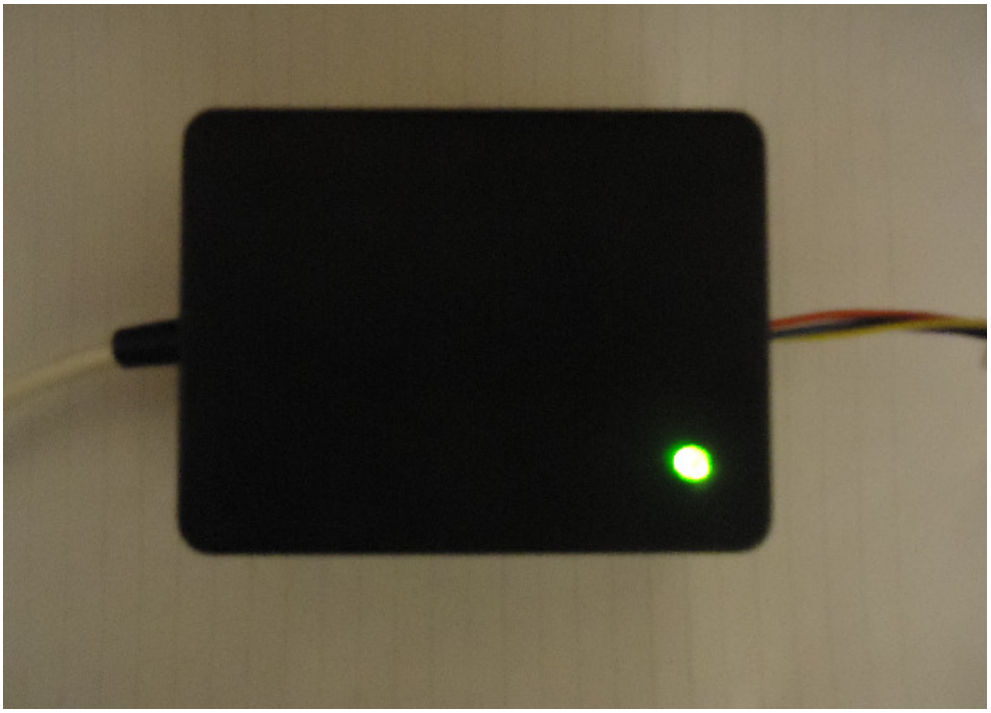

Figure 30 - Power Supply Unit - AC input (left) and DC output (right)

Figure 31 - Device side view with bluetooth LED status indicator (red)

Figure 32 - Device and power supply front view

Figure 33 - Device and power supply back view

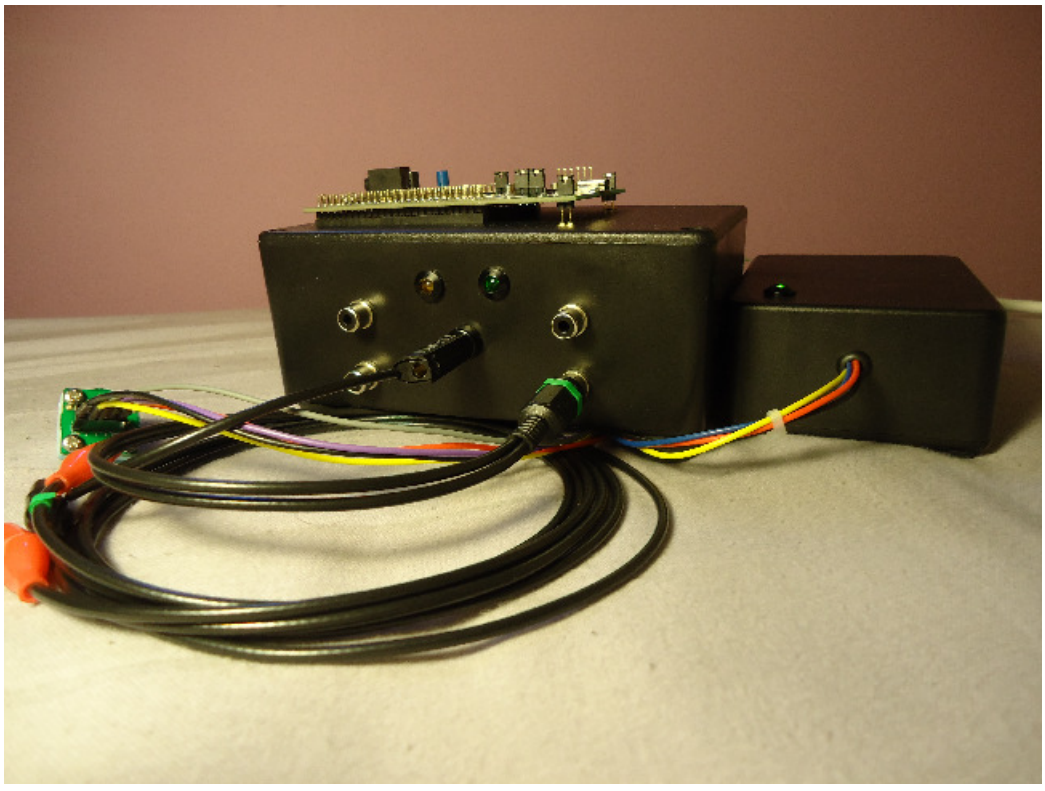

Figure 34 - Device with cables connected (channel 2 connected)

4.3 – MEASURING

For measuring the EMG amplified signals but also other important voltages of the circuits, it was used a portable oscilloscope DSO201 nano, from the SainSmart manufacturer. The main features of this instrument are [25]:

- Super portable and lightweight
- 2.8” color 320*240 display
- Basic 1Msps sample rate with 12bit resolution
- Various measurement markers
- Various trigger mode
- Build-in test signal
- USB chargeable battery
- Open source
- 700Mah 3.7V chargeable Lithium Battery, over 2.5 Hours battery lasting

Signals can be measured and recorded in a micro SD card. The data is transferred to the computer using a SD card adapter and images of signals can be displayed easily in monitor. Figure 35 is a picture of the oscilloscope used in this project.

Figure 35 - DSO 201 nano oscilloscope

4.4 – COMMUNICATION

The HC-05 connection terminals are shown in figure 36. The STATE pin is normally connected to a LED and blinks depending on the status of Bluetooth connection. VCC and GND pins are used to provide supply voltage to module and 3.3V is used in this project. The KEY pin is normally connected to GND but when the module is at configuring mode, this pin is set to VCC. RXD and TXD pins are responsible for the communication with microcontroller and are connected to TX (PB6) and RX (PB7), respectively.

While in configuration mode, a program communicates with module and sends AT commands. These commands are responsible for changing the firmware present in module, and change parameters as name of Bluetooth module as password for pairing.

To connect this module with a computer, a paring operation must be done. In this operation, the computer identifies the Bluetooth module by its name and asks for a password. The password can be the default one programmed on Bluetooth firmware or a modified one. After entering the correct password in computer, the two devices are paired and data transmission can occur.

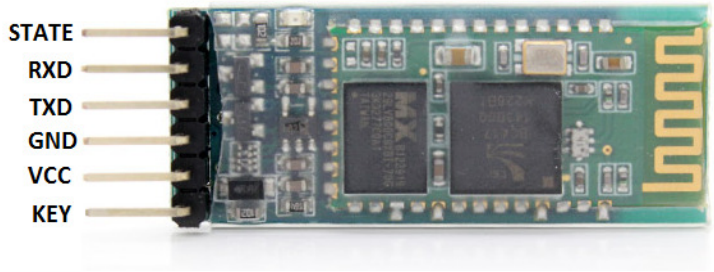

Figure 36 - Bluetooth module terminals

UART was configured to transmit 8 bits of data, with no parity and 1 stop bit, using a data rate of 115200 baud. This value was calculated based on the amount of information necessary to be transmitted.

Information that will be transmitted from the equipment to computer by Bluetooth is stored in an array of characters and has the following form: **[aaaaxbbxccxddxee\n].** In this array, *aaaa* represents the voltage detected by the electrodes and amplified by EMG amplifier board, for example 1204 represents 1.204V. Next, *bb*, *cc* and *dd* represent three angles (theta, psi and phi), in degrees, as explained in previous sections. The double *e* is a generic command which for example can represent a flag (01 or 05 or 11 are examples). Each *x* is a data separator which will be used in the next program. The last is a special character '\n' called *line feed*. This character is used to represent the end of a line or end of transmitted data.

4.5 – PROGRAMMING

The experimental techniques for programming the firmware and the application software were basically the same. It consisted on determining what functions would be necessary for accomplish the main requisites of project and studying lots of code examples whose can be found on internet. After studying the examples, the code started being written and was tested.

The development of firmware present on microcontrollers board started by choosing the appropriate toolchain, studies on the manufacturer's (STMicroelectronics) example codes and templates and then writing the firmware.

The choice of Keil MDK ARM as a toolchain was made by comparing this software with others of same type suggested by manufacturer. The chosen program presented as advantage, the possibility of using a debugger tool, the fact of have full functionality, with a limitation on code length of 32kB, which is enough for application. Numerous examples of projects and tutorials were available and would help on understanding the toolchain and how to work with its functions. The software also was free and capable of working with the STM32F4 Discovery evaluation board.

The familiarisation with microcontroller and development board is an essential step necessary before the implementation of functionalities required by a project. Example codes and application notes are usually provided by any development board manufacturer to help the user get started quickly with their product. On example codes, preconfigured functions and libraries can be found and they help the user to understand how that board and microcontroller works and the knowledge acquired by studying examples, can be used by the engineer or programmer to think the best ways of programming that device to execute the functions needed. Application notes are also helpful because they present detailed description of case studies using the microcontroller or board. The problems described on applications notes can have the same solutions of problems that could happen on the process of creating a new application for a device.

For the equipment developed in this project, some specific tasks should be accomplished, including converting the signal from the accelerometer into a signal that can be processed by microcontroller, processing the angle information and the EMG signals, and send information via serial communication. Some peripheral firmware examples were used and the most helpful studied for creating the firmware will be explained.

The first program studied was the GPIO toggle example. This program has the function of making the four LEDs present on board to blink from times to times. By studying this program was possible to understand how to configure de LED's on board using built in functions and how to creating a delay function, using the timer of microcontroller.

After that, an example called EXTI was studied. This program shows how to configure an external interrupt line. An interruption is a function or feature present on most microcontrollers and

this function can make the microcontroller stop of executing an instruction, execute an important instruction when required, and return executing the previous instruction. The useful information about this program was to learn how to configure the external interruption that could be used to make buttons activating specific functions of the program.

The next programs were the related to analog to digital conversion. It was possible to learn how to convert an analog signal connected to the pin of microcontroller into digital information, which could be processed. The variation of the analog quantity, in this case a voltage signal was made using a potentiometer. The potentiometer is an electrical resistance that can have its value changed by rotating a knob. When a fixed voltage is applied to its first and last terminals, a varying voltage can be measured on the middle terminal. The middle terminal was connected on the correct pin of microcontroller. The maximum voltage that can be read by microcontroller is 3.3 V and this value is because the power supply used for the board.

From the knowledge obtained from studying example codes, it was possible to write the firmware with all its functionalities. The process of developing the firmware was making functions, testing of function isolated, joining functions and testing integration.

The process of programming the application software was basically the same used for the firmware: study of examples, write basic functions, test smaller programs, join programs creating new functions and modules, searching for required modules on internet and test of functionalities and integration.

The application program consists on the integration of seven python scripts. The name of each script and its function is listed below:

app_gui.py: This script is responsible to generate the graphical interface. Improvements can be done in future to make interface more friendly and interactive.

app_main.py: Main program. This program is responsible to execute and call the other files, assign the main variables and play sound.

app_serial.py: Class for receiving data. This class is responsible to configure the serial port, open connection, receive data and close connection.

app_manage_data.py: Data handling class. This class will split the values of the received string accordingly to the protocol.

app_processing.py: Functions to convert data received to appropriate values.

app_sound.py: This script is used to generate sound using Python Musical and Pygame modules.

app_setup.py: This script is used to generate the EXE file from python script.

CHAPTER 5 – RESULTS

It shall be shown in this chapter the results obtained from the final working device. The evaluation of angle measurements will be compared for the analog and digital versions. The measured EMG signals will also be presented as images from the oscilloscope. Circuits simulation will be presented for illustrate further discussion.

5.1 - ELECTRONIC CIRCUITS SIMULATION

As mentioned previously, the simulations were made by software, using the LTSPICE IV software. The component models were found on the standard components library, which comes with the main program. To draw the circuit, it is needed to choose the necessary components in the components library list place them on the screen, assign values for the components which need it, and then draw wires connecting the terminals.

After running the simulation of the circuit shown in figure 19, the following graphs were generated. Figure 37 represents the source signal, generated by the difference of power supplies V3 and V4, with 100Hz and 53Hz respectively, and 100µV amplitude. Its maximum amplitude is nearly the value from the output of electrodes and the frequencies are two components of the EMG signal spectrum. Figure 38 represents the signal after the amplification of the instrumentation amplifier. The gain of this first stage is 10. The signal with total amplification and first filtering can be found in figure 39. At this stage, the total gain is about 1000 and a transient behaviour can be verified. The amplitude varies in a range of 400mV. All signals can be compared in figure 40, as they are presented in the same graph just to give a notion of the amplification process.

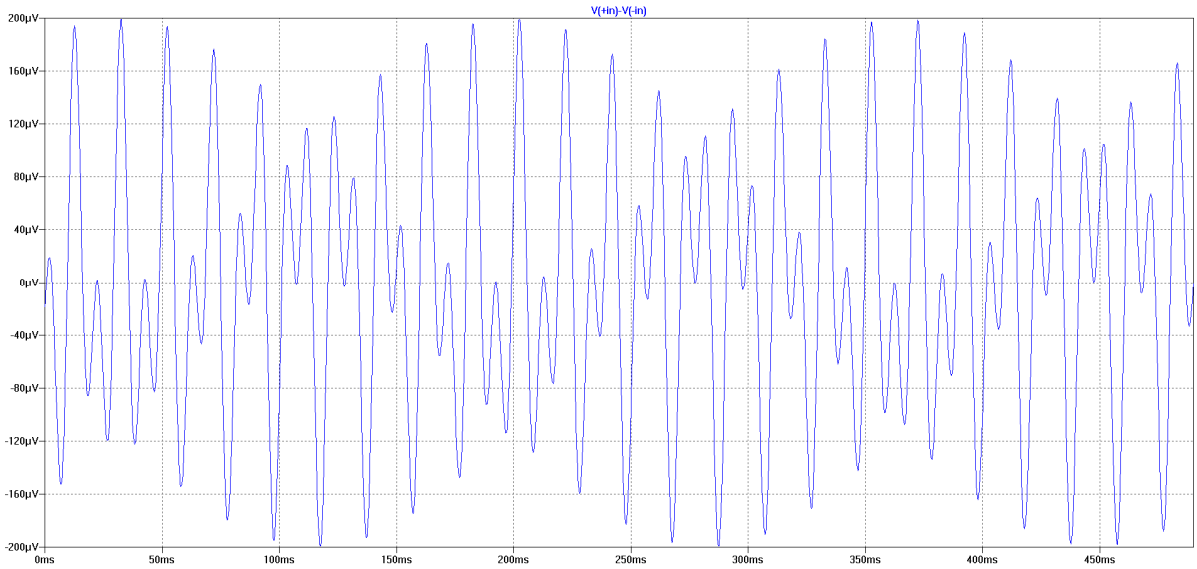

Figure 37 - signal source simulating EMG signal

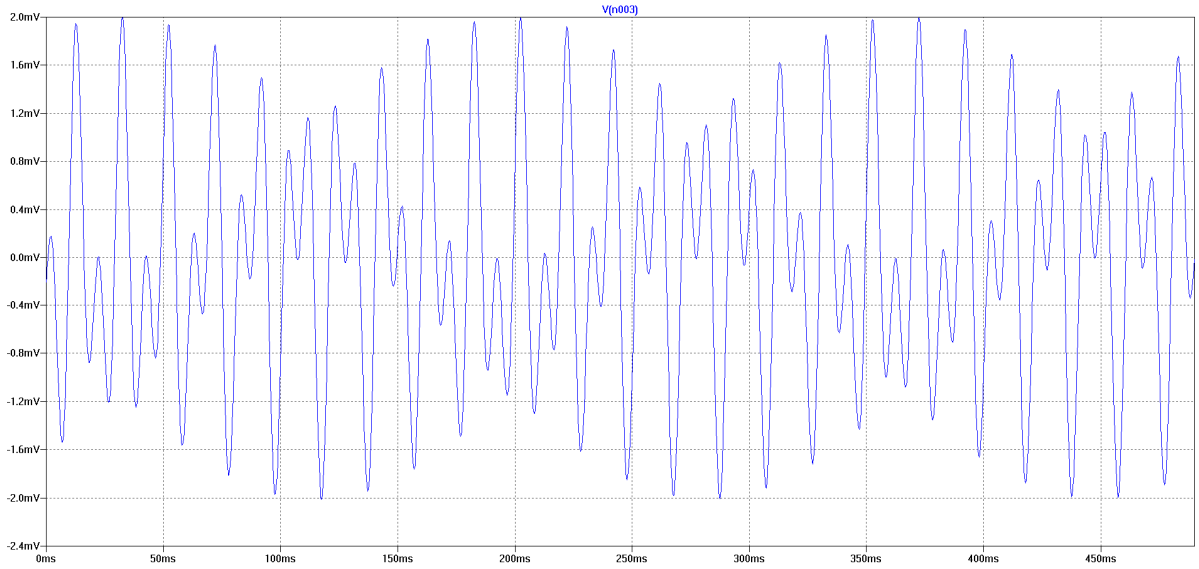

Figure 38 - signal after instrumentation amp, gain = 10

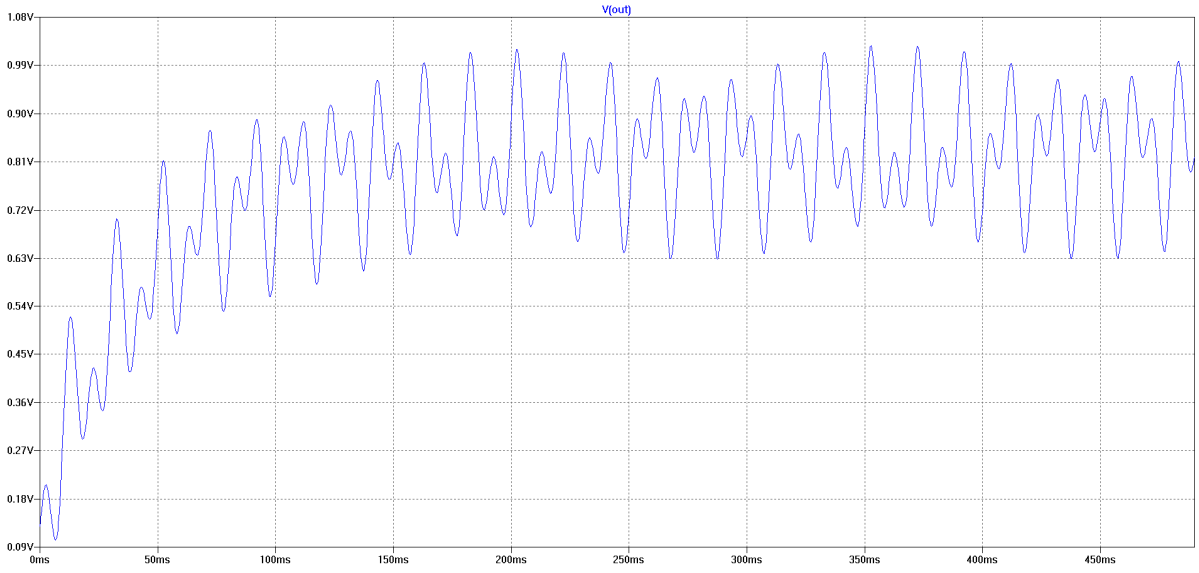

Figure 39 - signal after second stage, gain = 1000 and filtered

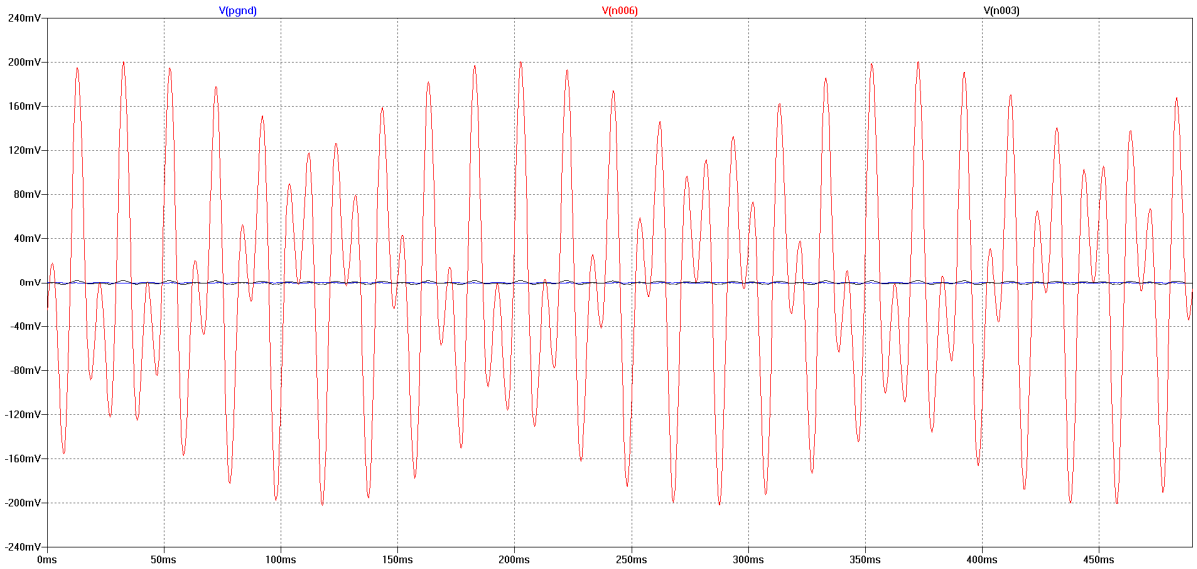

Figure 40- output signal (red), signal after first stage (blue) and from source (black)

Figures 41, 42, 43, 44 and 45 represents different frequency responses for the filters of the passive filters block (figure 20). The circuit's schematic was generated the same way as the amplifier circuit, by placing components on screen and assigning values for it. After running, it is necessary to click in the point which the measurement is required to be taken. Figure 41 represent the signal in the point between R1 and C1 and is the signal filtered by a low pass filter, with about 300Hz as cut-off frequency. The previous measured signal, after being filtered by a high pass filter, is shown in figure 42. The cut-off frequency in this stage is 1Hz and signal is taken between C2 and R2. As all the frequency response graphs, y axis represents the amplitude in dB and x axis is the frequency in Hz. The trace-line and the right side of the y-axis represent the phase response. The effect of the addition of the 60Hz notch filter is represented in figure 43. Figure 44 represents the effect of all filters, with the addition of another high-pass filter with cut-off frequency 1Hz, present on the ADC interface circuit. In figure 45 is possible to check all the frequency responses in the same graph. The colours dark-blue, black, light-blue and red represents the 1st, 2nd, 3rd and final stages of amplification respectively.

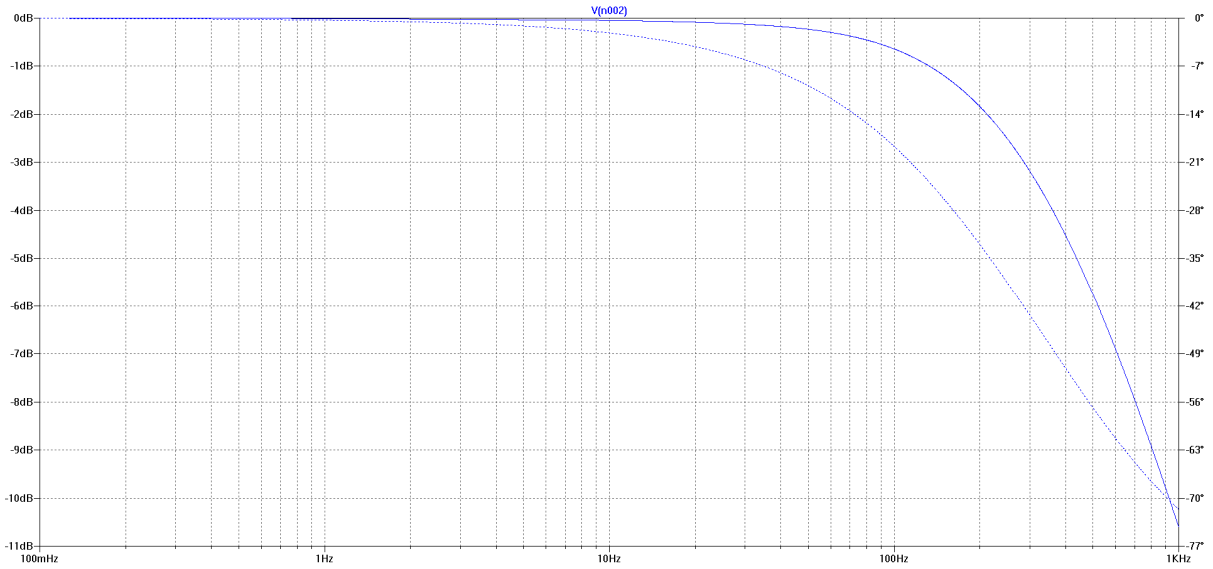

Figure 41 - LP fc = 300Hz frequency response

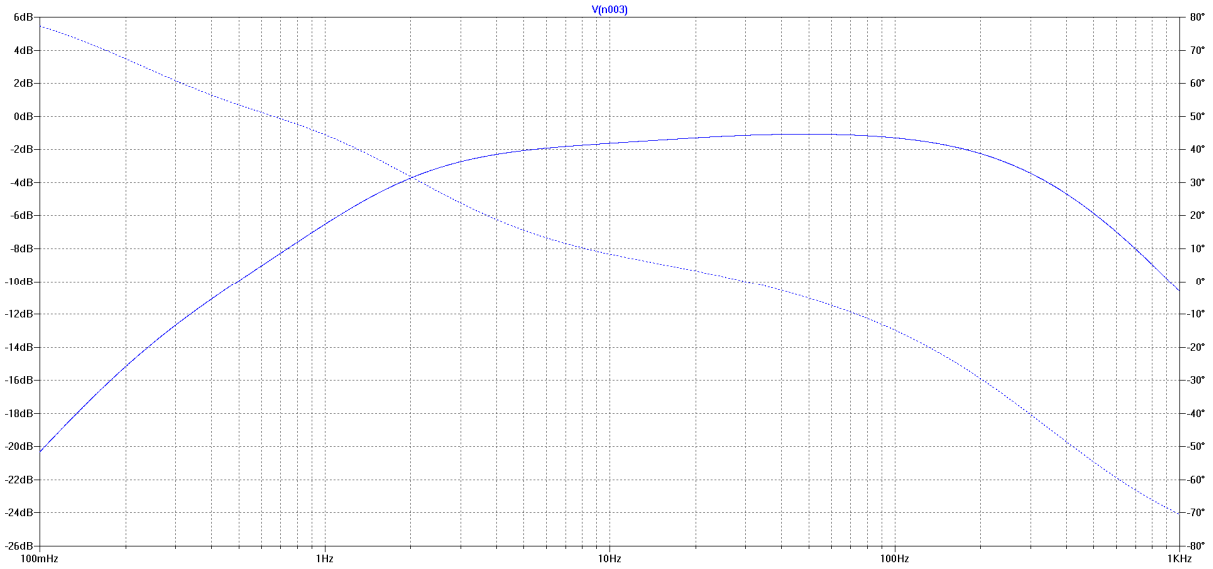

Figure 42 - LP(300Hz) + HP(1Hz)

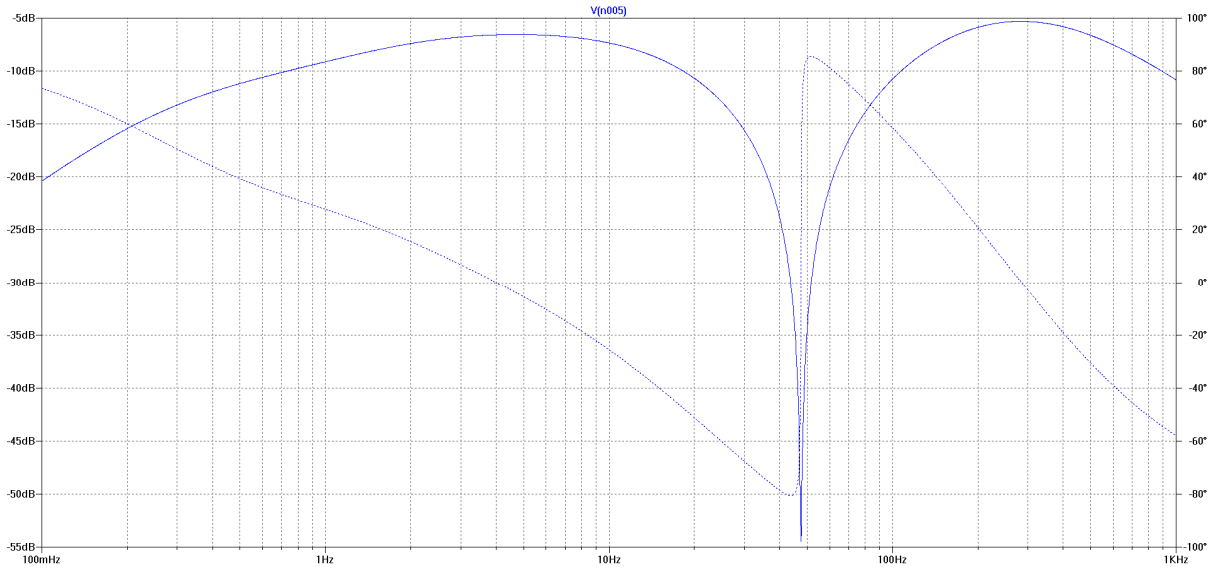

Figure 43 - LP, HP and Notch (60Hz) frequency response

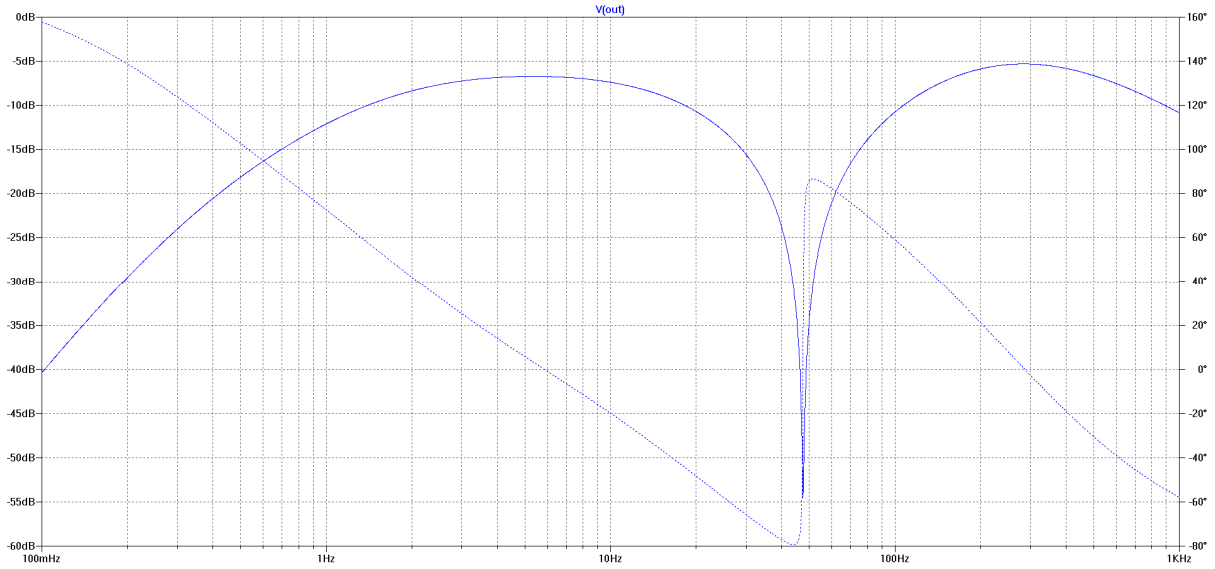

Figure 44 - all filters frequency response

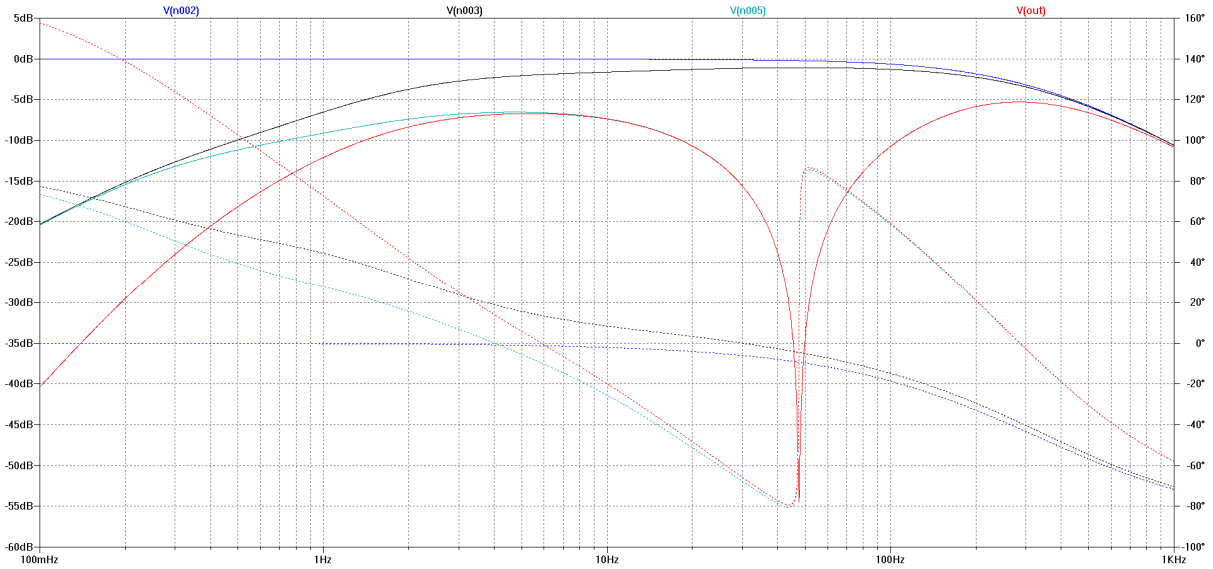

Figure 45 - frequency responses in the same graph

The active filters present on figure 21 were simulated and their frequency response is show in figure 46. The frequency parameter of each filter is has the same value of the equivalent passive filter, i.e. LP (300Hz), HP (1Hz) and Notch (60Hz).

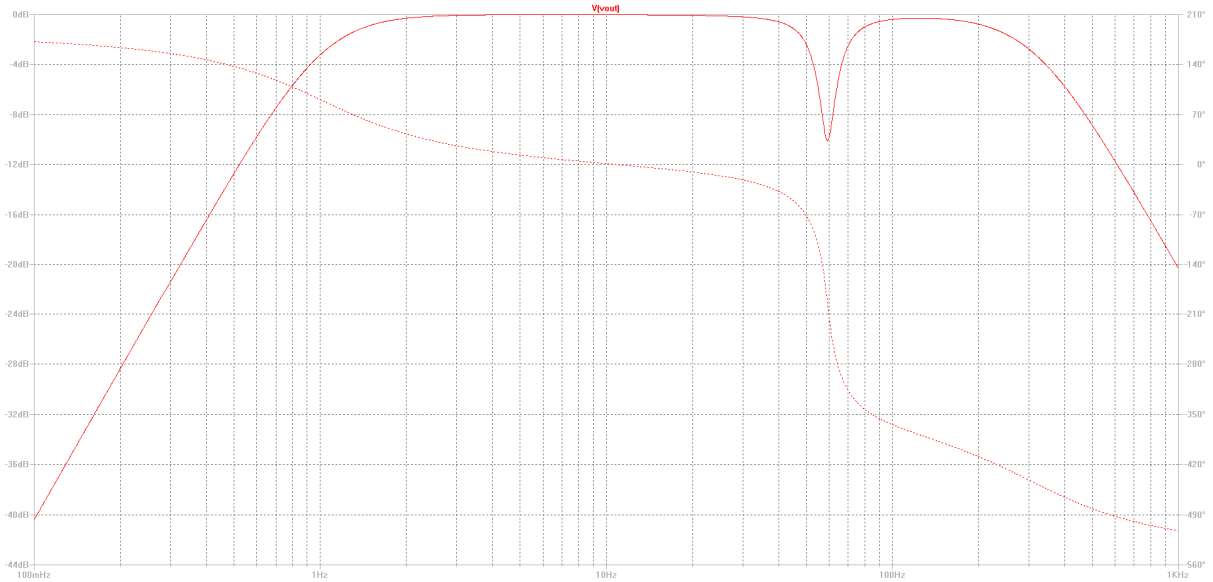

Figure 46 - Active Filters frequency response

5.2 - EMG SIGNAL MEASUREMENT

In this section, the signals measured from the PCB outputs are shown, in the format of oscilloscope measurements. All measurements were made by placing the electrodes correctly on the arm of the user as explained previously. The signals were measured using a DSO-201 portable oscilloscope, with 1x amplification probe. The electrodes were placed on the downside of the arm, 10 cm distant from the wrist and 2 cm apart each other. The reference electrode was placed on the left thigh.

The resting arm signal is shown in figure 47. The arm was stretched with the palms facing down. Figure 48 represents the EMG signal amplified, caused by the flexion of the thumb. The flexion was made by continually closing the finger and returning to the rest position. Figures 49, 50, 51 and 52 represent the measurement of the same procedure for the other fingers.

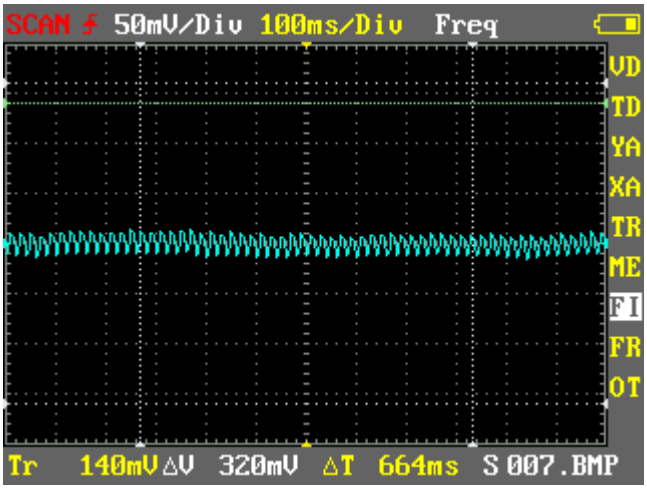

Figure 47 - arm resting position

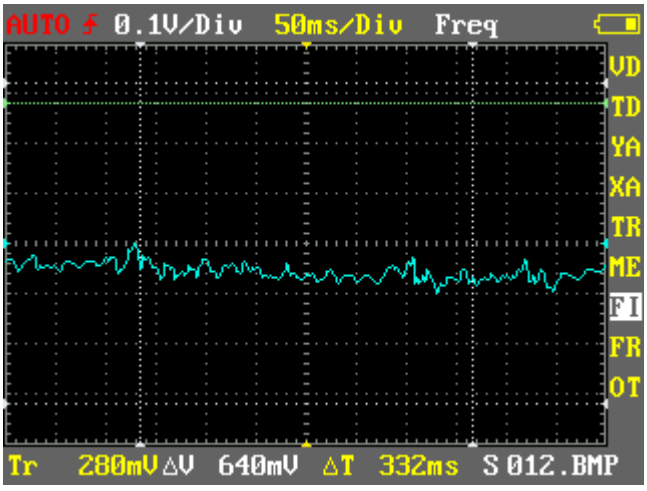

Figure 48 - thumb flexing

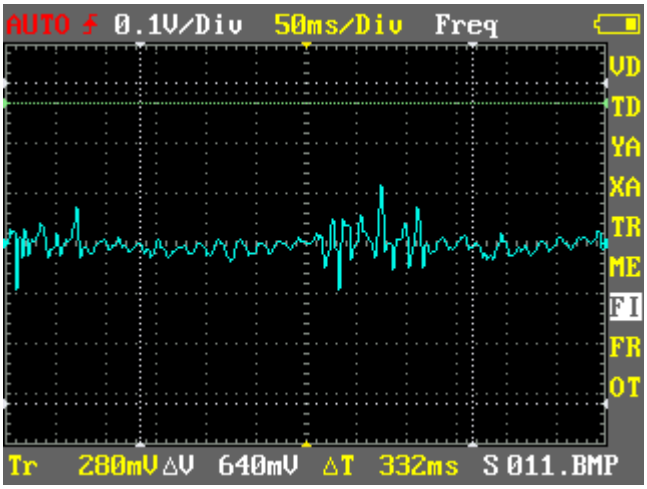

Figure 49 - index finger flexing

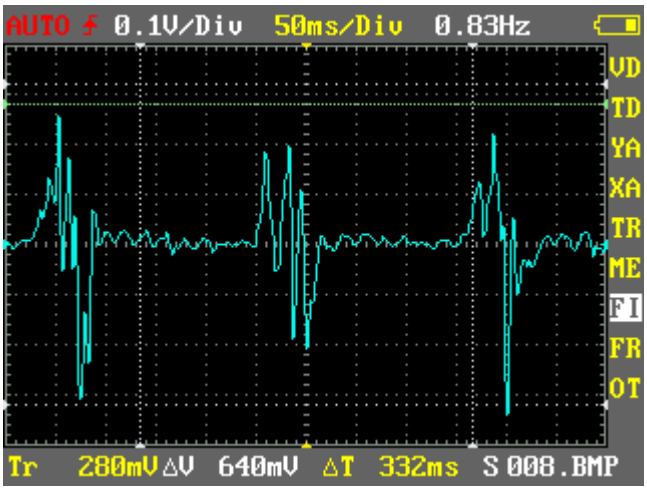

Figure 50 - middle finger flexing

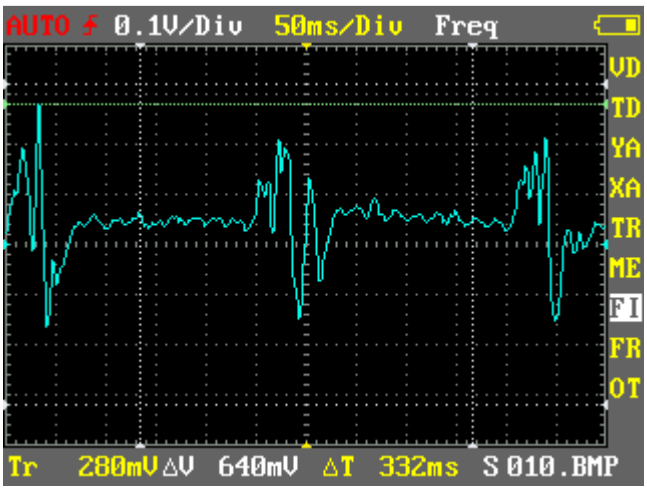

Figure 51 - ring finger flexing

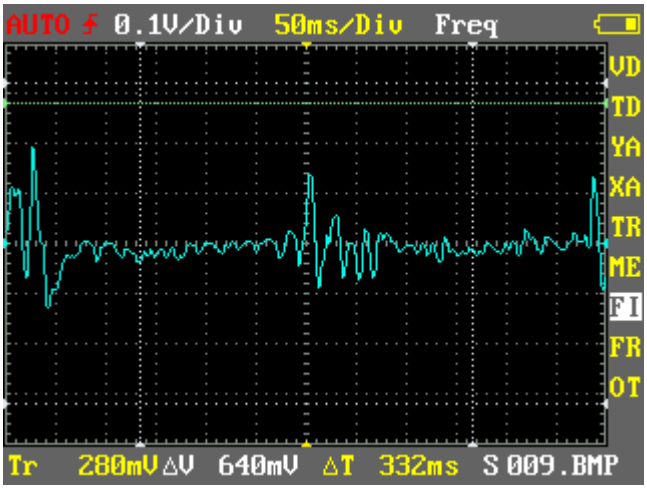

Figure 52 - little finger flexing

The effect of filters can be verified by comparing figures 53 and 54. They represent not an EMG signal, but an ECG signal measured by the device. The reason of measuring this signal was the simplicity of being measured while operating the measurement instruments, as the hands are free because electrodes and wires are placed on chest. Note that the ECG signal is not used in this project, but the device is capable of measuring it and sampling for post processing and future use. Figure 53 represents the signal without filtering and figure 54 represents it filtered.

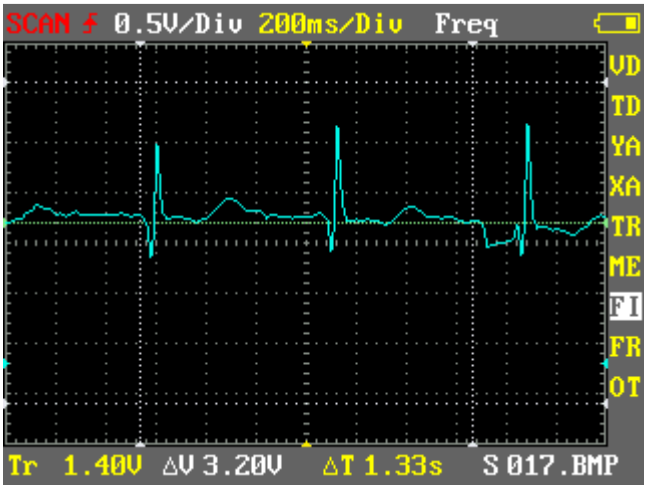

Figure 53 - ECG signal not filtered

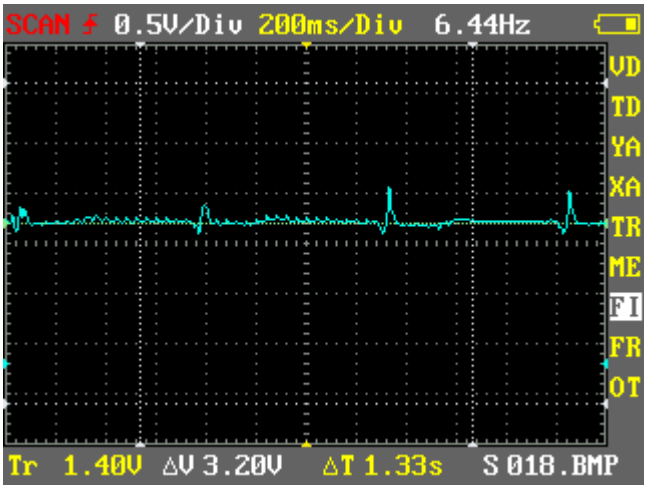

Figure 54 - ECG signal filtered

The signal with the effect of ADC interface circuit, i.e. the addition of a DC offset and high pass filter, is shown in figure 55. The signal is an ECG, again used only to exemplify the application.

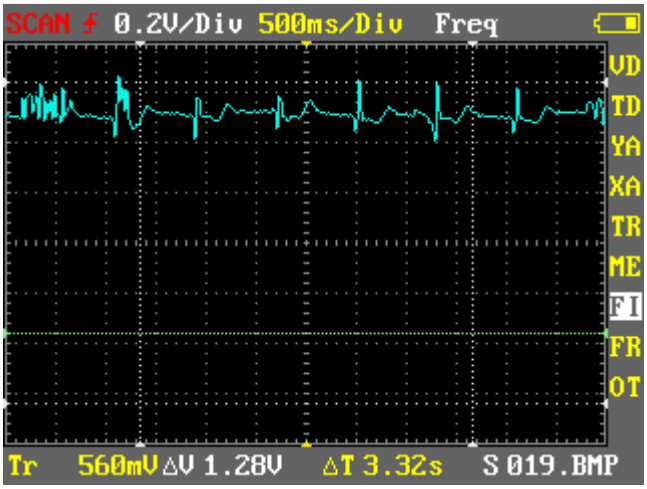

Figure 55 - Signal with DC bias

5.3 – BLUETOOTH COMMUNICATION

After configuring the serial communication settings by programming the microcontroller board and connecting the Bluetooth module, the communication could be verified by using a terminal simulation program, as the *RealTerm*. First it was needed pairing computer with Bluetooth module. A serial port was created and the *COM5* name was assigned to it. On RealTerm software, it was needed to configure the baud rate and select the correct serial port. After that, the transmitted data is shown

on the screen in real time. Figure 56 is a screenshot of the RealTerm software screen while receiving the data from the Bluetooth module.

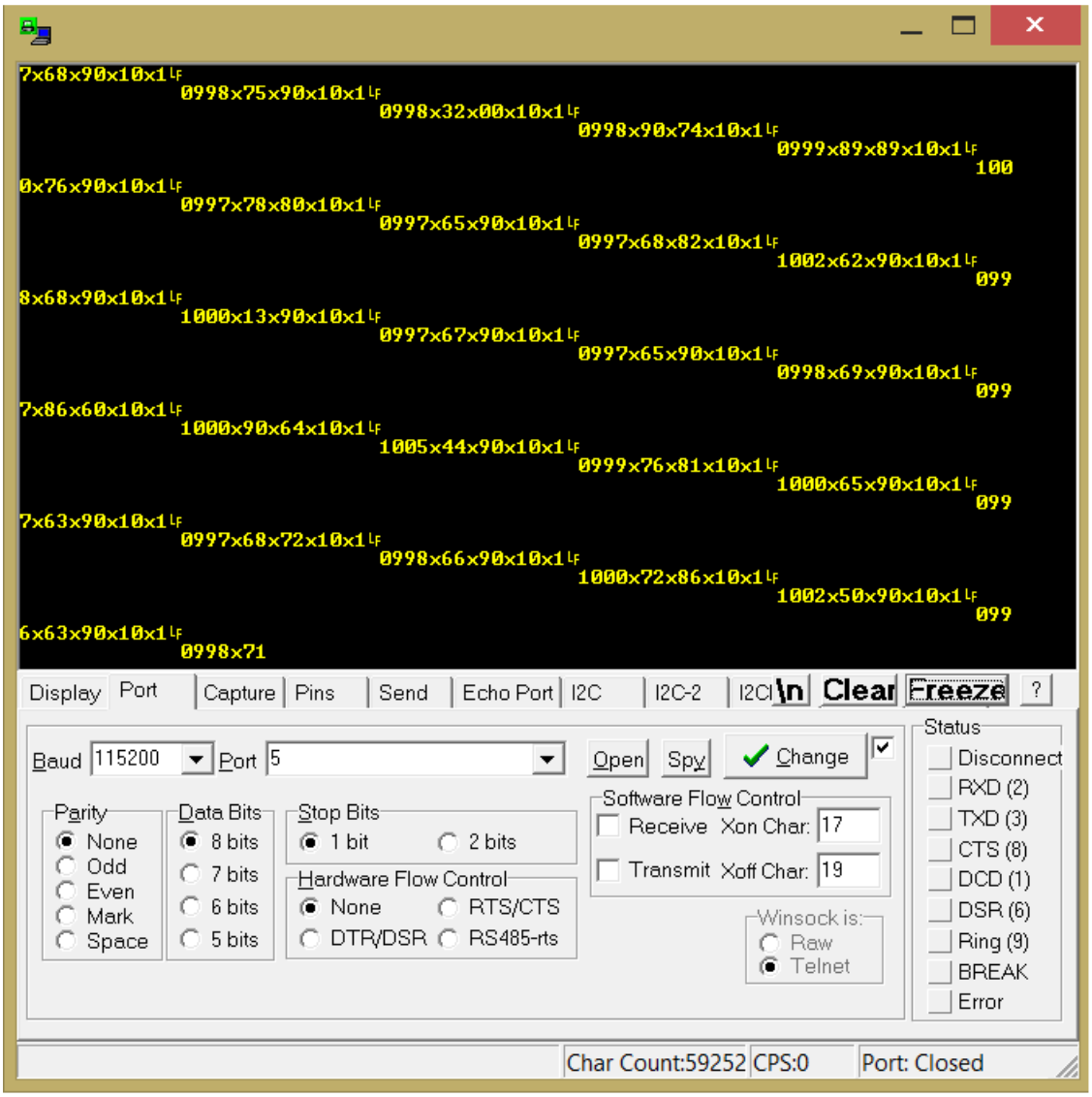

Figure 56 - RealTerm screenshot

5.4 – ACCELEROMETER ANGLE MEASUREMENTS

The accelerometer angle measurements were made by two ways: the analog and the digital one. In the analog version, after positioning the sensor, the voltage was measured on the pin corresponding to each axis. The values were converted to angles using the formulas presented in figure 3. The digital measurements were taken directly of the debugger of the toolchain, using the

variables of the microcontroller board program. The digital angles are calculated by applying the same formulas of figure 3 but the voltage values are results of analog-to-digital conversion.

Positions used for measurements are presented in figure 57 and the corresponding angle measurements are presented in figure 58.

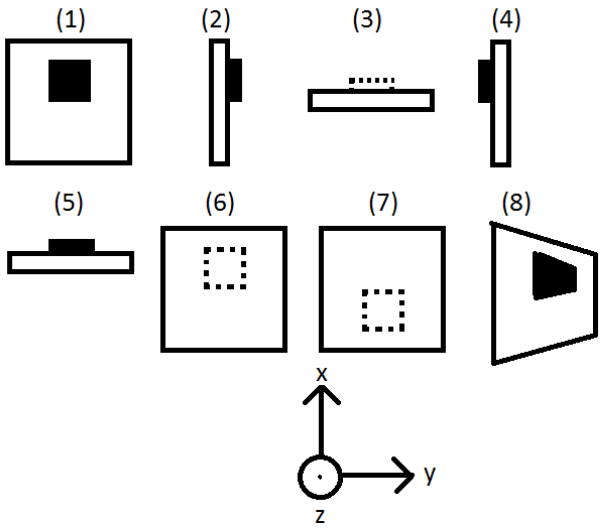

Figure 57 - positions for testing – top view

Angle Measurements									
	Analog			Digital			Error		
Position	θ (angle1)	*Ψ (angle2)*	*Φ (angle3)*	θ (angle1)	*Ψ (angle2)*	*Φ (angle3)*	θ	*Ψ*	*Φ*
1	0°	0°	0°	1.002°	0°	0.808°	1.002°	0°	0.808°
2	46.0416°	0°	-46.0416°	48°	1.216°	-46.04°	1.9584°	1.216°	0.0016°
3	3.0413°	50.2919°	-50.4561°	2.410°	47.3°	-51.21°	0.6313°	2.9919°	0.7539°
4	-47.0368°	2.9963°	-47.1938°	-44.85°	2.52°	-48.42°	2.1868°	0.4763°	1.2262°
5	-1.5282°	-46.0598°	-46.1005°	0.918°	-42.053°	-47.85°	2.4462°	4.0068°	1.7495°
6	0°	3.0665°	3.0665°	0.255°	1.655°	5.06°	0.255°	1.4115°	1.9935°
7	0°	-1.0230°	-1.0230°	0.412°	0°	-2.102°	0.412°	1.023°	1.079°
8	61.3895°	0°	-61.3899°	68.052°	1.23°	-57.865°	6.6625°	1.23°	3.5249°

Figure 58 - angle measurement values

The angles that effectively work for generating sounds are presented on figure 59. They are denoted as angle A and angle B but correspond to twice the absolute value of angles 2 and 1 respectively.

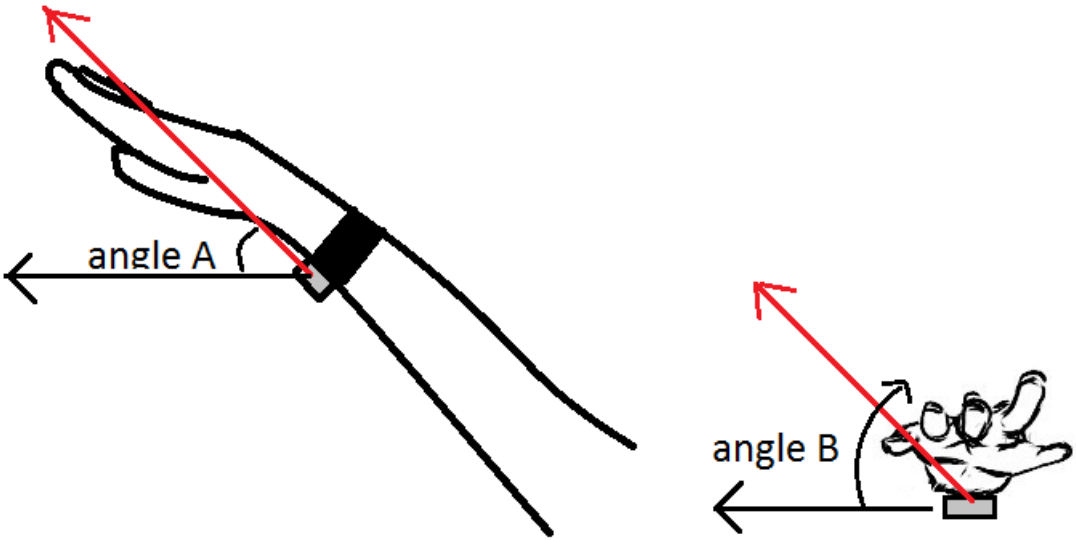

Figure 59 - angles used to generating sound

A simulation of the equipment functioning was also carried out using a 3D model of a hand. This simulation made using the Blender software. Blender is free and open-source 3D computer graphics software, visual effects, 3D models and interactive 3D applications. Python scripting is also possible when using Blender. A script can be written and included in the software alongside other modules. These scripts can be used for controlling a 3D model for example. In this project, the angle measured by the accelerometer is transferred to the computer using Bluetooth communication protocol and a Python program is responsible for receiving and processing this data. The same script, with some adaptation, can be included in Blender and the angle information are used do rotate the 3D model of a human hand. The script can also use the information of muscular activity measured by the electrodes to trigger an animation of the hand opening and closing, accordingly to the movements of the person using the equipment. Figures 60, 61 and 62 show the simulation running on Blender environment. The 3D hand model was downloaded from [26].

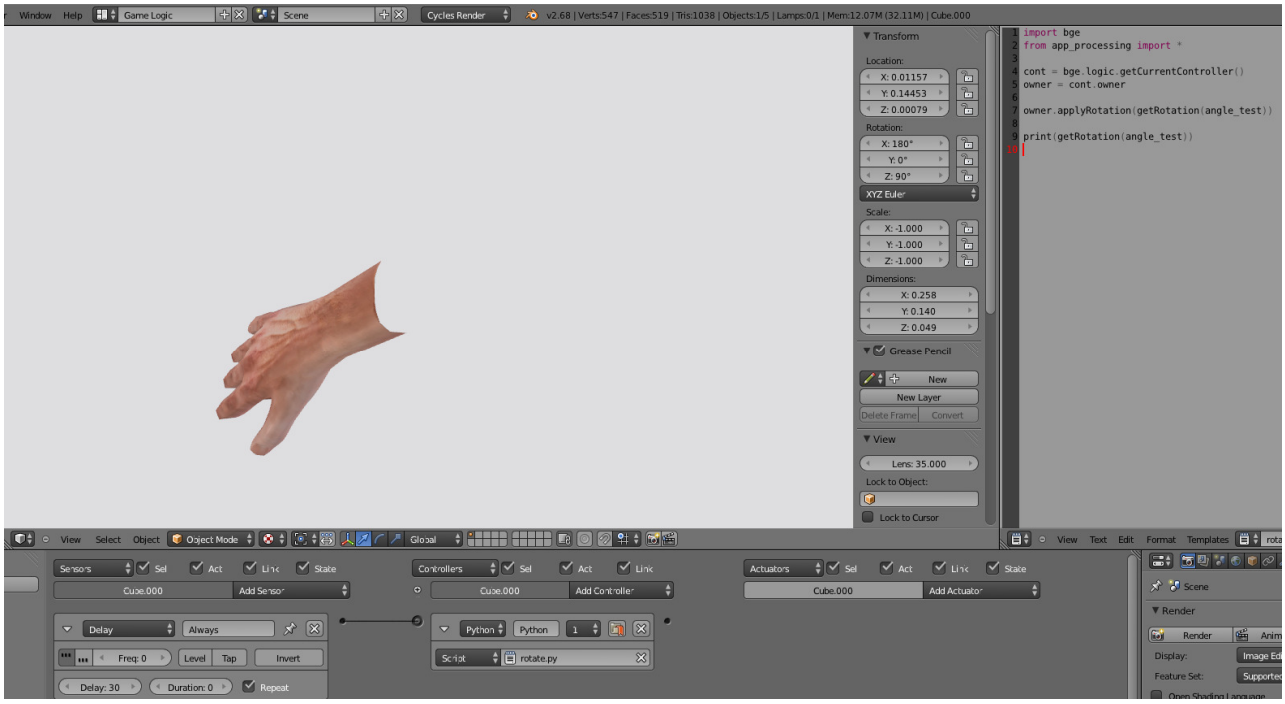

Figure 60 - 3D model simulation - hand down

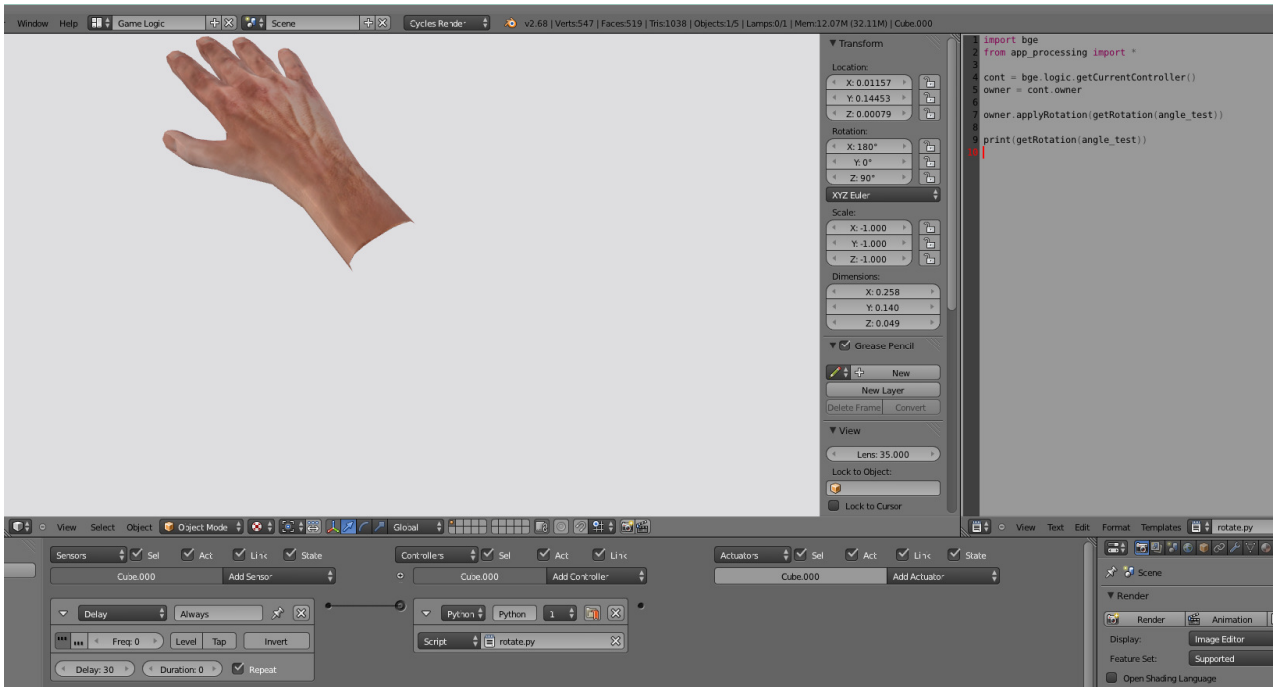

Figure 61 - 3D model simulation - hand up

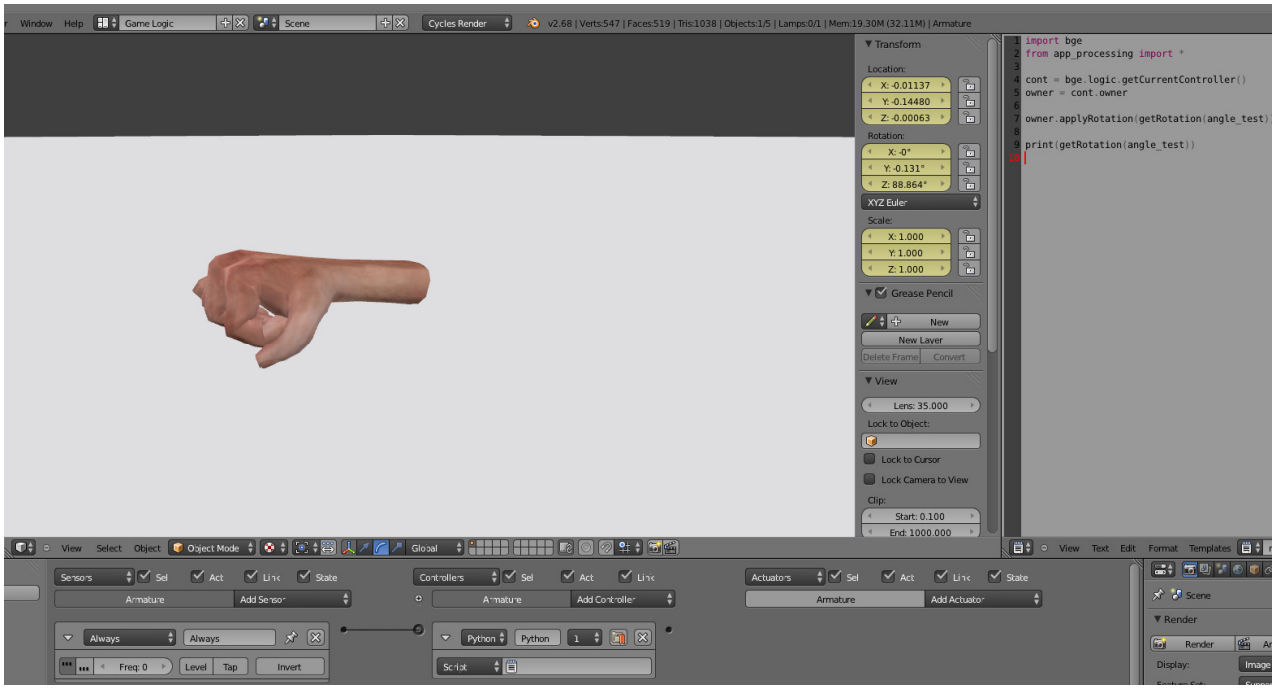

Figure 62 - 3D model simulation - hand closed

The final result of the Python programming was the application program itself. An executable file was created from the scripts, so there is no need of having Python installed on the computer where the application program is running and the equipment is connected. Figures 63 to 67 represent the screens in the order they appear when the application program is running.

Figure 63 - Application Icon

Figure 64 - Application initial screen

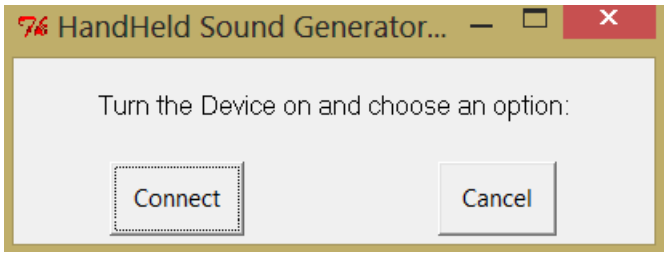

Figure 65 - Connect command screen

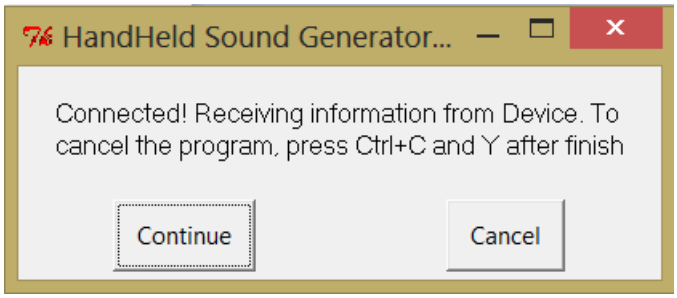

Figure 66 - Information Screen

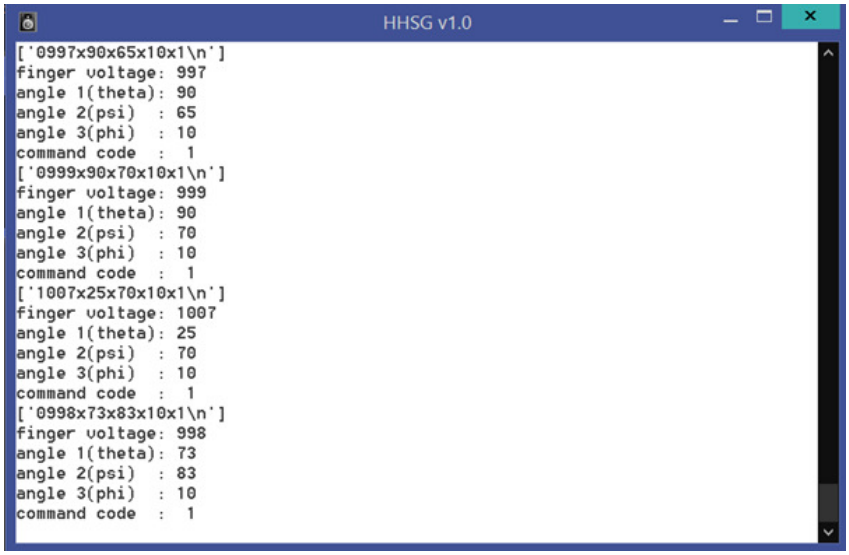

Figure 67 - Received data screen (negative colours)

CHAPTER 6 – DISCUSSIONS

In this chapter, the evaluation of results presented is done and an analysis over factors that could have influenced results will be considered.

The results from section 5.1 were made by the circuit simulation software. The simulation worked as expected since the models used were correct and the circuit diagram was drawn by following the instructions of the datasheet. For the amplifier circuit, the gain was 1000 and the signal successfully was amplified and filtered as expected. Even with a high gain value, the output isn't saturated because the maximum amplitude is smaller than the power supply of amplifiers (3.3V). Considering the DC offset, which has a value of half of the amplifier source, i.e. 3.3V/2 = 1.65V, this value allows the output not being saturated and consequently, all amplitude values of the EMG signal can be read by the microcontroller. If other values of R9 and R5 in circuit of figure 6 are used, the calculation must consider the maximum permitted output value, 3.3V. For passive filters circuit, it can be noticed the effect of each filter by looking at the frequency response diagrams, in figures 41 to 45. At the cut-off frequency, it is expected to each filter has attenuation of 3dB but as the effect of filters is added to the previous filter, the attenuation is also summed. The final response of all filters together has maximum amplitude which is approximately 5dB attenuated in comparison with the raw signal. The attenuation occurs in the respective cut-off frequency of each filter, presenting satisfactory results but it can be noticed the notch filter isn't very sharp, i.e. it may filter some important frequencies near its cut-off frequency. The active filters presented a unitary gain and a frequency response corresponding to the desired values. Considering the circuit uses components with commercial values, the frequency is not much different from the frequency response using exact calculated values of components.

Section 5.2 shows that the EMG signal can be measured successfully. The signal from the electrodes is amplified by the amplifier circuit and the result of a finger motion, for example, can be measured by an oscilloscope. The signal from electrodes couldn't be measured because its amplitude is very small and the oscilloscope is not sensible enough to detect it. In resting position, it can be noticed the interference from 60Hz noise, which is not totally filtered by the passive filters circuit. For the other finger motions, the signal was acquired successfully and, as the electrodes were placed on the middle of arm, the strongest signal measured corresponds to the motion of middle finger. The position of electrodes influences the amplitude of the measured EMG signal. Figures 53 and 54 show the effect of the attenuation caused by the passive filters. The maximum amplitude of the measured signal is about 1.25V for the raw signal and 0.5V for the filtered signal. This attenuation corresponds to a value of -7.95dB which is slightly different from the simulated version (about -7dB). Figure 55 shows the signal with the DC offset applied. The average voltage is about 1V and the expected value

was 1.65V. This means that the circuit must be draining current at some point or the measurement method was incorrect, with the reference terminal placed wrongly.

The Bluetooth communication module worked as expected. The transmitted data is shown in terminal software with the settings chosen correctly. The transmission rate of 115200 baud is enough for this application but if more data would be transmitted per second, the baud rate must be calculated again and changed even in microcontroller board program and in terminal software.

Analysing the angle measurements, it can be noticed in figure 58 the digital and analog measurements present approximately the same values. The difference may be caused by bad calibration of analog to digital converter or rounding error in formulas. All the measured angles are correct and follow the position represented in figure 27. Angle A and angle B results are not show in table because are direct result from a multiplication of the corresponding angle for 2, and an approximation to an integer value.

To successfully integrating Blender simulation software and the equipment's application software, it was needed to use a socket programming approach. By using this technique, it was possible to run both software at same time and see on a computer screen, the 3D model of a hand rotating, following the real motion of the user's arm.

The developed software present in computer worked as expected. The user interface was self-explanatory and had an intuitive way of using. The program could receive the data from the equipment using the Bluetooth receiver from the computer where it was installed process it and apply the angular and muscular activity on the generation of sound. The sound generated was in the form of pulse of square wave, with a certain duration and pitch note.

For effect of comparison, the results obtained from the developed electromyograph weren't compared with measurements of commercial electromyograph equipment. In respect of the application described in this work, the developed equipment had an enough performance, but for some other application, for example diseases diagnosis, it may not provide enough signal information.

CHAPTER 7 – CONCLUSIONS

The equipment developed in this project accomplished the desired aims. Music could be created by sensing the motion of the player and the sound generation was triggered after detection of finger flexion, by an EMG signal amplifier. The inclination of the sensor placed in the wrist in one direction changes the frequency of the sound, in a range of 440 to 880Hz. This range can be changed on the Python program but not when application is running. By tilting the accelerometer in the second way, the duration of generated sound is changed. When the hand is positioned horizontally, the duration is maximum and equal to 1 second. This parameter can also be changed in code. By turning the wrist, the duration of note decreases until no note is played. The sound generation is triggered by flexing the finger and depending on the position of electrodes one of the fingers will generate a stronger signal than the others. The correct positioning of the electrodes is essential for the good functioning and detection of the muscular activity or finger flexing. The Python application runs as expected. The generated sound has a fixed timbre as it is detected by the same function of the Python module every time. The device can be played generating sounds which changes its properties depending on the actions of the player.

Regarding to the hardware chosen for developing this equipment, considerations can be taken: the selected microcontroller board (STM32F4 discovery) presented enough processing capacity for performing the functions of the equipment, for example sampling the EMG signals, converting it to a digital version with enough resolution, acting as a host regarding the wireless communication and calculating all the necessary parameters, especially using floating point units. No delay could be verified in any stage where the microcontroller's board was present. As the idea of the project is to add more complex signal processing functions, the board will continue being used, as its architecture has some special features for DSP, but in the present application, a simpler microcontroller and board could be used, for example a homemade PCB using a Microchip PIC microcontroller, considering the board and microcontroller has the minimum peripherals needed (ADC, UART, Timers).

Still commenting on hardware used, the manufactured EMG amplifier performed well, as they could amplify the biosignals, as demonstrated on results chapter. An improvement which can be done is to change all the electronic components for their SMD package version. This would reduce considerably the dimensions of the PCB and also the size of box for allocating all circuit modules.

The passive filters present on the EMG board were able to filter the amplified EMG signals and transmit it to the microcontroller board but, as show in result sections, the caused an attenuation on the filtered signal in relation to the not-filtered signal. This attenuation is not a problem but is unnecessary as other type of filters can reduce this effect. For changing in this characteristic, the passive filters can be turned off by using the jumpers present on board correctly, and then connecting the EMG board output to the active filters board, which was also designed and shown in this work. As

the gain of active filters is unitary, no attenuation would be verified in the pass band, only where it is really necessary, for example in the cut-off frequency.

Considering the application software, the results obtained, as data transmission and sound generation, overcome expectations. The application could successfully open a connection with the equipment, using Bluetooth protocol, receive information and the close the connection, preventing problems of pairing in future connections. The timbre used by generating the sound came from a native python module and was based on Windows OS system sounds. The sound characteristics could be modified as explained during the work but the timbre can be improved. One possible way of improving the timbre is to interfacing the application program with synthesizer software. Many open source software can be found on internet or even developed using the appropriate programming, signal processing and music concepts. The integration can be based on MIDI protocol, for example.

In general, both hardware and software accomplished their goals successfully and their integration provided the necessary functioning of a new concept of electronic equipment, which has potential use in many areas, for example entertainment and medical.

Future improvements can be made as the hardware is fully functional and tools for programming are already implemented. As the device includes a 4 channel EMG amplifier, more electrodes can be connected to perform a full signal acquisition of signals in different parts of arm. The signals can be sampled and processed with a proper algorithm for a complex detection of which movement is being performed. The third axis of accelerometer can also be added for controlling another property of the generated sound. The sound generation program can be changed to produce more rich sounds in terms of timbre. For last but no less important, passive electrodes can be changed by active electrodes. This would remove the necessity of using conductive gel and discarding the electrodes after using. The usage of information obtained by the equipment can also be expanded to a number of other applications. For example, the motion of user can be used to control a character in video-games or even control a prosthesis used by a person who lost a member, but is still capable of send nervous impulses through muscles. The equipment can also measure other signals with not many alterations on hardware and software. ECG and EEG (electroencephalogram) could also be measured and used in any application.

BIBLIOGRAPHY

[1] FLINDERS UNIVERSITY; Biomedical Engineering [on-line]. [2013-10-19]. Available from: < http://www.flinders.edu.au/science_engineering/csem/disciplines/bme/>.

[2] PEB – UFRJ; Definindo Engenharia Biomédica; Engineering [on-line]. [2013-10-20]. Available from: <http://www.peb.ufrj.br/eb.htm>

[3] KAMEN, GARY. Electromyographic Kinesiology. In Robertson, DGE et al. Research Methods in Biomechanics. Champaign, IL: Human Kinetics Publ., 2004.

[4] R S KHANDPUR; Handbook of Biomedical Instrumentation, Second Edition, Tata McGraw-Hill, 2003.

[5] MARK MAXWELL; 7 Top reasons why music is so important. [on-line]. [2013-07-12]. Available from:<http://ezinearticles.com/?7-Top-Reasons-Why-Music-is-So-Important&id=566580>.

[6] DOTAN; The importance of music in your life. [on-line]. [2013-06-30]. Available from: <http://www.pianoacrossamerica.com/the-importance-of-music-in-your-life/>

[7] MARCELO B. JOAQUIM, M.A. ROMERO; SEL 346 – Comunicação Digital I – lecture notes, 2013.

[8] B. H. SUITS; Physics of Music – Notes, Physics Department, Michigan Technological University, (copyright 1998-2013). [on-line]. [2013-04-02]. Available from: <http://www.phy.mtu.edu/~suits/NoteFreqCalcs.html>

[9] UNKNOWN AUTHOR; Chapter 2 - Accelerometer Theory & Design. [on-line]. [2013-07-12]. Available from: <http://shodhganga.inflibnet.ac.in/bitstream/10603/2272/8/08_chapter%202.pdf>

[10] ANALOG DEVICES; Using accelerometer for Inclination Sensing. [on-line]. [2013-01-16]. Available from: <http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf>

[11] TASSINARY; CACIOPPO (2000); "The Skeletomotor system: surface electromyography". In Cacioppo, John T.; Tassinary, Luois G.; Bernston, Gary G. Handbook of Psychophysiology (Second ed.) Cambridge: Cambridge University Press.

[12] UNIVERSITY OF GLASGOW; EMG - Biosignatures signal processing – Lecture Notes, 2013.

[13] WILLIAM C. SHIEL JR; Electromyogram (EMG). [on-line]. [2013-03-27]. Available from: < http://www.medicinenet.com/electromyogram/article.htm>

[14] JOSÉ HIGINO CORREIA, JOÃO PAULO CARMO; Introdução à Instrumentação Médica – Editora Lidel, 2013.

[15] GEOFREY BROWN; Discovering the STM32 Microcontroller, 2012.

[16] LINEAR TECHNOLOGY; Single Resistor Gain, precision instrumentation amplifier Datasheet. [on-line]. [2013-05-18]. Available from: <http://cds.linear.com/docs/en/datasheet/1167fc.pdf>

[17] 5-30 Watts ECL Series from XPPOWER [on-line]. [2013-06-10]. Available from: <http://www.xppower.com/pdfs/SF_ECL05-30.pdf>.

[18] ANALOG DEVICES; ADXL 335 Datasheet. [on-line]. [2013-02-09]. Available from: <http://www.analog.com/static/imported-files/data_sheets/ADXL335.pdf>

[19] *RC low-pass filter design tool* from OKAWA Electric Design. [on-line]. [2013-05-12]. Available from: <http://sim.okawa-denshi.jp/en/CRlowkeisan.htm>.

[20] *RC high-pass filter design tool* from OKAWA Electric Design. [on-line]. [2013-05-12]. Available from: <http://sim.okawa-denshi.jp/en/CRhikeisan.htm>

[21] *Twin-T Notch filter design tool* from OKAWA Electric Design. [on-line]. [2013-05-12]. Available from: <http://sim.okawa-denshi.jp/en/TwinTCRkeisan.htm>

[22] BRUCE CARTER, RON MANCINI; Op Amps for Everyone, Newnes, 2009.

[23] CHANGPUAK LABORATORY; Active Twin T – Notch Filter Calculator. [on-line]. [2013-06-01]. Available from: <http://www.changpuak.ch/electronics/Active_Notch_Filter.php>.

[24] TEXAS INSTRUMENTS; 3-Terminal Adjustable Regulator. [on-line]. [2013-04-22]. Available from: <http://www.ti.com/lit/ds/symlink/lm317.pdf>.

[25] SainSmart ARM NANO DSO201 Oscilloscope Mini Storage Digital Pocket-Sized Portable Kit. [on-line]. [2013-09-20]. Available from: < http://www.sainsmart.com/sainsmart-arm-nano-dso201-oscilloscope-mini-storage-digital-pocket-sized-portable-kit.html>.

[26] DENNISH2010; Rigged Hand. [on-line]. [2013-10-23]. Available from: <http://www.blendswap.com/blends/view/66039>.

FIGURES REFERENCE

Figure 1 - Accelerometer triple axis inclination calculation

ANALOG DEVICES; Using accelerometer for Inclination Sensing . [on-line]. [2013-01-16]. Available from: < http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf>

Figure 2 - angles for independent inclinantion sensing

ANALOG DEVICES; Using accelerometer for Inclination Sensing. [on-line]. [2013-01-16]. Available from: < http://www.analog.com/static/imported-files/application_notes/AN-1057.pdf>

Figure 3 - floating electrode

[on-line]. [2013-10-25]. Available from: <http://www.global-medical-solutions.com/Skintact-FS-TC110up-Foam-ECG-Gel-Electrode_p_135.html>

Figure 5 - Ideal frequency response of (a)low-pass filter (b) high-pass filter (c)band pass filter (d) band rejection filter

[on-line]. [2013-09-26]. Available from: <-line]. [2013-01-16]. Available from: <http://cnx.org/content/m19836/latest/?collection=col10667/latest>

Figure 6 - data frame

[on-line]. [2013-03-05]. Available from: <http://en.wikipedia.org/wiki/Universal_asynchronous_receiver/transmitter>

Figure 12 - STM32F4 Discovery development board

[on-line]. [2013-01-20]. Available from: <http://www.st.com/web/en/catalog/tools/FM116/SC959/SS1532/PF252419>

Figure 13 - Xppower - ac/dc power supply

[on-line]. [2013-07-09]. Available from: <http://www.xppower.com/pdfs/SF_ECL05-30.pdf>

Figure 15 - ADXL 335 accelerometer board

[on-line]. [2013-01-16]. Available from: <http://www.evilmadscientist.com/2009/basics-updated-using-an-accelerometer-with-an-avr-microcontroller/>

Figure 35 - DSO 201 nano oscilloscope

[on-line]. [2013-10-13]. Available from: <http://www.thebgastation.com/index.php?route=product/product&product_id=344>

Figure 36 - Bluetooth module terminals

[on-line]. [2013-05-01]. Available from: <http://www.fasttech.com/product/1129200-jy-mcu-hc-05-bluetooth-wireless-serial-port-module>

Appendix A

EMG amplifier:

Schematics:

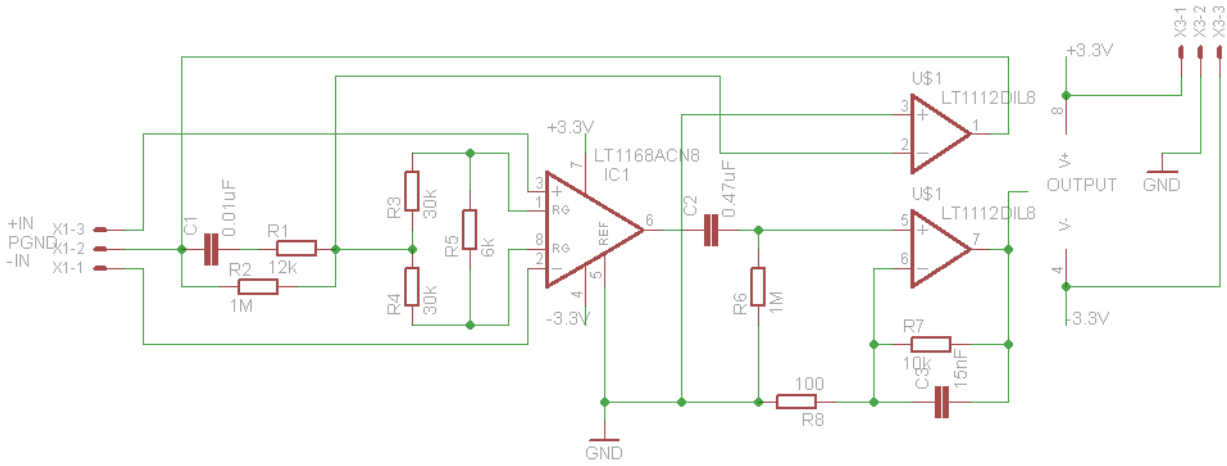

Figure 68 - instrumentation amplifiers and gain amplifier

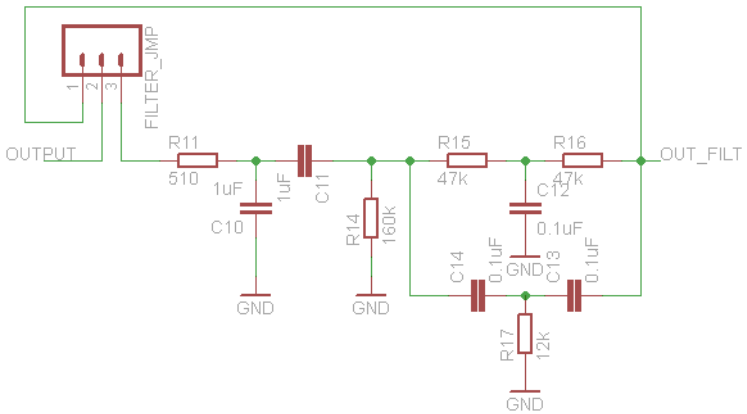

Figure 69 - Filters: LP (300Hz), HP (1Hz) and Notch (50Hz)

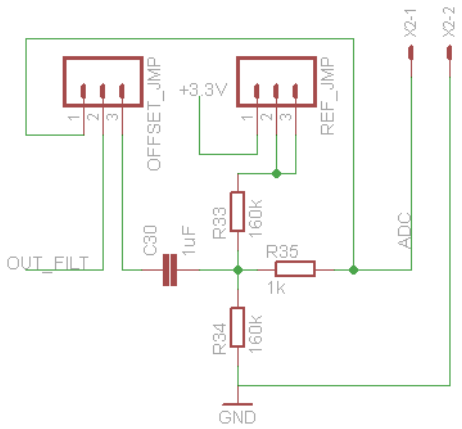

Figure 70 - ADC interface: DC offset and HP filter (1Hz)

Board:

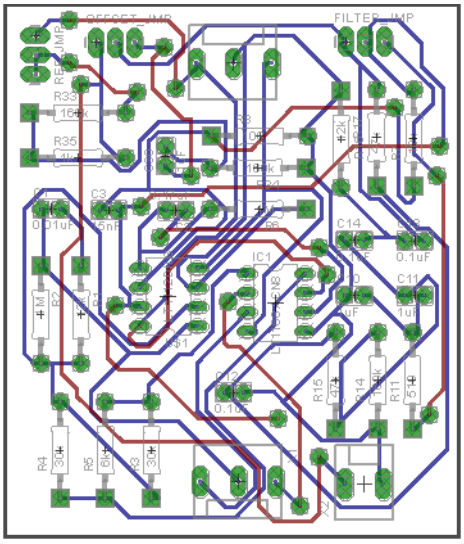

Figure 71 - EMG amplifiers board (original size), red is top layer and blue is bottom layer

DC-DC voltage converter:

Schematics:

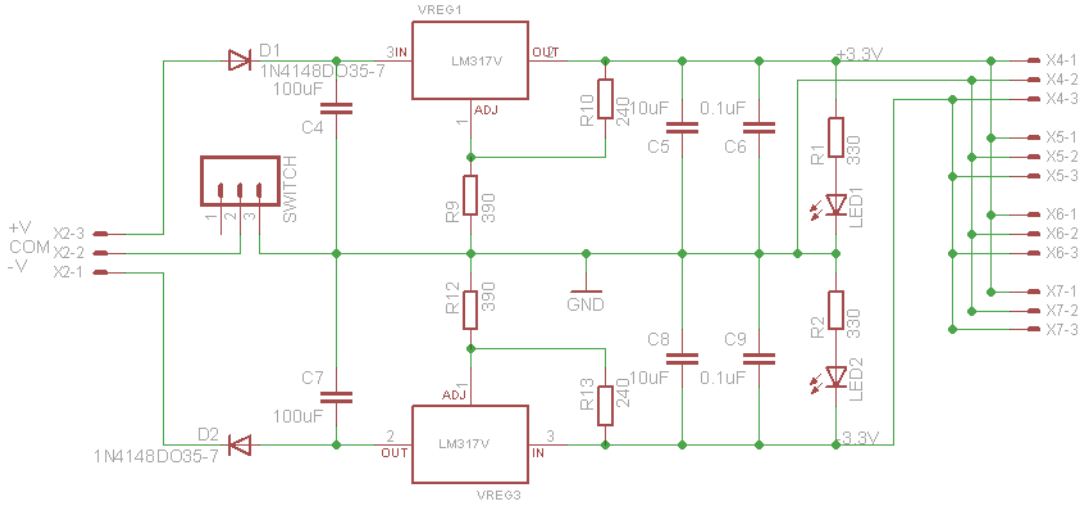

Figure 72 - voltage regulators and LED indicators

Board:

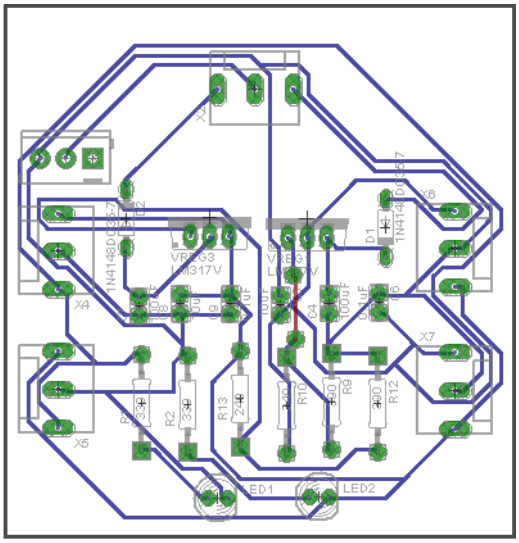

Figure 73- DC-DC voltage converter (original size), red is top layer and blue is bottom layer

Active Filters Board:

Schematics:

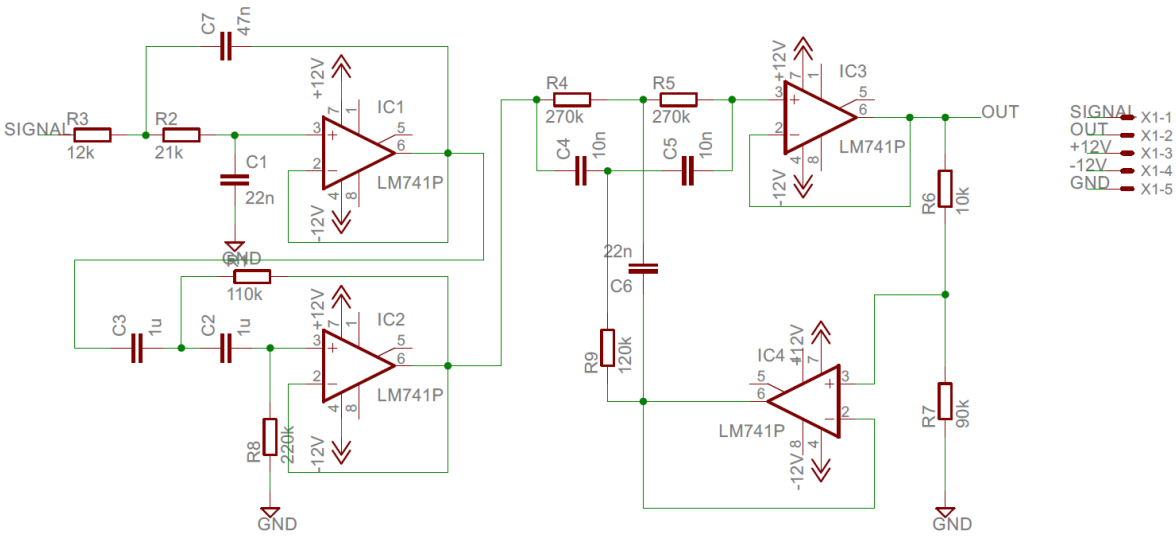

Figure 74 - voltage regulators and LED indicators

Board:

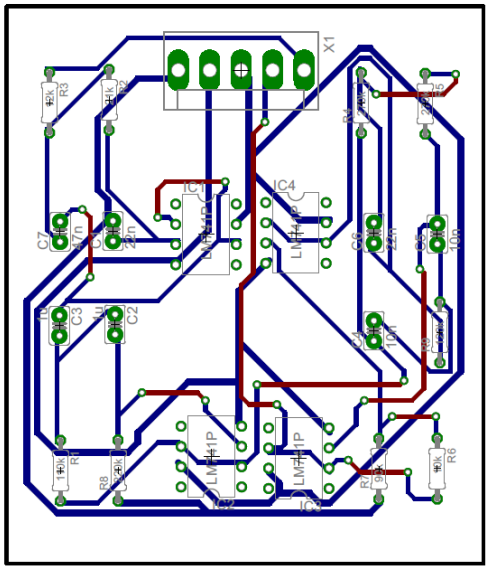

Figure 75- Active Filters Board, red is top layer and blue is bottom layer

Appendix B

Firmware code:

main.c

```
/**
  ******************************************************************************
  * @file    Handheld Sound Generator/main.c
  * @author  Leonardo Mariano Gomes
  * @version V2.0.0
  * @date    30-July-2013
  * @brief   Main program body.
  ******************************************************************************
  * @attention
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2011 STMicroelectronics</center></h2>
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4_discovery.h"
#include <stdio.h>
#include <math.h>
#include "stm32f4xx_it.h"
#include <string.h>
#include <main.h>

/** @addtogroup STM32F4_Discovery_Peripheral_Examples
  * @{
  */

/** @addtogroup ADC_ADC1_DMA
  * @{
  */



/**
  * @brief  Main program
  * @param  None
  * @retval None
  */
int main(void)
{
  /*!< At this stage the microcontroller clock setting is already configured,
       this is done through SystemInit() function which is called from startup
       file (startup_stm32f4xx.s) before to branch to application main.
       To reconfigure the default setting of SystemInit() function, refer to
       system_stm32f4xx.c file

                    */


                    /* Configures User Button */
  STM_EVAL_PBInit(BUTTON_USER, BUTTON_MODE_EXTI);


                    STM_EVAL_LEDInit(LED4);                                         //green
                    STM_EVAL_LEDOn(LED4);


  /* ADC1 configuration ------------------------------------------------------*/
  /*  - Enable peripheral clocks                                 */
  /*  - DMA2_Stream0 channel2 configuration                      */
  /*  - Configure ADC Channel12 pin as analog input              */
  /*  - Configure ADC1 Channel12
                    */

  ADC1_CH12_DMA_Config();
                    init_USART1(BT_BAUD);


                    /* SysTick end of count event each 1ms */
```

```
                if (SysTick_Config(SystemCoreClock / 1000))
 {
 /* Capture error */
   while (1);
 }

 /* Start ADC1 Software Conversion */
 ADC_SoftwareStartConv(ADC1);

                /* Get accelerometer reference position voltages */
                x_ref = acc_ref(4);
                y_ref = acc_ref(5);
                z_ref = acc_ref(6);

                inittim3();



 while (1)
 {

                                //strcpy(buf,create_packet( finger, angle1, angle2, angle3, mode_flag));
                                //UARTSend(buf, sizeof(buf));

                                strcpy(prot,create_protocol(ADC1ConvertedVoltage[0], angleA, angleB, angle3, mode_flag));
                                UARTSend(prot, sizeof(prot));

 }
}

/*
* Sampler function makes the sampling of the input signals and stores it in different arrays
*/


void sampler(void)
{

                                for (i=0;i<n_channels;i++)
                                {
                                                /* convert the ADC value (from 0 to 0xFFF) to a voltage value (from 0V to 3.0V)*/
                                                ADC1ConvertedVoltage[i] = ADC1ConvertedValue[i] *3000/0xFFF;
                                }


                                                ch1_buf[sample] = ADC1ConvertedVoltage[0];
                                                ch2_buf[sample] = ADC1ConvertedVoltage[1];
                                                ch3_buf[sample] = ADC1ConvertedVoltage[2];
                                                ch4_buf[sample] = ADC1ConvertedVoltage[3];

                                                sample++;

                                if (sample>n_samples)
                                {
                                                                sample = 0;
                                }

                                x_voltage = ADC1ConvertedVoltage[4];
                                y_voltage = ADC1ConvertedVoltage[5];
                                z_voltage = ADC1ConvertedVoltage[6];

}

/*
 * Get the reference voltage for the required accelerometer axis
 */

int acc_ref(uint8_t channel)
{
                uint32_t v_ref = 0;

                for (i=0; i<50; i++)
                {
                                ADC1ConvertedVoltage[channel] = ADC1ConvertedValue[channel] *3000/0xFFF;
                                v_ref = ADC1ConvertedVoltage[channel] + v_ref;
                                Delay(1);
                }
                v_ref = (float)v_ref/50;

return v_ref;
}


/*
*calc_angles calculates the 3-axis accelerometer angles
*/
void calc_angles(void)
{

                const float angle_const = 180/3.1415;

                double x_real = 0;
                double y_real = 0;
                double z_real = 0;

                x_real = abs(x_voltage - x_ref);
                y_real = abs(y_voltage - y_ref);
                z_real = abs(z_voltage - z_ref);

                angle1 = atan(x_real/sqrt((y_real*y_real)+(z_real*z_real)));
                angle1 = angle1*angle_const;
```

```
		angle2 = atan(y_real/sqrt((x_real*x_real)+(z_real*z_real)));
		angle2 = angle2*angle_const;

		angle3 = atan((sqrt((x_real*x_real)+(y_real*y_real)))/z_real);
		angle3 = angle3*angle_const;
}

/*
*Adjust the angles of accelerometer to be used in human arm, as arguments of application program
*/
int angle2arm(float angle)
{
		int angle_arm;

		angle_arm = (int)(2*angle);

		if (angle_arm>90)
			angle_arm = 90;


		return angle_arm;
}



/**
* @brief Method to create data packet.
* @param custom parameters
* @return returns packet buffer
*/
char * create_packet(int par1, float par2, float par3, float par4, int par5)
{
		int n;
		char data_end = '\0';
		char title[] = "Data Packet: \r\n";
		static char arr [150];

		n = sprintf		(arr, "\n %s", title);
		n += sprintf	(arr+n, " - Detected Finger: %d\r\n", par1);
 n += sprintf	(arr+n, " - X-axis angle: %f\r\n", par2);
		n += sprintf	(arr+n, " - Y-axis angle: %f\r\n", par3);
 n += sprintf	(arr+n, " - Z-axis angle: %f\r\n", par4);
		n += sprintf	(arr+n, " - Mode: %d\r\n", par5);
		n += sprintf	(arr+n, "%c", data_end);

		return &(arr[0]);
};

/**
* @brief Method to create protocol and data package.
* @param finger emg voltage, angleA(angle1), angleB(angle2), angle3(empty for now), command
* @return returns packet buffer
*/
char * create_protocol(int f_voltage,int angle1,int angle2,int angle3,int command)
{
		int n;
		static char arr [20];
		char data_end = '\n';
// Data for testing communication
// f_voltage = 1234;
//		int angle1 = 45;
//		int angle2 = 38;
		angle3 = 10;
		command = 1;

		if (f_voltage<10)
			n = sprintf		(arr, "000%dx", f_voltage);
		else{
			if (f_voltage<100)
				n = sprintf		(arr, "00%dx", f_voltage);
			else{
				if (f_voltage<1000)
					n = sprintf		(arr, "0%dx", f_voltage);
				else
					n = sprintf		(arr, "%dx", f_voltage);
					}
				}


		if (angle1<10)
			n += sprintf		(arr+n, "0%dx", angle1);
		else
			n += sprintf		(arr+n, "%dx", angle1);


		if (angle2<10)
			n += sprintf		(arr+n, "0%dx", angle2);
		else
			n += sprintf		(arr+n, "%dx", angle2);


		if (angle3<10)
			n += sprintf		(arr+n, "0%dx", angle3);
		else
			n += sprintf		(arr+n, "%dx", angle3);

 n += sprintf	(arr+n, "%d", command);
		n += sprintf	(arr+n, "%c", data_end);

		return &(arr[0]);
```

```
};

/**
* @brief Method to create output stream.
* @param sampled signal buffer
* @return returns output stream buffer
*/
char * create_stream(int *buf)
{
		int n,i;
		char data_end = '\0';
		static char arr[1000];

		for (i=0;i<4;i++)
		{
			sprintf (arr, "%d", buf[i]);
			strncat (arr,arr,10);
		}

		sprintf		(arr, "%c", data_end);
		return &(arr[0]);
};

/**
* @brief test.
* @param custom parameters
* @return returns packet buffer
*/
char * create_test(int par1)
{
		int n;
		char data_end = '\0';
		static char arr [6];

		n = sprintf			(arr, "%d\n", par1);
		//n += sprintf	(arr+n, "%c", data_end);

		return &(arr[0]);
};


/**
* Method that send a string to the UART.
* @param *pcBuffer buffers to be printed.
*@param ulCount the buffer's length
*/
void UARTSend(unsigned char *pucBuffer, unsigned long ulCount)
{
  //
  // Loop while there are more characters to send.
  //
  while(ulCount--)
  {
    while(USART_GetFlagStatus(USART1, USART_FLAG_TXE) == RESET)
    {
    }
    USART_SendData(USART1, (uint8_t) *pucBuffer++);
    /* Loop until the end of transmission */

  }
}

/* This funcion initializes the USART1 peripheral
*
* Arguments: baudrate --> the baudrate at which the USART is
*                                                                                                 supposed to operate
*/
void init_USART1(uint32_t baudrate){

  /* This is a concept that has to do with the libraries provided by ST
  * to make development easier the have made up something similar to
  * classes, called TypeDefs, which actually just define the common
  * parameters that every peripheral needs to work correctly
  *
  * They make our life easier because we don't have to mess around with
  * the low level stuff of setting bits in the correct registers
  */
  GPIO_InitTypeDef GPIO_InitStruct; // this is for the GPIO pins used as TX and RX
  USART_InitTypeDef USART_InitStruct; // this is for the USART1 initilization
  NVIC_InitTypeDef NVIC_InitStructure; // this is used to configure the NVIC (nested vector interrupt controller)

  /* enable APB2 peripheral clock for USART1
  * note that only USART1 and USART6 are connected to APB2
  * the other USARTs are connected to APB1
  */
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);

  /* enable the peripheral clock for the pins used by
  * USART1, PB6 for TX and PB7 for RX
  */
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

  /* This sequence sets up the TX and RX pins
  * so they work correctly with the USART1 peripheral
  */
  GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7; // Pins 6 (TX) and 7 (RX) are used
  GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;                                     // the pins are configured as alternate function so the USART peripheral has access
to them
  GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;                  // this defines the IO speed and has nothing to do with the baudrate!
  GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;                                   // this defines the output type as push pull mode (as opposed to open drain)
  GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;                                 // this activates the pullup resistors on the IO pins
```

```
 GPIO_Init(GPIOB, &GPIO_InitStruct);                                                    // now all the values are passed to the GPIO_Init() function which
sets the GPIO registers

 /* The RX and TX pins are now connected to their AF
 * so that the USART1 can take over control of the
 * pins
 */
 GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_USART1); //
 GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_USART1);

 /* Now the USART_InitStruct is used to define the
 * properties of USART1
 */
 USART_InitStruct.USART_BaudRate = baudrate;                                            // the baudrate is set to the value we passed into this init function
 USART_InitStruct.USART_WordLength = USART_WordLength_8b;// we want the data frame size to be 8 bits (standard)
 USART_InitStruct.USART_StopBits = USART_StopBits_1;                                    // we want 1 stop bit (standard)
 USART_InitStruct.USART_Parity = USART_Parity_No;                    // we don't want a parity bit (standard)
 USART_InitStruct.USART_HardwareFlowControl = USART_HardwareFlowControl_None; // we don't want flow control (standard)
 USART_InitStruct.USART_Mode = USART_Mode_Tx | USART_Mode_Rx; // we want to enable the transmitter and the receiver
 USART_Init(USART1, &USART_InitStruct);                                                 // again all the properties are passed to the
USART_Init function which takes care of all the bit setting


 /* Here the USART1 receive interrupt is enabled
 * and the interrupt controller is configured
 * to jump to the USART1_IRQHandler() function
 * if the USART1 receive interrupt occurs
 */
 USART_ITConfig(USART1, USART_IT_RXNE, ENABLE); // enable the USART1 receive interrupt

 NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;                                      // we want to configure the USART1 interrupts
 NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;// this sets the priority group of the USART1 interrupts
 NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;                  // this sets the subpriority inside the group
 NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;                                        // the USART1 interrupts are globally enabled
 NVIC_Init(&NVIC_InitStructure);                                                                                // the properties are passed to the
NVIC_Init function which takes care of the low level stuff

 // finally this enables the complete USART1 peripheral
 USART_Cmd(USART1, ENABLE);
}

/*- Interruption handler -------------------------------------------------*/

// this is the interrupt request handler (IRQ) for ALL USART1 interrupts
void USART1_IRQHandler(void){

 // check if the USART1 receive interrupt flag was set
 if( USART_GetITStatus(USART1, USART_IT_RXNE) ){

  static uint8_t cnt = 0; // this counter is used to determine the string length
  char t = USART1->DR; // the character from the USART1 data register is saved in t

  /* check if the received character is not the LF character (used to determine end of string)
  * or the if the maximum string length has been been reached
  */
  if( (t != 'n') && (cnt < MAX_STRLEN) ){
   received_string[cnt] = t;
   cnt++;
  }
  else{ // otherwise reset the character counter
   cnt = 0;
  }
 }
}



/* This function is used to transmit a string of characters via
 * the USART specified in USARTx.
 *
 * It takes two arguments: USARTx --> can be any of the USARTs e.g. USART1, USART2 etc.
 *                                                                                (volatile) char *s is the string you want to send
 *
 * Note: The string has to be passed to the function as a pointer because
 *                                the compiler doesn't know the 'string' data type. In standard
 *                                C a string is just an array of characters
 *
 * Note 2: At the moment it takes a volatile char because the received_string variable
 *                                  declared as volatile char --> otherwise the compiler will spit out warnings
 * */
void USART_puts(USART_TypeDef* USARTx, volatile char *s){

                while(*s){
                                // wait until data register is empty
                                while( !(USARTx->SR & 0x00000040) );
                                USART_SendData(USARTx, *s);
                                *s++;
                }
}


/*
        inittim3() configures timer 3 to roll over at a
        the desired sample frequency, and to trigger the ADC every
        time it rolls over.

        Timer 3 is associated with the APB1 bus.  The timer 3 clock frequency
        twice the APB1 bus frequency: 2*42MHz = 84 MHz.  The input parameter
        daccount should be the desired number of timer counts before each adc output
        should be generated.  For example, to generate ADC outputs at a 48k samples/sec,
```

```
        set adccount= (84 MHz)/(48 ksps) = 1750.
  */
  void inittim3(void)
  {
   TIM_TimeBaseInitTypeDef timinfo;
                                NVIC_InitTypeDef  NVIC_InitStructure;

   RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM3, ENABLE);

   TIM_TimeBaseStructInit(&timinfo);
   timinfo.TIM_Period = 168000-1;
   timinfo.TIM_Prescaler = 1;
   timinfo.TIM_ClockDivision = TIM_CKD_DIV1;     // no division, 42 MHz timer clock
   timinfo.TIM_CounterMode = TIM_CounterMode_Up;
   timinfo.TIM_RepetitionCounter = 0;            // Not used for TIM3

   TIM_TimeBaseInit(TIM3, &timinfo);

   /* TIM3 TRGO selection */
   TIM_SelectOutputTrigger(TIM3, TIM_TRGOSource_Update);


                                NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
                                NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 13;
                                NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
                                NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
                                NVIC_Init(&NVIC_InitStructure);
                                TIM_ITConfig(TIM3, TIM_IT_Update, ENABLE) ;



   /* TIM3 enable counter */
   TIM_Cmd(TIM3, ENABLE);
  }

/**
  * @brief  ADC1 channel12 with DMA configuration
  * @param  None
  * @retval None
  */
void ADC1_CH12_DMA_Config(void)
{
  ADC_InitTypeDef       ADC_InitStructure;
  ADC_CommonInitTypeDef ADC_CommonInitStructure;
  DMA_InitTypeDef       DMA_InitStructure;
  GPIO_InitTypeDef      GPIO_InitStructure;

                //NVIC_InitTypeDef nvic;

  /* Enable ADC1, DMA2 and GPIO clocks ****************************************/
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);
                RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOC, ENABLE);
                RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
                RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);

                 /* Enable ADC1 clock so that we can talk to it */
                RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);

//              nvic.NVIC_IRQChannel = DMA2_Stream4_IRQn;       // ADC 1 buffer half full and complete
//              nvic.NVIC_IRQChannelPreemptionPriority = 0;
//              nvic.NVIC_IRQChannelSubPriority = 0;
//              nvic.NVIC_IRQChannelCmd = ENABLE;
//   NVIC_Init(&nvic);
//              DMA_ITConfig(DMA2_Stream4, DMA_IT_TC, ENABLE);



                //DMA_DeInit(DMA2_Stream4);

  /* DMA2 Stream0 channel0 configuration **************************************/
  DMA_InitStructure.DMA_Channel = DMA_Channel_0;
  DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)ADC1_DR_ADDRESS;
  DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&ADC1ConvertedValue;
  DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
  DMA_InitStructure.DMA_BufferSize = n_channels;
  DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
  DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
  DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_HalfWord;
  DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_HalfWord;
  DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
  DMA_InitStructure.DMA_Priority = DMA_Priority_High;
  DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
  DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
  DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
  DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
                DMA_Init(DMA2_Stream4, &DMA_InitStructure);

  DMA_Cmd(DMA2_Stream4, ENABLE);




  /* Configure ADC1 Channel12 pin as analog input *****************************/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4|GPIO_Pin_6|GPIO_Pin_7;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOA, &GPIO_InitStructure);

                  /* Configure ADC1 Channel12 pin as analog input *****************************/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4| GPIO_Pin_5;
```

```
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOC, &GPIO_InitStructure);

                                /* Configure ADC1 Channel12 pin as analog input ******************************/
  GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0| GPIO_Pin_1;
  GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
  GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
  GPIO_Init(GPIOB, &GPIO_InitStructure);



//   /* ADC Common Init **********************************************************/
  ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
  ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div8;
  ADC_CommonInitStructure.ADC_DMAAccessMode = ADC_DMAAccessMode_Disabled;
  ADC_CommonInitStructure.ADC_TwoSamplingDelay = ADC_TwoSamplingDelay_5Cycles;
  ADC_CommonInit(&ADC_CommonInitStructure);

  /* ADC1 Init ****************************************************************/
  ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;
  ADC_InitStructure.ADC_ScanConvMode = ENABLE;
  ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
  ADC_InitStructure.ADC_ExternalTrigConvEdge = ADC_ExternalTrigConvEdge_Rising;
                ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConv_T3_TRGO;
  ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
  ADC_InitStructure.ADC_NbrOfConversion = n_channels;
  ADC_Init(ADC1, &ADC_InitStructure);

  /* ADC1 regular channel12 configuration *************************************/
  ADC_RegularChannelConfig(ADC1, ADC_Channel_4, 1, ADC_SampleTime_3Cycles);
                ADC_RegularChannelConfig(ADC1, ADC_Channel_7, 2, ADC_SampleTime_3Cycles);
                ADC_RegularChannelConfig(ADC1, ADC_Channel_6, 3, ADC_SampleTime_3Cycles);
                ADC_RegularChannelConfig(ADC1, ADC_Channel_15, 4, ADC_SampleTime_3Cycles);
                ADC_RegularChannelConfig(ADC1, ADC_Channel_14, 5, ADC_SampleTime_56Cycles);
                ADC_RegularChannelConfig(ADC1, ADC_Channel_9, 6, ADC_SampleTime_56Cycles);
                ADC_RegularChannelConfig(ADC1, ADC_Channel_8, 7, ADC_SampleTime_56Cycles);

 /* Enable DMA request after last transfer (Single-ADC mode) */
  ADC_DMARequestAfterLastTransferCmd(ADC1, ENABLE);


                ADC_DiscModeChannelCountConfig(ADC1,7);
  ADC_DiscModeCmd(ADC1,ENABLE);
                //ADC_ContinuousModeCmd(ADC1, ENABLE);

  /* Enable ADC1 DMA */
  ADC_DMACmd(ADC1, ENABLE);

  /* Enable ADC1 */
  ADC_Cmd(ADC1, ENABLE);
}

//  void DMA2_Stream4_IRQHandler(void)
// {
//                 if(DMA_GetITStatus(DMA2_Stream4, DMA_IT_TC) == RESET)
//                                 {
//
//                                                 /* Toggle LED4 */
//                                                 //STM_EVAL_LEDToggle(LED4);
//                                                                                 sampler();
//                                                 calc_angles();


//                                                 /* Clear the Right Button EXTI line pending bit */
//                                                 DMA_ClearITPendingBit(DMA2_Stream4, DMA_IT_TC);
//                                 }
//
// }




void TIM3_IRQHandler(void)
{
  TIM_ClearITPendingBit(TIM3, TIM_IT_Update);
  STM_EVAL_LEDToggle(LED4);
                sampler();
                calc_angles();
                angleA = angle2arm(angle2);
                angleB = angle2arm(angle1);

}



void Delay(__IO uint32_t nTime)
{
  TimingDelay = nTime;

  while(TimingDelay != 0);
}

/**
  * @brief  Decrements the TimingDelay variable.
  * @param  None
  * @retval None
  */
void TimingDelay_Decrement(void)
```

```
{
  if (TimingDelay != 0x00)
  {
    TimingDelay--;
  }
}

#ifdef  USE_FULL_ASSERT

/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t* file, uint32_t line)
{
  /* User can add his own implementation to report the file name and line number,
     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */

  /* Infinite loop */
  while (1)
  {
  }
}
#endif

/**
  * @}
  */

/**
  * @}
  */

/******************* (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE****/
```

main.h

```
#ifndef __MAIN_H
#define __MAIN_H

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
#define ADC1_DR_ADDRESS     ((uint32_t)0x4001204C)

#define n_samples                                                                       1000
#define n_channels                                                              7

#define BT_BAUD 115200
#define MAX_STRLEN 20 // this is the maximum string length of our string in characters


/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/
void                    ADC1_CH12_DMA_Config(void);

void                    inittim3(void);

void                    sampler(void);

void                    calc_angles(void);

int                     acc_ref(uint8_t channel);

void                    Delay(__IO uint32_t nTime);

void                    EXTILine1_Config(void);

void            init_USART1(uint32_t baudrate);

int                     angle2arm(float angle);



/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

/* You can monitor the converted value by adding the variable "ADC1ConvertedValue"
   to the debugger watch window */
__IO uint16_t ADC1ConvertedValue[n_channels*1000] ;
__IO uint32_t ADC1ConvertedVoltage[n_channels] ;

/* Buffers of sampled EMG signals */
__IO uint32_t ch1_buf[n_samples];
__IO uint32_t ch2_buf[n_samples];
__IO uint32_t ch3_buf[n_samples];
__IO uint32_t ch4_buf[n_samples];

/* Variables for Accelerometer output */
__IO uint32_t x_voltage = 0;
__IO uint32_t y_voltage = 0;
__IO uint32_t z_voltage = 0;

__IO uint32_t x_ref = 0;
```

```
__IO uint32_t y_ref = 0;
__IO uint32_t z_ref = 0;

__IO double_t angle1 = 0; //theta
__IO double_t angle2 = 0;     //psi
__IO double_t angle3 = 0;     //phi

__IO uint32_t angleA = 0;
__IO uint32_t angleB = 0;


/* Variables for UART - Bluetooth */
char * create_packet(int par1, float par2, float par3, float par4, int par5);
char * create_stream(int *buf);
char * create_test(int par1);
char * create_protocol(int f_voltage,int angle1,int angle2,int angle3,int command);

/* Delay variables----------------------------------------------------------*/
static __IO uint32_t TimingDelay;
void UARTSend( unsigned char * pucBuffer, unsigned long ulCount);
void USART_puts(USART_TypeDef* USARTx, volatile char *s);
volatile char received_string[MAX_STRLEN+1]; // this will hold the recieved string

unsigned char  prot [16];
unsigned char  buf [133];
char stream[1000];

int                 n;
int                 finger = 1;
// float x_angle = 49.5632;
// float y_angle = 0.1245;
// float z_angle = -9.6541;
int mode_flag = 3;


int i = 0;
int sample = 0;


#endif
```

Stm32f4xx_it.c (interrupt routines)

```
/**
  ******************************************************************************
  * @file    ADC3_DMA/stm32f4xx_it.c
  * @author  MCD Application Team
  * @version V1.0.0
  * @date    19-September-2011
  * @brief   Main Interrupt Service Routines.
  *          This file provides template for all exceptions handler and
  *          peripherals interrupt service routine.
  ******************************************************************************
  * @attention
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2011 STMicroelectronics</center></h2>
  ******************************************************************************
  */

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx_it.h"
#include "stm32f4_discovery.h"

/** @addtogroup STM32F4_Discovery_Peripheral_Examples
  * @{
  */

/** @addtogroup ADC_ADC3_DMA
  * @{
  */

/* Private typedef -----------------------------------------------------------*/
/* Private define ------------------------------------------------------------*/
/* Private macro -------------------------------------------------------------*/
/* Private variables ---------------------------------------------------------*/

void TimingDelay_Decrement(void);

/* Private function prototypes -----------------------------------------------*/
/* Private functions ---------------------------------------------------------*/

/******************************************************************************/
/*           Cortex-M4 Processor Exceptions Handlers                          */
/******************************************************************************/

/**
  * @brief   This function handles NMI exception.
  * @param  None
  * @retval None
```

```
  */
void NMI_Handler(void)
{
}

/**
  * @brief  This function handles Hard Fault exception.
  * @param  None
  * @retval None
  */
void HardFault_Handler(void)
{
  /* Go to infinite loop when Hard Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Memory Manage exception.
  * @param  None
  * @retval None
  */
void MemManage_Handler(void)
{
  /* Go to infinite loop when Memory Manage exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Bus Fault exception.
  * @param  None
  * @retval None
  */
void BusFault_Handler(void)
{
  /* Go to infinite loop when Bus Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles Usage Fault exception.
  * @param  None
  * @retval None
  */
void UsageFault_Handler(void)
{
  /* Go to infinite loop when Usage Fault exception occurs */
  while (1)
  {
  }
}

/**
  * @brief  This function handles SVCall exception.
  * @param  None
  * @retval None
  */
void SVC_Handler(void)
{
}

/**
  * @brief  This function handles Debug Monitor exception.
  * @param  None
  * @retval None
  */
void DebugMon_Handler(void)
{
}

/**
  * @brief  This function handles PendSVC exception.
  * @param  None
  * @retval None
  */
void PendSV_Handler(void)
{
}

/**
  * @brief  This function handles External line 0 interrupt request.
  * @param  None
  * @retval None
  */
void EXTI0_IRQHandler(void)
{
  if(EXTI_GetITStatus(USER_BUTTON_EXTI_LINE) != RESET)
  {

                                    /* Toggle LED4 */
    STM_EVAL_LEDToggle(LED4);


    /* Clear the Right Button EXTI line pending bit */
    EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE);
  }
```

```
}


/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
void SysTick_Handler(void)
{
                TimingDelay_Decrement();
}

/******************************************************************************/
/*                 STM32F4xx Peripherals Interrupt Handlers                   */
/*  Add here the Interrupt Handler for the used peripheral(s) (PPP), for the  */
/*  available peripheral interrupt handler's name please refer to the startup */
/*  file (startup_stm32f4xx.s).                                               */
/******************************************************************************/

/**
  * @brief  This function handles PPP interrupt request.
  * @param  None
  * @retval None
  */
/*void PPP_IRQHandler(void)
{
}*/

/**
  * @}
  */


/************************ (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE****/
```

stm32f4xx_it.h (interrupt routines header)

```
/**
  ******************************************************************************
  * @file    ADC3_DMA/stm32f4xx_it.h
  * @author  MCD Application Team
  * @version V1.0.0
  * @date    19-September-2011
  * @brief   This file contains the headers of the interrupt handlers.
  ******************************************************************************
  * @attention
  *
  * THE PRESENT FIRMWARE WHICH IS FOR GUIDANCE ONLY AIMS AT PROVIDING CUSTOMERS
  * WITH CODING INFORMATION REGARDING THEIR PRODUCTS IN ORDER FOR THEM TO SAVE
  * TIME. AS A RESULT, STMICROELECTRONICS SHALL NOT BE HELD LIABLE FOR ANY
  * DIRECT, INDIRECT OR CONSEQUENTIAL DAMAGES WITH RESPECT TO ANY CLAIMS ARISING
  * FROM THE CONTENT OF SUCH FIRMWARE AND/OR THE USE MADE BY CUSTOMERS OF THE
  * CODING INFORMATION CONTAINED HEREIN IN CONNECTION WITH THEIR PRODUCTS.
  *
  * <h2><center>&copy; COPYRIGHT 2011 STMicroelectronics</center></h2>
  ******************************************************************************
  */

/* Define to prevent recursive inclusion -------------------------------------*/
#ifndef __STM32F4xx_IT_H
#define __STM32F4xx_IT_H

#ifdef __cplusplus
 extern "C" {
#endif

/* Includes ------------------------------------------------------------------*/
#include "stm32f4xx.h"

/* Exported types ------------------------------------------------------------*/
/* Exported constants --------------------------------------------------------*/
/* Exported macro ------------------------------------------------------------*/
/* Exported functions ------------------------------------------------------- */

void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);
void SysTick_Handler(void);

#ifdef __cplusplus
}
#endif

#endif /* __STM32F4xx_IT_H */

/************************ (C) COPYRIGHT 2011 STMicroelectronics *****END OF FILE****/
```

Application Sotware code:

app_gui.py

```
""" This script is responsible to generate the graphical interface"""
""" Improvements can be done in future to make interface more friendly and interative"""

import easygui as eg
import sys
import os

image   = "description.gif"
msg     = " "
title           = "HandHeld Sound Generator v1.0"
choices = ["Start","Quit"]
reply   = eg.buttonbox(msg,title,image=image,choices=choices)

if reply == "Start":
        pass
else:
        sys.exit(0)

msg             = "Turn the Device on and choose an option:"
choices = ["Connect","Cancel"]
reply           = eg.buttonbox(msg,title,choices=choices)

if reply == "Connect":
        if eg.ccbox("Connected! Receiving information from Device. To cancel the program, press Ctrl+C and Y after finish", title):
                pass
        else:
                eg.msgbox("Device Disconnected. Program Stopped",title, ok_button="Finish")
                sys.exit(0)
else:
        sys.exit(0)

try:
        import app_main
except KeyboardInterrupt:
        eg.msgbox("Program Stopped",title, ok_button="Finish")
        sys.exit(0)
```

app_main.py

```
""" Main program  """
""" This program is responsible to execute and call the other files, assign the main variables and play sound"""

import app_serial
import app_manage_data
#import app_sound
from app_processing import *
import sys
import thread
import winsound

def play(a1,a2,a3,c):
        #c = app_sound.CreateSound()                                    # These two lines can generate sound but presented bad functioning
        #c.playsound()                                                  # while working on Windows environment. An
option was to use winsound to generate sounds.
        if convert_finger(f_voltage)==1:
                winsound.Beep(int(a2f(angle1)*octave(angle3)), int(a2l(angle2)*1000)) # Generate sound based on angle values of accelerometer

a = app_serial.SerialData()                                             # Configure and open serial port

try:
        while True:

                data = []
                values = []

                data = a.read_data(1)                                   # read one line per time
                #data = ["2000x10x89x10x1\n"]                           # array for testing
                print data

                b = app_manage_data.Manage(data)
                b.all_indices()
                values = b.split_data()                                 # Split the received string


                f_voltage       = int(float(values[0]))
                angle1                  = int(float(values[1]))
                angle2                  = int(float(values[2]))
                angle3                  = int(float(values[3]))
                command                 = int(float(values[4]))


                print "finger voltage:",f_voltage
                print "angle 1(theta):",angle1
                print "angle 2(psi)  :",angle2
                print "angle 3(phi)  :",angle3
```

```
                                        print "command code  : ",command

                                        # This thread is used to make possble to generate sound while receiving data from serial port
                                        # without this thread, the sound generation stopped abruptly after a while because the computer
                                        # couldnt handle the incomming information properly.

                                        thread.start_new(play, (angle1,angle2,angle3,command,))

except KeyboardInterrupt:
        a.close()                                       # Close serial to make possible a future connection without needing to reboot the device
```

app_serial.py

```
""" Class for receiving data """
""" This class is responsible to configure the serial port, open connection, receive data end close connection"""

import serial
import time
import numpy
import sys

# Based on the example found in: http://www.roman10.net/serial-port-communication-in-python/

#initialization and open the port
#possible timeout values:
#    1. None: wait forever, block call
#    2. 0: non-blocking mode, return immediately
#    3. x, x is bigger than 0, float allowed, timeout block call

class SerialData:
        def __init__(self):
                self.ser = ser = serial.Serial()
                self.ser.port = "COM5"                                                  # This value must be modified if bluetooth is
connected to another port
                self.ser.baudrate = 115200                                              # This values must be modified if transmission speed is changed on
the other side of connection
                self.ser.bytesize = serial.EIGHTBITS            #number of bits per bytes
                self.ser.parity = serial.PARITY_NONE            #set parity check: no parity
                self.ser.stopbits = serial.STOPBITS_ONE    #number of stop bits
                #self.ser.timeout = None                                #block read
                #self.ser.timeout = 0                           #non-block read
                #self.ser.timeout = 10000                       #timeout block read
                self.ser.xonxoff = False                                #disable software flow control
                self.ser.rtscts = False                                 #disable hardware (RTS/CTS) flow control
                self.ser.dsrdtr = False                                 #disable hardware (DSR/DTR) flow control
                self.ser.writeTimeout = 2                               #timeout for write
                #print"Serial Configured"

                self.ser.open()
                print "Serial Succesfully Connected"


        def read_data(self,lines):
                """ This method reads the incoming data and creates a list with chosen size"""

                self.lines = lines                      # This parameter corresponds to desired length of list
                datalist = []                           # Empty data list which will store the incoming data


                self.ser.flushInput()           #flush input buffer, discarding all its contents
                time.sleep(0.1)                         #give the serial port sometime to receive the data

                for i in range(0,lines+1):
                        data = self.ser.readline()
                        datalist.append(data)#concatenate the list
                        #print "Read Data: " + data

                del datalist[0]                         # delete first item which sometimes is received incorrectly

                #self.ser.close()
                #print "Serial Succesfully Closed"
                return datalist

        def close(self):
                self.ser.close()

        def __del__(self):
                self.ser.close()
                print "Serial Succesfully Closed"

# Lines for script testing

#data = []
#a = SerialData()

#while True:
        #data = a.read_data(1)
        #print data
        #print len(data[0])
```

app_manage_data.py

```
"""Data handling class """
"""This class will split the values of the received string accordingly to the protocol """

#data = "1234x45x38x10x1"
#print data

class Manage:

	def __init__(self,data):
		self.data = data[0]
		self.indices = []
		self.f_voltage  = 0
		self.angle1     = 0
		self.angle2     = 0
		self.angle3     = 0
		self.command = 0

	def all_indices(self, value = "x"):                                    # "x" is the protocol values separator
	   idx = -1
	   while True:                                                                                    # The index of each "x" is identified
in the input string
	      try:
	         idx = self.data.index(value, idx+1)
	         self.indices.append(idx)
	      except ValueError:
	         break

	def split_data(self):                                                                  # values in the input string are separeted following
the index of each separator

		self.f_voltage  = self.data[0:self.indices[0]]
		self.angle1     = self.data[(self.indices[0])+1:self.indices[1]]
		self.angle2     = self.data[(self.indices[1])+1:self.indices[2]]
		self.angle3     = self.data[(self.indices[2])+1:self.indices[3]]
		self.command = self.data[(self.indices[3])+1:len(self.data)]

		return self.f_voltage, self.angle1, self.angle2, self.angle3, self. command

# lines for script testing

#values = []
#a = Manage(data)
#a.all_indices()
#values = a.split_data()
#print values[0]
```

app_processing.py

```
"""Functions to convert data received to apropriate values"""

def convert_finger(f_voltage,base = 950, threshold = 400):
	'''
	calculates the derivative of a series of values to determine
	when to set a trigger
	'''
	if f_voltage > base + threshold or f_voltage < base - threshold:
		trigger = 1
	else:
		trigger = 0
	return trigger


def a2f(angle, f0 = 440):
	'''
	Converts the input angle to a frequency value linearly
	-90 < angle < 90 (degrees)
	'''
	angle		= abs(angle)                                                  # make all angle input positive
	semitones	= 12.                                                                  # number of semitones in range
	angle_lim	= 90.                                                                  # max angle permited
	n		= int(angle*(semitones/angle_lim))          # linear conversion of angle to semitones
	freq		= int(f0*(pow(2,n*0.08333333)))             # semitones to frequency conversion using 440Hz as base

	return freq

def a2l(angle,max_len = 1, max_angle = 90.):
	'''
	Converts the input angle to duration of note in seconds
	'''
	angle		= abs(angle)
	length		= max_len - ((max_len*angle)/max_angle)

	return length

def a2v(angle,max_angle = 90.):
	'''
	Converts the input angle to volume [0 100]
	'''
	angle		= abs(angle)
	volume		= int(((max_angle - angle)/max_angle)*100)

	return volume

def octave(angle):
	'''
	Changes base frequency Octave accordingly to input angle
	'''
```

```
		angle				= abs(angle)
		if angle < 22:
				factor = 1
		else:
				if angle < 45:
						factor = 2
				else:
						if angle < 67:
								factor = 3
						else:
								factor = 4
		return factor

# Lines for script testing

#for i in range(91):
#		print("angle:", i,"freq:", angle2freq(i))

#for i in range(91):
#		print(i," --> ",(a2v(i)))
```

app_sound.py

```
""" This script is used to generate sound using Python Musical and Pygame modules """

# Python-Musical Module ps://code.google.com/p/python-musical/
import source
import playback
import effect
import encode

import math


class CreateSound:
		def __init__(self):
				self.freq			= 0
				self.length			= 0
				self.data			= []

		def playsound(self,freq=440,length=1,volume = 100,command = 1):
				self.freq			= freq
				self.length	= length
				self.command	= command
				self.volume	= volume

				self.volume	=		0.108268*(math.exp(self.volume*0.02325944) - 1)
				# logarithmic response to compensate human loudness hearing

				if command == 1:
						self.data = source.sine(self.freq,self.length)
				if command == 2:
						self.data = source.square(self.freq,self.length)
				if command == 3:
						self.data = source.sawtooth(self.freq,self.length)

				self.data = self.volume*self.data
				self.data = encode.as_uint16(self.data)
				playback.play(self.data)
				#print self.data
				#winsound.PlaySound(self.data ,winsound.SND_MEMORY)


#a = CreateSound()
#data = []
#data = a.playsound(440,0.4,100,2)
#data = a.playsound(440,0.4,10,2)
#data = a.playsound(440,0.4,25,2)
#data = a.playsound(440,0.4,100,2)
#data = a.playsound(440,0.4,0,2)
#for i in range(0,100,5):
		#data = a.playsound(440,0.5,i,3)
```

app_setup.py

```
""" This script is used to generate the EXE file from python script"""

from distutils.core import setup
import py2exe

setup(console=['app_gui.py'])
```