8-2014

Electroencephalography(EEG) Data Collection and Processing through Machine Learning

Jayshree Desai
University of Arkansas, Fayetteville

Electroencephalography (EEG) Data Collection and Processing through Machine Learning

Electroencephalography (EEG) Data Collection and Processing through Machine Learning

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Electrical Engineering

By

Jayshree Desai
Birla Institute of Technology and Science Pilani-Dubai
Bachelor of Engineering in Electronics and Communication Engineering, 2012

August 2014
University of Arkansas

This thesis is approved for recommendation to the Graduate Council.

_____

Dr. Jingxian Wu
Thesis Director

_____ _____

Dr. Baohua Li
Committee Member

Dr. Jing Yang
Committee Member

Abstract

Machine learning methods are an excellent way for understanding the neural basis of human decision making. Some machine learning systems try to eradicate the need for human intuition in data analysis whereas others embrace a collective approach between humans and machines. The objective of this project is to collect Electroencephalography (EEG) signals through wireless sensors, and the process the collected signals through machine learning methods. The EEG data is collected using three EEG electrodes, each being the positive, negative and ground terminals respectively. Since the signal measures in the unit of micro-volts it needs to be amplified using amplifier circuit. This circuit consists of five stages, namely, the Instrumentation Amplifier, 60 Hz Notch Filter, 31Hz Low Pass Filter, Gain Stage, and Clamper Circuit. Each of these stages contribute in their own way to amplify and also filter the noise from the EEG signal. The gain of the entire circuit is about 5140 V/V. The EEG signal is observed on the cathode-ray oscilloscope and the data is collected on an Arduino Uno microcontroller using the Hyperterminal software. The data is sampled at a sampling rate of 838 Hz. Any remaining noise from the signal can be removed by passing it through a digital low-pass filter, if required.

EEG data is collected from 10 people while they are made to concentrate on a particular thought. The subjects were asked to imagine moving an object towards the direction ‘Right’ for the first 150 data sets collected and then the same for the direction ‘Left’ for another 150 data sets. All the data was collected with complete supervision and without any kind of movement of the subject during the collection process. This part of this data was then used to design a Machine Learning model called Support Vector Machine (SVM) on Matlab, using which the data was then processed. The data was divided into two sets, namely. The training data, and testing data.

One-third of the total data sets is used for training the SVM model and then the rest of the data sets are used to test the accuracy of the model. Data from the 10 individuals is processed individually and also after being mixed. The SVM model randomly selects the training and testing data and accordingly gives results for the accuracy. Finally, results for the individual data and mixed data are tabulated and presented in the following thesis.

Acknowledgements

I would like to sincerely thank Dr. Jingxian Wu for his invaluable mentorship and support through my MS degree and giving me the opportunity to work on this project.

I would like to thank Dr. Baohua Li and Dr. Jing Yang for agreeing to serve as part of my committee, and for the guidance and support they have offered throughout.

I would like to thank my co-partner Andrew Simms for his complete support and help in a part of the project.

I would like to thank Iroshi (Ro) Windwalker (IRB/RSC Coordinator, University of Arkansas) for helping me get the IRB Approval to collect EEG data from people.

I would like to appreciate all the help and support from my friends and colleagues Qing Guo, Ali Ojutiku, Huong Tran, Vinith Bejugam, Israel Akingeneye, Quang Li, Venkatesan Rajagopalan, Andrew Simms, M. Theerth Raj, Aric Fisher, Nikhil Basutkar, Shivendra Tripathi, and Rohit Narala for volunteering to collect EEG data for the project.

Table of Contents

Chapter 1: Introduction 1

1.1 Introduction to Electroencephalography (EEG) 1

1.2 An Overview of the Thesis 3

1.3 Literature Study 7

1.3(A) Classification of Support Vector Machine 7

Chapter 2: EEG Signal Collection 10

2.1 Constructing the EEG Circuit 11

2.1(A) Hardware parts required to construct the EEG circuit 11

2.1(B) Compiling the Hardware together to form the EEG circuit 16

2.2 Collecting the EEG Data 24

Chapter 3: Machine Learning and Support Vector Machines (SVM) 26

3.1 Introduction to Machine Learning 26

3.2 Introduction to SVM 26

3.2(A) Linear Classification of Binary Data 27

3.2(B) Non- Linear Classification for Binary Data 31

3.3 Experimental Design 34

Chapter 4: Results and Discussion 37

4.1 Results from the EEG Circuit 37

4.2 Results after Processing the EEG Signal using SVM 43

Chapter 5: Conclusion and Future Work 48

Bibliography 50

Appendix 52

Chapter 1: Introduction

1.1 Introduction to Electroencephalography (EEG)

EEG is the measure of the electrical activity inside a human brain. It measures the voltage changes that are caused due to the ionic movement within the neurons of the brain. Clinically, it is said that EEG records the spontaneous electrical activity taking place inside the brain. This activity is monitored for a short period of time, example, 20-30 minutes. Some of the main neurological diagnostic applications of EEG are in case of epilepsy, coma and brain death. However, it is also used for studies of sleep and sleep disorders. This is can be done because of the clear abnormalities observed in the EEG signals during these disorders. It can also be used to diagnose tumors and strokes.

The electrical charge of the brain is maintained by billions of neurons that are electrically charged by membrane transport proteins that pump ions across their membranes. There is constant exchange of ions going on between the neurons and the extracellular milieu. Ions of similar charge repel each other. When ions from many neurons are pushed out at the same time they tend to push their neighbors who thereby push their neighbors, creating a wave. This process is called volume conduction. When this wave of ions reaches the scalp and comes in contact with the metal of the electrode it pushes or pulls electrons on the metal. The metal used is a good conductor of electrons it causes is voltage difference between any two electrodes. The recording of these voltage changes is what forms the EEG signal. The electric potential generated by a single neuron is extremely small and hence cannot be picked up by the electrodes. Therefore, EEG activity is always the summation of the synchronous activity of thousands or millions of neurons with similar spatial orientation.

An EEG signal is measured in the range of micro volts (μV), making it a very small signal to be observed on any instrument. Also it is divided into different types of signals based on the frequency of the signal. Table 1.1 below shows the differentiation of EEG signals.

Table 1.1 Bands of EEG Signals

Band Name	Frequency Range (Hz)	Location of the signal	Activity
Delta	< 4	Found in the Frontal lobe in adults; Posterior lobe in children	During slow sleep range in adults and in babies.
Theta	4-7	Found in locations that are not related to the task at hand	During drowsiness and idling in teens and adults.
Alpha	8-15	Found on the posterior region of the head on both sides.	When relaxing or reflecting back on thoughts.
Beta	16-31	Symmetrically found on both sides of the head but most evidently found in the frontal lobe.	During active thinking, stress, focusing, high alertness and anxiousness.

Table 1.1 Bands of EEG Signals Cont.

Band Name	Frequency Range (Hz)	Location of the signal	Activity
Gamma	> 32	Found in the Somatosensory cortex.	When two different senses are combined, example: sound and sight.
Mu	8-12	Found in the Sensorimotor cortex	"Shows rest state motor neurons" [1].

From the above table it can be observed that the very small Delta and Theta EEG signals are usually observed in people only while they are drowsy or asleep. The Alpha and Beta signals are the ones observed when people are active and the high frequency gamma signals are observed only in case of cross-modal sensory processing. This shows that the alpha and beta signals can be most easily be collected from people. Hence, in this thesis we have concentrated on mainly collecting the alpha and/or beta signals.

1.2 An Overview of the Thesis

This thesis deals with the collection of EEG signals and then processing them and using that data to control a robot. The purpose of this research is to help the physically disabled people control objects with just their brain. They can perform various actions for example, switching on the television or moving things around by just strongly thinking about it. A technology like this is of great importance as it can give complete independence to disabled people in their day to day life.

A brief review of the entire procedure used to collect and process the EEG data is explained further.

A head band is constructed in order to collect the EEG signals. This band is made of a three Disposable MR Conditional Cup Electrodes EEG sensors which are directly in contact with the human head. The electrodes are worn using the International 10-20 system. Based on this system one electrode is worn on either corner of the forehead and the third one is worn on the bone behind the ear.

The range of EEG signal in a typical adult human is about 10 μV to 100 μV in amplitude when measured from the scalp and is about 10-20 mV when measured using subdural electrodes. Hence, in order to be able to view the EEG signal on a CRO (Cathode Ray Oscilloscope) an amplifier circuit is required to be built. This circuit consists of three main stages; the pre-amplification stage, the gain stage and a notch filter. It is constructed using two TL084 IC's, one AD620AN Instrumentation amplifier and one notch filter. The overall gain of this circuit is 5140 V/V. The circuit has three inputs and one output. The three inputs are connected to the three EEG electrodes and the output is connected to the oscilloscope. When a 12 V signal is supplied to the circuit while the EEG head band is worn, it displays the amplified EEG signal on the oscilloscope. The output of the circuit is connected to an Arduino UNO microcontroller. The Arduino UNO and the series one XBee are used to collect the EEG signal data and communicate it to the robot.

"An Arduino is a single-board microcontroller used to make applications involving interactive objects more accessible"[2]. The XBee is a radio module that is "designed for point-to-point and star communications at over-the-air baud rates of 250 kbits/s"[3]. These two together are used to

collect and then transmit the EEG data to the robot. The Hyperterminal software is used to collect the EEG data from the Arduino connected to the output of the amplifier circuit. The data is collected at a sampling frequency close to 1k Hz for about 3-4 seconds, which gives a total of 3000 data collected in each sample. A huge number of such samples are collected while strongly thinking about taking a “Right” and taking a “Left” respectively. These samples are then processed using the concept of SVM. This data is also filtered using a digital low pass filter designed on Matlab. The sampling frequency used to design the low pass filter is 837 Hz, since that is a close estimation of the actual sampling frequency at which the data is being collected. The filter is designed using the ‘FDAtool’ graphical user interface available in the signal processing toolbox on Matlab.

As defined on Wikipedia [4], Machine learning is a branch of artificial intelligence that can be used to construct and study systems that can learn from data. Taking the example of an email message, a machine learning system can be trained to distinguish between spam and non-spam messages and then after learning, it can be used to allocate new emails into the spam and non-spam folders. The basis of machine learning consists of two steps, representation and generalization. The representation learning algorithms preserves the information in their input and then transforms it making it useful. In this step several data samples are trained and then based on that they are classifications and predictions about the information are made. After representation the learner has to then generalize from its experience. It has to accurately perform on new tasks after experiencing a set of learning data. The training data belongs to an unknown probability distribution and the learner builds a general model about this space, which helps it to make sufficiently accurate predictions about new cases.Machine learning either tries to entirely eliminate the need for human intuition during data analysis or it uses a combined approach of

both human and machine. However, it is impossible to completely eliminate human intuition, since the system's designer has to specify how the data is represented and the mechanisms that will be used to characterize the data.

As mentioned on the Wikipedia website[5], SVM is a supervised machine learning model that is used to analyze data, recognize its pattern and then classify new data based on the analysis. When a set of training data (having data points belonging to two different categories) is processed using SVM, it constructs a model that distinguishes new data into either of the two categories. This makes it a "non-probabilistic binary linear classifier"[5]. Data belonging to different categories is divided by a definite wide gap in between the points. New samples of data are then mapped into the same space, allocating them a side on either side of the gap depending on the category they are predicted to belong to. Apart from performing linear classification, SVM can also perform non-linear classification using the kernel trick, which implicitly maps the inputs into high-dimensional feature spaces. In case of EEG, the linear classification is performed.

A matlab code is written that processes all the data samples. Data is collected from 15 people which is then processed. Each data set has about 150-200 EEG samples of thinking ‘Right’ and ‘Left’ respectively. The EEG data is processed using a 5-fold cross validation method. Cross validation also known as rotation estimation, is a model validation technique that is used to determine if the results of a statistical analysis can generalize to a set of new data. It is mainly used during prediction, to estimate the accuracy of a predictive model in practice. In case of a prediction problem, the model is usually divided into a set of known data on which training is run and another set of unknown data on which testing is performed. In a single fold cross validation the data sample is divided into complementary subsets, performing training on one

subset and then testing on the other one. In order to reduce variability, several loops of cross validation are implemented using different partitions, and the results are averaged over the loops. For 5-fold cross validation the data is divided into five subsets, performing training on four subsets and then testing one the fifth one. This process is repeated five times making each subset the testing data once and the remaining four as the training data. After completing the 5-fold cross validation process, the accuracy of the model is computed. This result is what gives us the accuracy of the SVM model designed on Matlab.

1.3 Literature Study

1.3(A) Classification of Support Vector Machine

SVM is a classification method that separates data by constructing hyperplanes that separate each class of data. It can perform classification as well as regression tasks for both continuous and categorical variables. The categorical variables are usually replaced by a dummy variable, which is either 0, or 1. The SVM uses a training algorithm that minimizes the error function and hence constructing a hyperplane to classify the data. Based on this error function used, it is divided in to four different types of groups:

1. "Classification SVM Type 1 (also known as C-SVM classification)"[6]:

The error function to be minimized for the SVM classification Type 1 is as follows:

$$\frac{1}{2}w^{T}w + C\sum_{i=1}^{N}\xi_{i}\text{[6];}$$

Subject to $y_i\,(w^{T}\phi(x_i) + b) \geq 1- \xi_i$ and $\xi_i \geq 0$, i=1,….N[6].

Here, C= Capacity constant, *w*= vector of coefficients, and b= constant.

It is observed that the greater the value for C, the larger is the error. Hence, the C needs to be chosen such that it reduces the error function.

2. "Classification SVM Type 2 (also known as nu-SVM classification)" [6]:

The error function to be minimized for the SVM classification Type 2 is as follows:

$\frac{1}{2}w^{T}w - \upsilon\rho + \frac{1}{N}\sum_{i=1}^{N}\xi_i$[6];

Subject to $y_i\,(w^{T}\phi(x_i) + b) \geq \rho - \xi_i$ and $\xi_i \geq 0$, i=1,….N and $\rho \geq 0$ [6].

3. "Regression SVM:

In this case there noise is added to the signal ("y= f(x) + noise" [6]). For any Regression model in general the SVM is assigned to find a function for 'f' that is capable of predicting how to classify any new set of data. This can be implemented by training the SVM model on some initial sample sets and then testing it on the new data. The training is done the same way as for the Classification Type SVM. It is dived into two types based on the error function that is optimized.

a. Type 1 (also known as epsilon-SVM regression):

The error function minimized for this type is SVM is as follows:

$\frac{1}{2}w^{T}w + C\sum_{i=1}^{N}\xi_i + C\sum_{i=1}^{N}\xi_i^{*}$[6]; subject to,

$w^{T}\phi(x_i) + b - y_i \leq \varepsilon + \xi i^{*}$ [6]

$y_i - w^{T}\phi(x_i) - b \leq \varepsilon + \xi_i$[6]

$\xi i, \xi i^{*} \geq 0$, i= 1,…N

b. Type 2 (also known as nu-SVM regression):

$\frac{1}{2}w^T w - C\{ \upsilon\varepsilon + \frac{1}{N}\sum_{i=1}^{N} (\xi_i + \xi_i^*) \}$[6]; subject to,

$(w^T\phi(x_i) + b) - y_i \leq \varepsilon + \xi_i$[6]

$y_i - (w^T\phi(x_i) - b_i) \leq \varepsilon + \xi_i^*$[6]

$\xi_i, \xi_i^* \geq 0$, i= 1,…N; $\varepsilon \geq 0$ [6]

There are also different types of Kernel Functions that can be used for designing a SVM Model. "A Kernel function, represents the dot product between the input data points mapped into a higher dimensional feature space by transformation ϕ" [6].

1. Linear: $K(X_i, X_j) = X_i . X_j$[6]
2. Polynomial: $K(X_i, X_j) = (\gamma X_i . X_j + C)^d$ [6]
3. RBF: $\exp(- \gamma |X_i . X_j|^2)$ [6]
4. Sigmond: $\tanh (\gamma X_i . X_j + C)$ [6]

Here $K(X_i, X_j) = \phi(x_i) . \phi(x_j)$ [6]

The kernel function chosen for this project is the RBF, as it is one of the most widely and commonly used kernel function for designing a SVM model. "This is mainly because it's localized and finite responses across the entire range of the real x-axis" [6].

Chapter 2: EEG Signal Collection

This chapter mainly contains the detailed explanation of all the equipment and hardware used during the process of this experiment with an account of how each of them were used to achieve the final results. The process of collecting EEG signals can be classified into three main parts. The first and foremost step is to select the right kind of EEG electrodes. There are several types of electrodes available in the market which can be used for different types of purposes. It is essential to select the electrode that best suits the experiment and will be able to help extract a strong signal from the brain. The next step is to build an EEG circuit that would amplify the EEG signal and make is viewable on a Cathode Ray Oscilloscope. The EEG circuit itself is divided into several sections which will explained subsequently in this chapter. This circuit not only amplifies the signal but also removes some of the noise from it. After the circuit is ready and tested, the final step is to collect the EEG data. This was collected using an Arduino UNO microcontroller since it is easy to program and can also give a very high sampling frequency for the signal collection. The entire process of these three main steps is elucidated at length in the further sections of this chapter.

This chapter is divided into three major sections,

1. Constructing the EEG Circuit: This will be further divided into sub-sections each describing the hardware parts that were used to construct the circuit, the process of assembling them together and then lastly, testing the functionality of the circuit.

2. Verification of the signal for EEG: After compiling all the hardware together and a sample signal was collected to test for EEG. It was then verified whether the signal contained any useful EEG information.

3. Collecting the EEG data: This section comprises of a detailed summary of collecting the EEG data and transmitting it wirelessly using an Arduino UNO and XBee. It explains how the microcontroller was programmed to collect the data and then transmit it to another XBee.

2.1 Constructing the EEG Circuit

This section explains in depth how to construct a simple EEG circuit that will help in monitoring the EEG signal on an oscilloscope. The circuit is constructed in a way that it can be used for measuring both EEG and ECG (measure of the heartbeat). It uses three EEG electrodes, two for measuring the change in voltage across the scalp and one is used as the ground terminal. This project uses a minimum of three electrodes. In order to be able to use more than three electrodes, several such EEG circuits need to be constructed and linked together. Constructing this circuit is very reasonable economically.

2.1(A) Hardware parts required to construct the EEG circuit

Following is the list of all the parts needed to construct a simple EEG circuit with a brief description of each part.

1. 1x Instrumentation Amplifier:

AD620ANZ: The datasheet for the AD620 given by Digi-Key Corporation states that [7], the AD620 is an instrumentation amplifier with low cost and high accuracy. It requires only one external resistor to adjust its gain from 1 to 10,000. It has a 8-lead SOIC and DIP packaging which is smaller as compared to the discrete designs and offers a lower power value of 1.3mA maximum supply current. This makes is a perfect fit for battery-powered and portable applications. AD620 has accuracy up to 40 ppm, a low offset voltage of max 50µV and a max

offset drift of 0.6µV/ºC. This makes it ideal for being used in precision data acquisition systems. Also, its low noise, low input bias current and low power makes it perfectly suited for being used in medical applications, for example EEG, ECG and noninvasive blood pressure monitors. The AD620 can be used as a pre-amplifier because of its low input voltage noise. An instrumentation amplifier can also be built using three operational amplifiers, but it very rare that it would give a result as good as the AD620ANZ.

The connections diagram for the AD620ANZ is shown in figure 2.1 below.

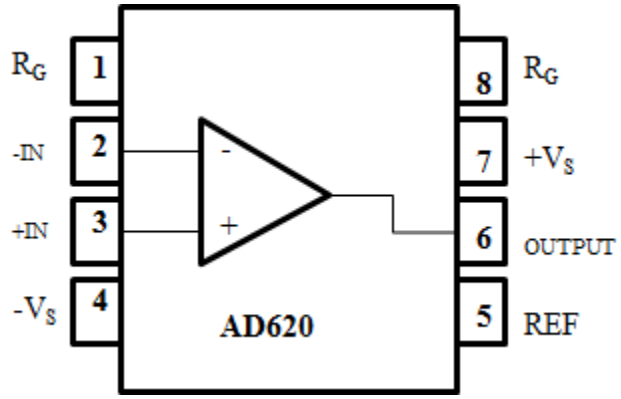


Figure 2.1 Pin Diagram of AD620 Instrumentation Amplifier [7].

2. 2x Operational Amplifier:

TL084IN: As mentioned in the datasheet for the TL084 [8], it is a high-speed JFET input quad operational amplifier. It is a monolithic integrated circuit incorporating high voltage JFET, and bipolar transistors. It has a high slew rate, low input bias and offset current, and a low offset voltage temperature coefficient. The pin connections diagram for TL084 is shown in figure 2.2 below.

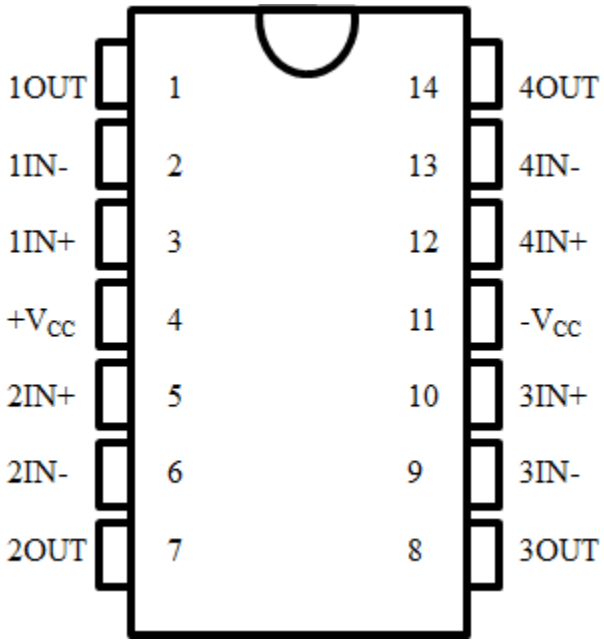


Figure 2.2 Pin Connection Diagram for TL084 Op-Amp [8].

3. 1x Notch Filter- UAF42AP-ND:

"The UAF42 is an universal active filter which can be used to configure a large range of low pass, high pass, and band pass filter" [9]. "It uses an inverting amplifier and two integrators having a classic state-variable analog architecture" [9]. The integrators consist of 1000pF capacitors on the chip, cut down to 0.5%.

4. Capacitors
5. Resistors
6. 3x EEG Electrodes:

This circuit design uses three EEG electrodes, two for measuring the voltage across the scalp and the third one is used as a ground electrode. The disposable cup EEG Ag/AgCl electrodes with a cup size of 10mm are used. The disposable cups ensure the safety of the people and eliminate the time spent in cleaning and disinfecting the electrodes.

7. 1x Breadboard:

A large enough breadboard is need to wire the entire circuit on.

8. Connecting Wires:

Several connecting wires of various sizes are required to make connections in the circuit on the breadboard.

9. 2x Arduino UNO:

An Arduino UNO is the microcontroller which is used to collect the EEG signal data and then transmit it to another arduino. It consists of 6 analog inputs, 14 digital input/output pins, a 16 MHz resonator, a USB connection, a power jack, an ICSP header, and a reset button. It is a self-sufficient microcontroller and simply needs to be connected to a computer with a USB cable, power it with an AC-to-DC converter or use with a battery. An arduino can receive inputs from a variety of sensors and thereby control the surroundings such as lights, motors, and other actuators. It can be programmed using the Arduino programming language. The figure 2.3 below shows an image of the Arduino UNO used in this project.

Figure 2.3: An image of the Arduino UNO.

10. 2x XBee PRO:

A pair of series 1 XBee's are programmed on the Arduino and then used to collect the EED data on one Arduino and then transmit it to the other arduino. It is a radio module which is used for over-the-air communication at a baud rate of 250kbits/s. An XBee is perfect for applications that need a low latency and predictable communication timing. It provides a rapid and robust point-to-point, peer-to-peer, and multipoint/star configuration communication. It can be used instead of a pure cable in case of simple serial communication or in a complex hub-and-spoke network of sensors. The XBee multipoint RF modules increase wireless performance and make development much easier. The figure 2.4 below shows the picture of the series 1 XBee used in this project.

Figure 2.4: An image of the series 1 XBee.

2.1(B) Compiling the Hardware together to form the EEG circuit

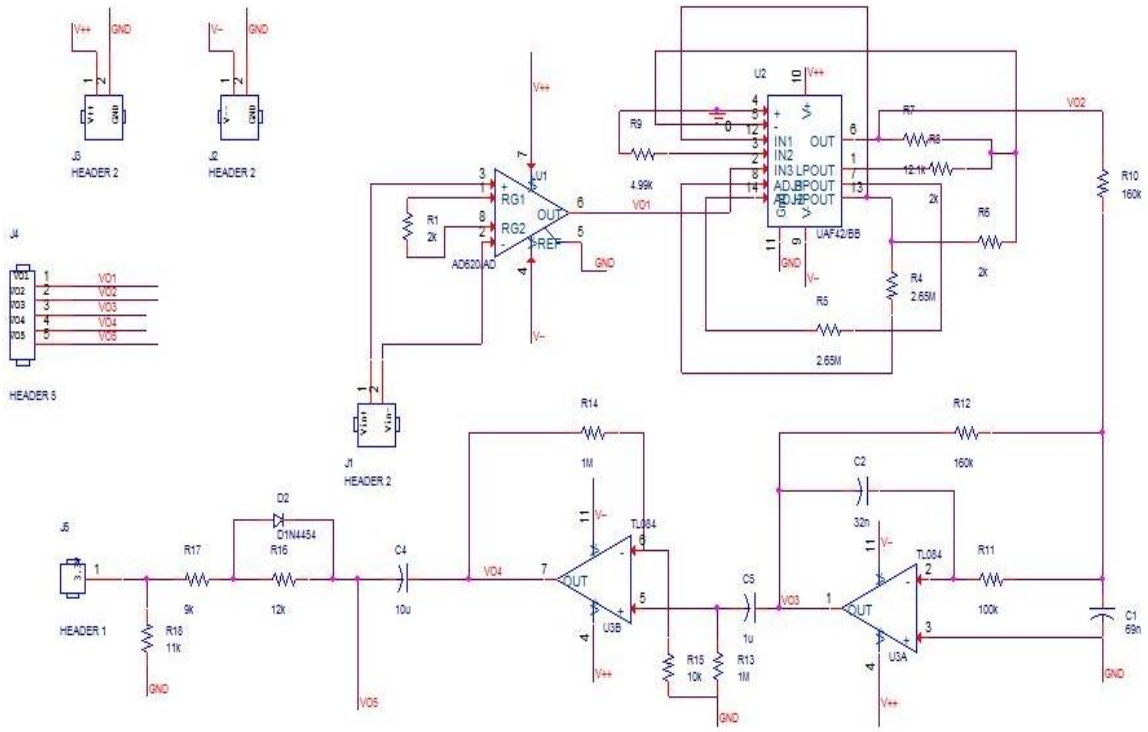


Figure 2.5 Schematic diagram of the overall EEG Circuit.

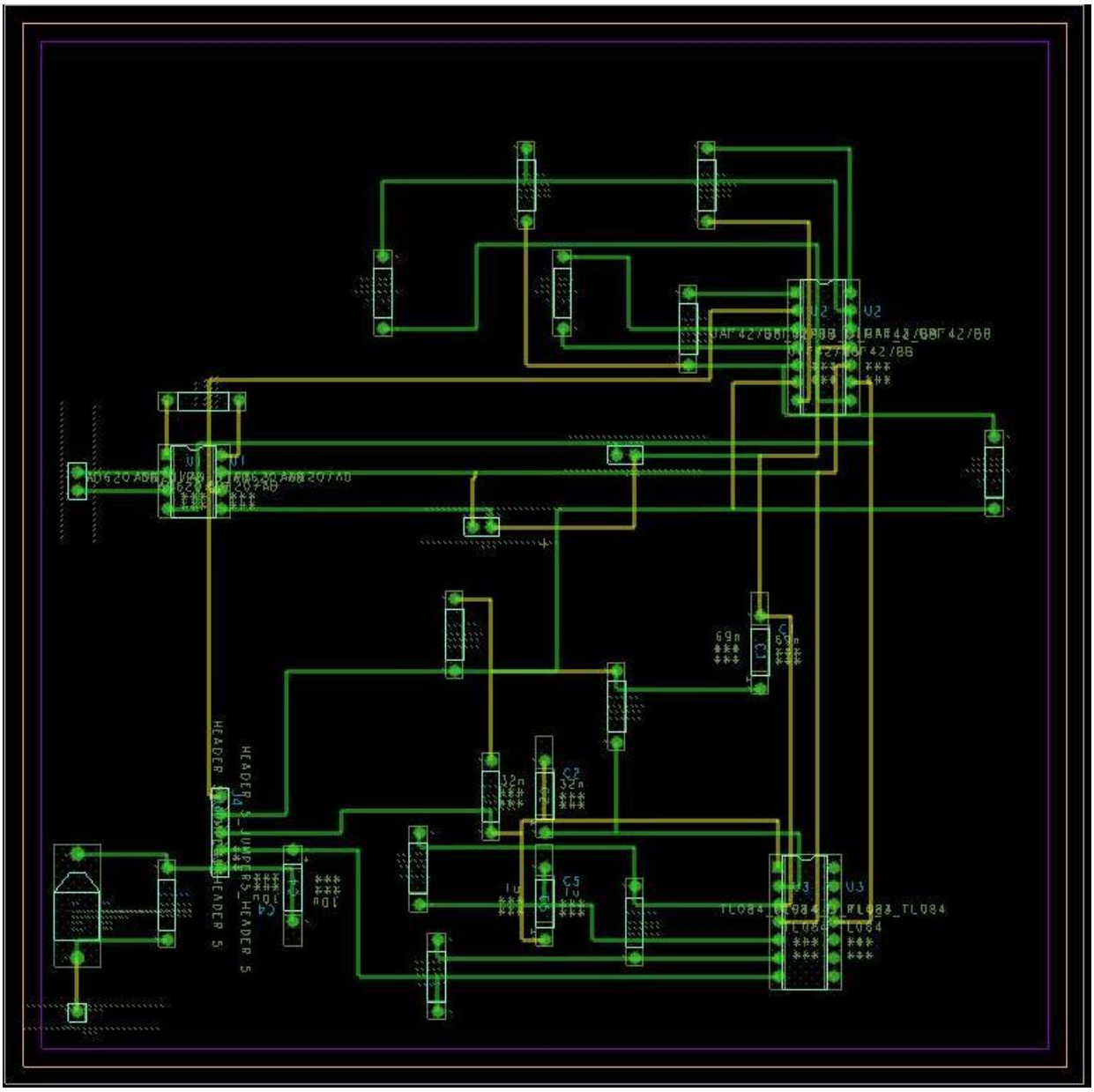


Figure2.6: PSpice model of the EEG Circuit.

Given above in figure 2.5 is the final schematic diagram of the EEG circuit, formulated on P-Spice. This circuit has been slightly influenced from the ECG circuit given in the DIY Instructables [10]. "The instrumentation amplifier is used in the beginning of the circuit after which each box is a simple op-amp, used to construct the 60 Hz notch filter, the high-pass filter, low-pass filter and the gain stage" [10].

The circuit is powered with a 12V Power supply from the frequency generator. To power the op-amps with a -12V to +12V power supply connect positive terminal of the frequency generator to the positive terminal of the breadboard and the negative terminal of the frequency generator to the negative terminal of the bread board. The positive and negative terminals are also shorted and that is connected to the ground terminal of the breadboard. The main goal of this circuit is to retrieve EEG data by reducing the noise enough to obtain a good signal into the computer. The signal will then be further passed through a digital low-pass filter to negate as much noise as possible. This is done by designing a digital low-pass filter on Matlab. The filtered data is then processed on matlab using SVM, which gives the accuracy of the data.

The EEG circuit is divided into several stages for compilation simplicity. Given below are the division on the circuit into stages and an explanation of each stage:

1. Stage 1- Instrumentation Amplifier stage:

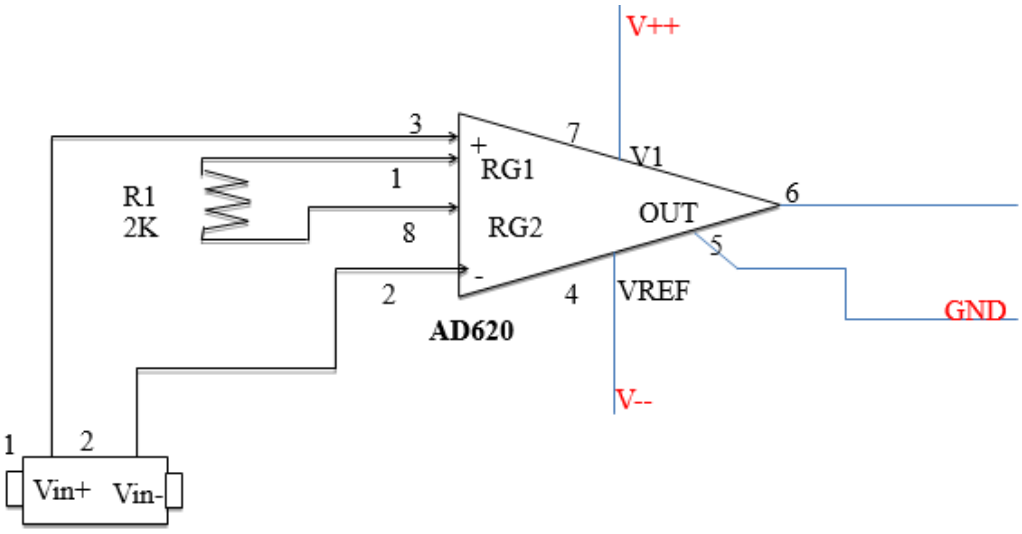


Figure 2.7: Connections for the Instrumentation amplifier.

The positive and negative EEG electrodes are connected to the two input pins (pin 3 and pin 2 respectively) of the AD620ANZ Instrumentation Amplifier. “The output is the difference between the two input voltages multiplied with a gain value of” [10] 25.7 V/V (G). However, the instrumentation amplifier is not perfect which results in a slight change in the output when the inputs are offset the same by some amount. Considering an example from the DIY Instructables [10], an ideal instrumentation amplifier with inputs 3.1V and 3.2V will give an output of 0.1V*G, whereas the real one will be influenced by the common offset, and will marginally change the output. This change in the output value is not significant enough and can be neglected.

The value of the resistor connected across pins 1 and 8 determines the gain of the instrumentation amplifier. The datasheet for AD620 is attached in the abstract [7], from which the formula for calculating the gain can be seen (G= 49.4k Ω/R1 + 1). In order to observe the EEG signal a gain of about 25V/V is required. Hence, using the formula we can calculate the value of the resistor (Rg= 2k Ω) which gives us a gain value of 25.7 V/V. Pin 7 is given the 12 V power supply from the frequency generator and pin 5 is grounded. It is possible to build an instrumentation amplifier using op-amps, but it is advisable to use an AD620 instead to get a good reading.

2. Stage 2- 60Hz Notch Filter:

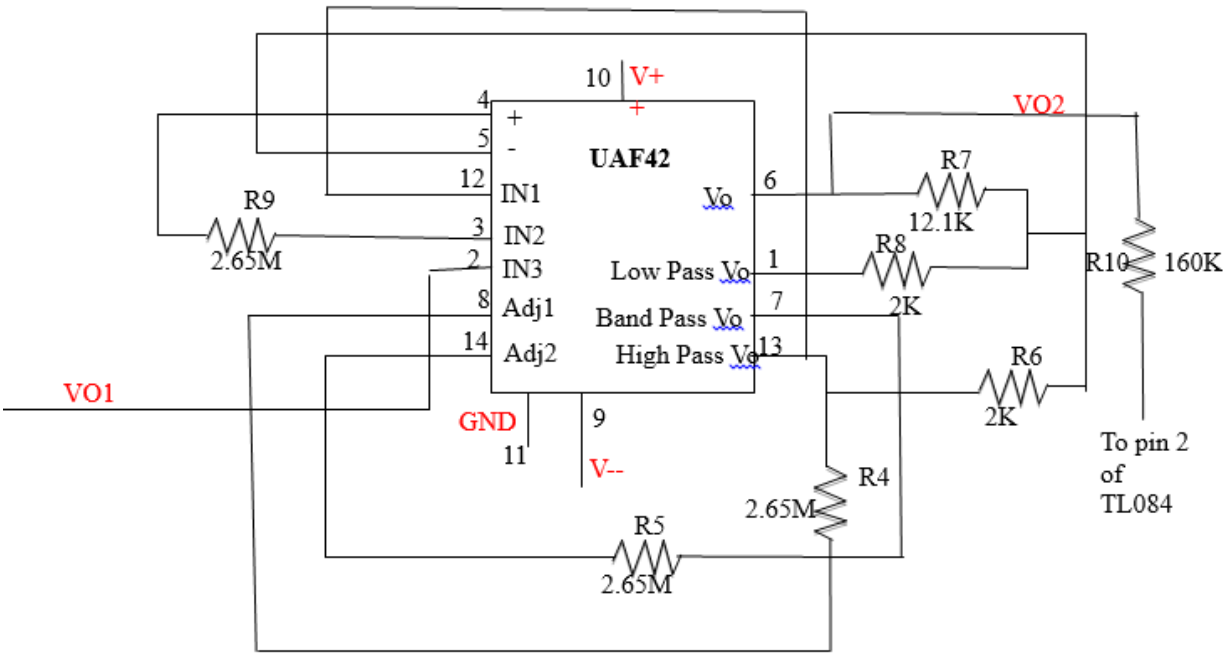


Figure 2.7: Connections for the Notch Filter.

“The biggest source of noise in the system will be centered at 60Hz” [10], whether you use an external power supply or batteries. Due to this reason a notch filter has to be incorporated in the circuit. It will negate as much interference as possible before we apply further gain to the circuit. A second notch filter can be built at the end of the circuit too to final cut off any more interference that may have been added to the circuit. The notch filter at the end can also be replaced by a digital notch or low pass filter for simplicity of the circuit and also to get better results.

3. Stage 3- 31Hz Low Pass Filter

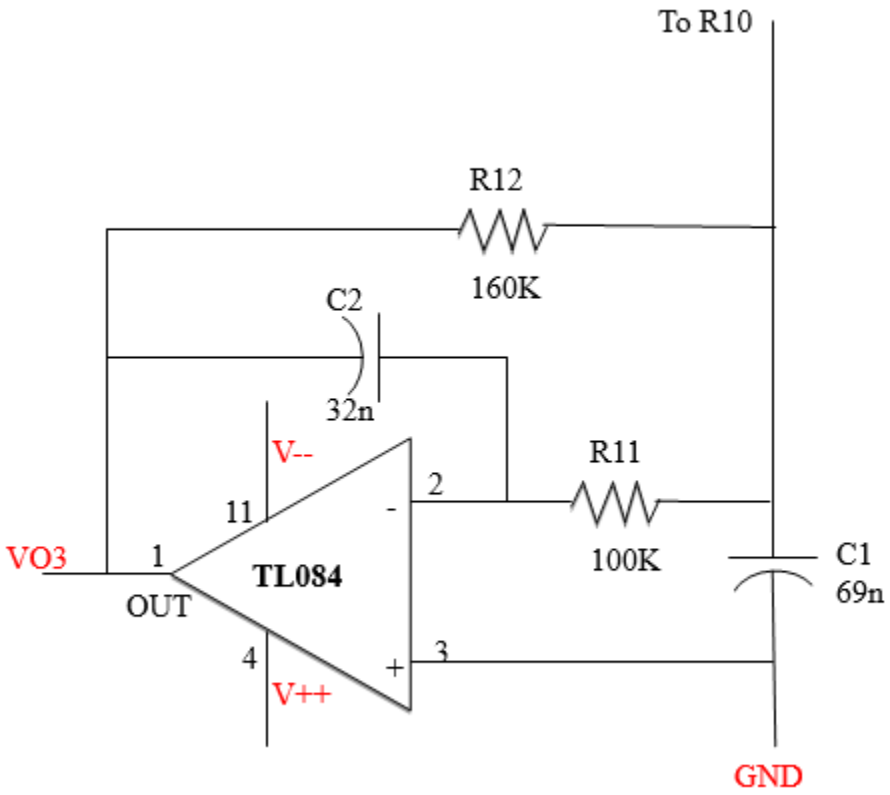


Figure 2.8: Connections for the 31Hz Low Pass Filter.

"A low pass filter is built to filter out the frequencies that are above the alpha/beta/gamma frequency range" [10]. Not filtering out these frequencies can lead to a good amount of noise in our final output. It has a cut-off frequency of 31Hz.

4. Stage 4- Gain Stage:

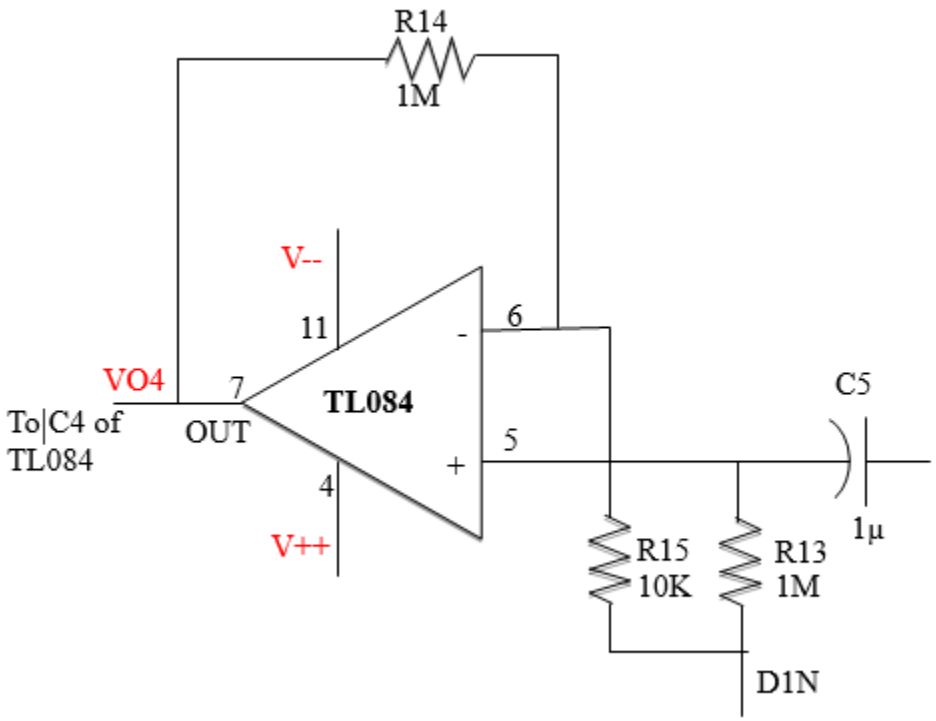


Figure 2.9: Connections for the Gain Stage

"This section of the circuit has a quick high pass filter with a cut-off frequency of 1Hz (Fc= 1/2*pi*R13*C5) in the beginning" [10]. The high pass filter is built to get rid of some extra unwanted noise. However, the main purpose of this circuit is to adjust the gain of the EEG circuit using the 1M potentiometer. "The potentiometer is variable resistor, with the input connected to the first pin and the output to the second pin and on turning the wiper the resistance varies between 0 to 1M ohms" [10]. Adjust the potentiometer such that when viewing the EEG signal without any body movement the voltage should not fluctuate off-screen. The amplitude of the signal need not be at its highest value but at the same time it should not be too small that it causes an error while digitally reading it on the computer. It is also important to make sure that

gain is not too high causing the signal to get clipped, as in such a case there is a possibility of losing some important information. The gain of the EEG Circuit excluding the instrumentation amplifier is calculated to be 200.

5. Stage 5- Clamper Circuit:

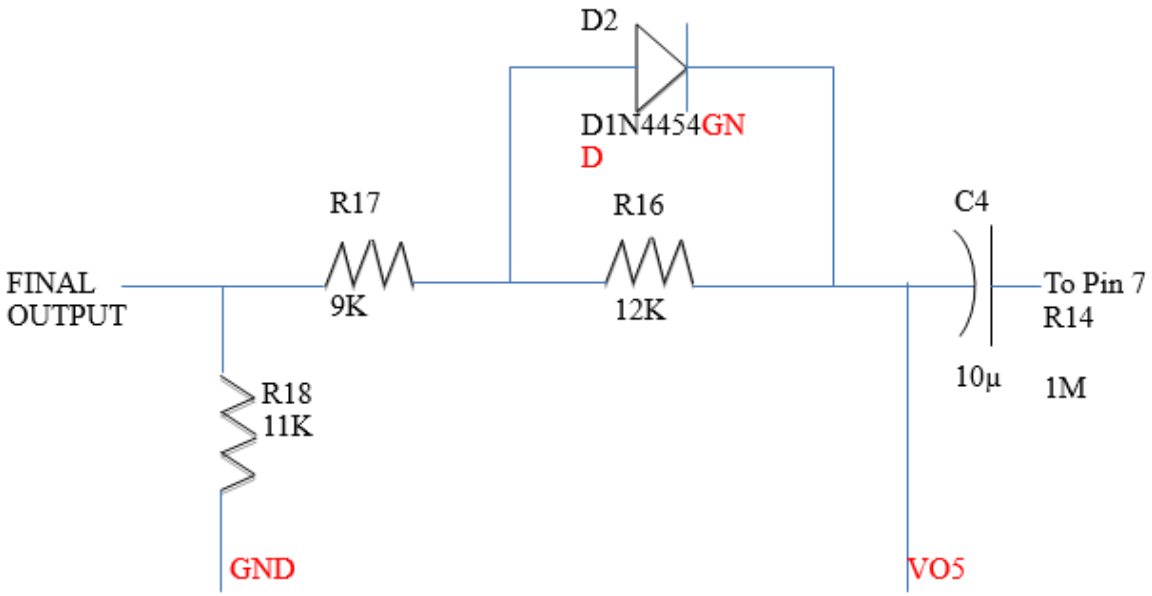


Figure 2.10: Connection diagram for the Clamper Circuit.

The voltage of the signal obtained at the end of the gain stage has some values below 0 Volts which cannot be read by the arduino. The Arduino UNO reads all values below 0 Volts as zero, this can completely change the signal causing the loss of some important EEG data. Hence, a clamper circuit needs to be added to the end of the EEG circuit as it shifts the entire signal up such that all the peaks are above 0 Volts. The output from the clamper circuit is the final EEG signal that is seen on the oscilloscope.

Once the entire circuit is built it needs to be tested before proceeding to the step of collecting EEG signals. The results from testing each stage of the circuit is explained in detail in the results and discussion chapter (chapter 4).

2.2 Collecting the EEG Data

After observing the signal on the oscilloscope, the EEG data is collected using the arduino and XBee. The XBee is programmed using the Arduino to collect samples of the EEG signal. The sampling frequency is 838Hz precisely, which is very close to 1K Hz, a very good sampling frequency. The Arduino code used to program the XBee is given in the Appendix.

Firstly, the baud rate of the XBee is set to 115200 and the same baud is set in the code too. The sampling frequency is set to 838 Hz and the sampling time is set to 3.5 seconds. So, each set of data contains 3000 (838*3.5) samples of data. This code collects 100 such sets of data with 10 sets collected continuously. After every 10 sets of data the reset button on the arduino needs to be pressed to start the next set of data collection. It takes 35 seconds to collect 10 sets of data. For this thesis, 250 sets of data were collected while thinking of the left and right directions respectively. Such 250 sample sets were collected twice using two different collection techniques.

In the first technique, 250 data sets of thinking right were collected all together, with and interval of 2-3 minutes after every 50 sets. A 10 minute break was taken then taken, after which 250 data sets of thinking left were collected again with a 2-3 minute break after every 50 sets. The second technique followed almost the same procedure except that after every 50 sets of data collected the thought was changed. For example, if 50 sets of thinking right were collected first, then 50 sets of thinking left would be collected after that with a 2-3 minute interval in between. The

software used to collect the signal is called HyperTerminal. Its saves the data into a notepad file with commas separating each value and every set of data starting from the next line.

The next step after collecting all this data is to process it and check its accuracy. The SVM algorithm is used to process the data. A detailed explanation of this algorithm is given in the following chapter.

The reason for collecting the data using two different techniques is to check how each technique affects the accuracy of the SVM algorithm. This would give an idea as to what technique should be used to collect the data to be finally processed. An IRB approval is taken to test this device on people and data is collected from 8 different people, asking them to think both left and right.

Chapter 3: Machine Learning and Support Vector Machines (SVM)

This chapter consists of the basics of Machine Learning, concentrating on the SVM algorithm. It give a brief introduction of machine learning and then explains in detail about SVM.

3.1 Introduction to Machine Learning

Machine learning is a branch of artificial intelligence that deals with the construction and study of systems that can be trained using data. Taking the example of an email message, a machine learning system can be trained to distinguish between spam and non-spam messages and then after learning, it can be used to allocate new emails into the spam and non-spam folders. The basis of machine learning consists of two steps, representation and generalization. The representation learning algorithm s preserves the information in their input and then transforms it making it useful. In this step several data samples are trained and then based on that they are classifications and predictions about the information are made. After representation the learner has to then generalize from its experience. It has to accurately perform on new tasks after experiencing a set of learning data. The training data belongs to an unknown probability distribution and the learner builds a general model about this space, which helps it to make sufficiently accurate predictions about new cases.

3.2 Introduction to SVM

SVM is a state-of-art classification method introduced in 1992 by Boser, Guyon and Vapnik" [11][12]. "The SVM classifier is widely used in bioinformatics (and also other disciplines) due to its high accuracy, ability to deal with high dimensional data such as gene expression, and , flexibility in modeling diverse sources of data"[11][13]. "A SVM belongs to the general class of

kernel methods" [11][14][15], an algorithm which is dependent on data only through dot-products. In such a case, one can replace the dot product with a kernel function that can compute the dot product in a high dimensional feature space. As inferred from the User's Guide SVM by Asa Ben-Hur, and Jason Weston published in 2010 [11], there are two main advantages of this method: firstly, the fact that it can develop non-linear decision boundaries from techniques used for linear classifiers; and secondly, kernel function permit user to classify data that have no fixed-dimensional vector space representation.

3.2(A) Linear Classification of Binary Data

The description given below has been obtained from the work by Tristan Fletcher, published in 2009 [16]. Let there be L training points, with input x_i each having D attributes and belonging to one of the two classes y_i= 0 or 1. Hence, the training data if of the form:

$\{x_i, y_i\}$ where i=1,.....,L; yi ϵ {-1, 1}; x ϵ R^D , where D is the dimensionality [16](3.1)

The data here is assumed to be linearly separable, that is a line can be drawn on the graph of x_1 vs x_2 when D = 2, and a hyperplane on the graphs of x_1, x_2....x_D when D > 2, to separate the two classes. "The hyperplane can be described by the equation, $\mathbf{w}.x + b = 0$, where" [16]

- **"w** is the normal to the hyperplane"[16].
- "b/||**w**|| is the perpendicular distance from the hyperplane to the origin"[16].

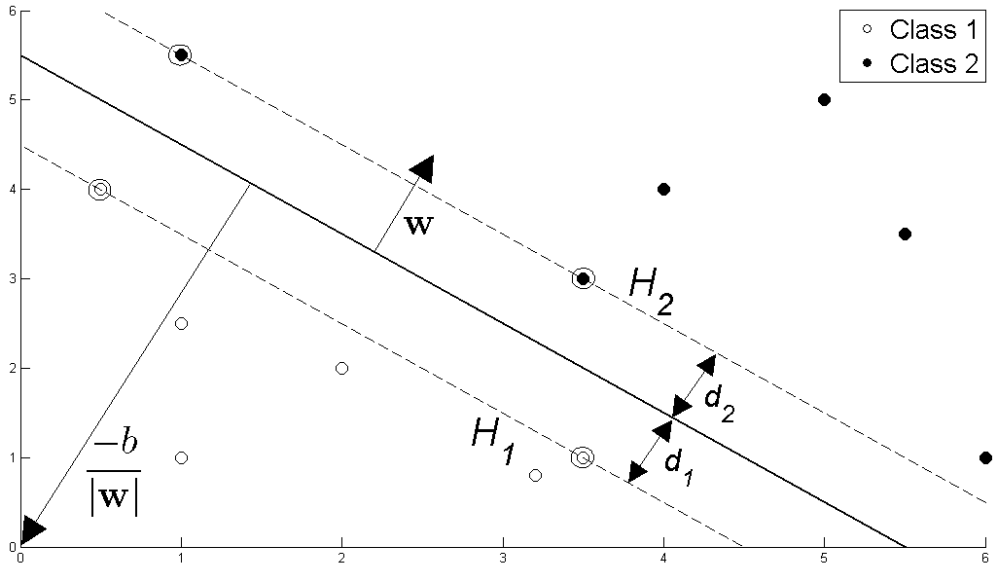


Figure 3.1: Hyperplane through Linearly separable classes [16].

The data points located closest to the hyperplane on either side are called Support Vectors. SVM aims at positioning the hyperplane in a way that it is as far as possible from the closest data point on either side. This means that the distance between the support vectors and the hyperplane must be maximized. Referring to figure 3.1, it can be said that enforcing SVM cuts down to choosing values for **w** and b such that the training data can be represented as:

$x_i \,.\, \mathbf{w} + b \geq +1$ for $y_i = +1$ [16] (3.2)

$x_i \,.\, \mathbf{w} + b \leq -1$ for $y_i = -1$ [16] (3.3)

Together these equations can be written as,

$y_i(x_i \,.\, \mathbf{w} + b) - 1 \geq 0 \; \forall_i$ [16](3.4)

From the above equations it can be implied that the planes H_1 and H_2 on which the Support Vectors (i.e. the data points closest to the hyperplane) lie can be represented using the following equations:

$x_i \,.\, \mathbf{w} + b = +1$, for H_1[16] (3.5)

$x_i \,.\, \mathbf{w} + b = -1$, for H_2[16] (3.6)

As, shown in figure 3.1, the distance from H_1 to the hyperplane is represented by d_1 and from H_2 to the hyperplane is d_2. Since, the hyperplane is equidistant from both H1 and H_2, $d_1=d_2$ (called the SVM's margin). This margin is what needs to be maximized in order to place the hyperplane as far as possible from the Support Vectors.

The margin is calculated to be $1/\|\mathbf{w}\|$ using simple vector geometry. "Maximizing it subject to the constraint in equation (3.4) is equivalent to finding" [16]:

$\min \|\mathbf{w}\|$ such that $y_i(x_i \,.\, \mathbf{w} + b) - 1 \geq 0 \forall_i$ [16] (3.7)

Equation (3.6) can also be written as:

$\min \frac{1}{2}\|\mathbf{w}\|$ such that $y_i(x_i \,.\, \mathbf{w} + b) - 1 \geq 0 \; \forall_i$ [16] (3.8)

"In order to minimize the constraints, a set of Lagrange multipliers $\boldsymbol{\alpha}$ need to be allocated to them, such that $\boldsymbol{\alpha_i} \geq 0 \; \forall_i$" [16]:

$$L_p \equiv \frac{1}{2}\|\mathbf{w}\|^2 - \boldsymbol{\alpha}[y_i(x_i \,.\, \mathbf{w} + b) - 1 \; \forall_i \,] \text{ [16]}$$

$$\equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{L} \alpha_i \,[\, y_i(x_i \,.\, \mathbf{w} + b) - 1] \text{ [16]}$$

$$\equiv \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{L} \alpha_i y_i (x_i \cdot \mathbf{w} + b) + \sum_{i=1}^{L} \alpha_i \quad [16] (3.9)$$

The values for w, b and α need to be found in equation (3.9) such that w and b minimize and α maximizes the equation. This can be achieved by differentiating L_p with respect to w and b and setting that to zero.

$$\frac{\partial L_p}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{L} \alpha_i y_i \mathbf{x}_i \quad [16] (3.10)$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_{i=1}^{L} \alpha_i y_i = 0 \quad [16] (3.11)$$

Substitute equations (3.10) and (3.11) in (3.9) in order to formulate a new equation L_D, which needs to be maximized as it is dependent only on α.

$$L_D \equiv \sum_{\alpha=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j x_i . x_j \text{ such that } \alpha_i \geq 0 \ \forall_i \ , \sum_{i=1}^{L} \alpha_i y_i = 0 \quad [16]$$

$$\equiv \sum_{\alpha=1}^{L} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i H_{ij} \alpha_j, \text{ where } H_{ij} \equiv y_i y_j x_i \cdot x_j \quad [16]$$

$$\equiv \sum_{\alpha=1}^{L} \alpha_i - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \text{ such that, } \alpha_i \geq 0 \ \forall_i , \sum_{i=1}^{L} \alpha_i y_i = 0 \quad [16] \quad (3.12)$$

Hence, we need to find:

$$\max_{\alpha} \left[\sum_{\alpha=1}^{L} \alpha_i - \frac{1}{2} \boldsymbol{\alpha}^T \mathbf{H} \boldsymbol{\alpha} \right] \text{ such that, } \alpha_i \geq 0 \ \forall_i , \text{ and} \sum_{i=1}^{L} \alpha_i y_i = 0 \quad [16] (3.13)$$

Solving the above equation gives the values for α, and w when substituted in (3.10).

Calculating the value for b:

Let x_S be any data point that satisfies (3.11). Hence, it will be for the form:

$y_S(x_S \cdot w + b) = 1$ [16] (3.14)

Substituting this in (3.10), we get:

$y_S (\sum_{m \epsilon S} \alpha_m y_m x_m \cdot x_S + b) = 1$[16] (3.15)

Here, S is the set of indices of Support Vectors and it can be obtained by determining the indices i (where $\alpha_i > 0$). Then, multiply (3.15) by y_S throughout, and from (3.2) and (3.3) let $y_S^2 = 1$.

$y_S^2(\sum_{m \epsilon S} \alpha_m y_m x_m \cdot x_S + b) = y_S$[16]

$b = y_S - \sum_{m \epsilon S} \alpha_m y_m x_m \cdot x_S$[16] (3.16)

Equation (3.16) can also be written using the overall average of the support vectors in S.

$b = \frac{1}{Ns} \sum_{s \epsilon S} (y_S - \sum \alpha_m y_m x_m \cdot x_s)$[16] (3.17)

Solving equations (3.13) and (3.17) give the values for **w** and b, the two parameters that define "the hyperplane's optimal orientation and hence the structure for the Support Vector Machine" [16].

3.2(B) Non- Linear Classification for Binary Data

This theorem has also been obtained from the work by Tristan Fletcher, published in 2009 [16]. Section 3.2(A) explained the method for constructing an SVM structure for linearly separable data points. But in the real world most of the data is not linearly separable. Hence, a methodology for constructing a SVM structure for non-linear data is explained in this section. A positive slack variable ξ_i, i= 1,...L, needs to be introduced in equations (3.2) and (3.3), making the equations as follows:

$x_i \cdot \mathbf{w} + b \geq +1 - \xi_i$ for $y_i = +1$ [16] (3.18)

$x_i \cdot \mathbf{w} + b \leq -1 + \xi_i$ for $y_i = -1$ [16], where $\xi_i \geq 0 \; \forall_i$ (3.19)

The above equations can be combined into a single equation as follows:

$y_i(x_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0 \; \forall_i$,where $\xi_i \geq 0 \; \forall_i$ [16]

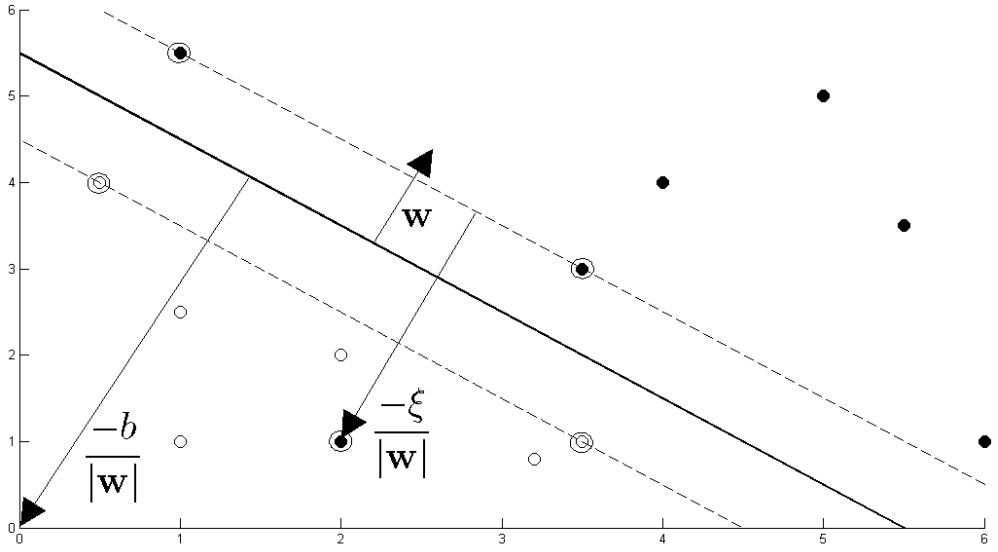


Figure 3.2: Hyperplane through two non-linearly separable classes [16].

In case of non-linear data, the data can be scattered anywhere in the plane and hence it is not possible to form a hyperplane to divide the data. In this case, the soft margin method is used to choose a hyperplane that divides the data points as cleanly as possible along with maximizing the distance to the nearest data point. In this way the number of miscalculations can be reduced. In order to do so the following equation is used as the objective function:

$\min \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{L} \xi_i$ such that $y_i(x_i \cdot \mathbf{w} + b) - 1 + \xi_i \geq 0 \; \forall_i$ [16] (3.20)

Here, the trade-off between the slack variable penalty and the margin size is regulated by the variable C. Lagrange Multipliers are used to solve the above equation (3.20) along with the given constraint.

$$L_p \equiv \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{L} \xi_i - \sum_{i=1}^{L} \alpha_i [\, y_i(x_i . \mathbf{w} + b) - 1 + \xi_i] - \sum_{i=1}^{L} \mu_i \xi_i [16] \quad (3.21)$$

The above equation (3.21) is then differentiated with respect to **w**, b, and ξ_i, resulting in the following:

$$\frac{\partial Lp}{\partial \mathbf{w}} = 0 \Rightarrow \mathbf{w} = \sum_{\mathbf{i=1}}^{\mathbf{L}} \alpha_i y_i x_i \textbf{[16]} (3.22)$$

$$\frac{\partial Lp}{\partial b} = 0 \Rightarrow \sum_{\mathbf{i=1}}^{\mathbf{L}} \alpha_i y_i = 0 [16] (3.23)$$

$$\frac{\partial Lp}{\partial \xi_i} = 0 \Rightarrow C = \alpha_i + \mu_i [16] (3.24)$$

Using the above equations, we can formulate L_D, as done for the linear data points. L_D for non-linear data has the same form as for the linear data. Now, combining equation (3.24) and $\mu_i \geq 0$ $\forall_i$, signifies that $\alpha \leq C$. Therefore, we have to find the following:

$$\max_{\alpha} [\sum \alpha_i - \frac{1}{2}\alpha^T H \alpha] \text{ such that } 0 \leq \alpha_i \leq C \ \forall_i \text{ and } \sum \alpha_i y_i = 0, \text{ where } L < i \leq 1 \ [16] (3.25)$$

"The value for 'b' is calculated the same way as it was done for linear data in section 3.2(A)"[16], except that the group of support vectors used are found by determining the values of indices i where $0 \leq \alpha_i \leq C$. Hence, the final structure of Non-Linear SVM is attained using this procedure.

3.3 Experimental Design

SVM has been used to classify the EEG data collected using the EEG Headband constructed. This section will give an explanation of the SVM structure designed and how the analysis is done. The SVM design is made using Matlab coding. A function is written to implement SVM on the data collected (attached in the Appendix). The data collected is firstly divided into two sets, the Training data set ($2/3^{rd}$ of the total data sets) and the Testing data set ($1/3^{rd}$ of the total data sets). The training data set is used to train the SVM design. Using this data set the structure for SVM is constructed. Once the SVM has been designed, the testing data set used to test if the SVM is correctly designed and gives the results as required. This is analyzed by checking the accuracy SVM.

The SVM design was made using the power spectral density of the data. Discrete Fourier Transform (DFT) was implemented on time domain data to convert it to frequency domain using a code written on Matlab. The following equation is used to perform Discrete Fourier Transform on the data:

$$X(k) = \sum_{n=1}^{N} x[n]\omega_N^{(n-1)(k-1)}, \text{ where } \omega_N = e^{-2\pi j/N} \tag{3.26}$$

Then the Power Spectrum Density (PSD) is computed using the equation given below:

$$S_{xx} = \frac{1}{Fs*N}|X(k)|^2 \tag{3.27}$$

After converting the data to frequency domain it is then processed using SVM. The data that has been collected in real time is non-linear and hence the methodology of designing a SVM explained in section 3.2(B) is implemented to form the SVM structure. In this project, we have collected 10 sets of data from 10 different people while they think of the direction 'Right' and

'Left' respectively. Each data set has 150 trials in it, i.e. 150 trials of thinking 'Right and 150 of thinking 'Left'. Each trail contains 3000 data valuesat a sampling rate of838 Hz. The first step is to find the PSD of the raw data. This is done using the Matblab code to convert time domain signal to frequency domain given in the Appendix of the thesis. The PSD is computed using the 'periodogram' function in Matlab. As explained on the Mathworks website [17]:

"Pxx= periodogram(x,window,nfft); Here nfft points are used in the discrete Fourier transform (DFT)" [17]. If the value of nfft is greater the length of the signal, the original data x is zero-padded to length nfft. But, if nfft has a value lesser than the signal length, x is wrapped modulo nfft and a summation is taken using the datawrap. Taking an example of the input signal [1 2 3 4 5 6 7 8] with a nfft of 4, the periodogram result will be sum([1 5; 2 6; 3 7; 4 8], 2). In the code used for this thesis the value for nfft is taken to be $F_s/2$=419. The periodogram function computes one-sided PSD and hence it is multiplied by 2 to get the two-sided PSD. Hence, after computing the PSD the dimension for the input data and w vectors is equal to $F_s/2$, which is 419.

After computing the PSD of the original EEG data, one-third of the samples were taken as testing data and two-third were used for training the SVM. After performing PSD on the data it is used to design the SVM using the equation 3.28 given below.

The following equation is used to design the structure of the SVM:

$$\text{argmin}_{w,\xi,b} \{\tfrac{1}{2}\|\mathbf{w}\|^2 + C\textstyle\sum_{i=1}^{n} \xi_i\} \text{ subject to } y_i(w \,.\, x_i - b) \geq 1 - \xi_i, \quad \xi_i \geq 0 \text{ for all } i=1,...n \qquad (3.28)$$

Solving this equation gives the position of the hyperplane that separates the data points. The first step to solve the equation is to optimize 'w'. This is done by minimizing ||w|| subject to the given constraint: $y_i(w \,.\, x_i - b) \geq 1 - \xi_i$. After minimizing 'w' the next step is to find the optimum value

for ‘C’ and ‘ξ_i’. The dimension of **w** is 419, which is also the dimension for the input vector x. In order to do so, a procedure known as 5-Fold Cross Validation is used.

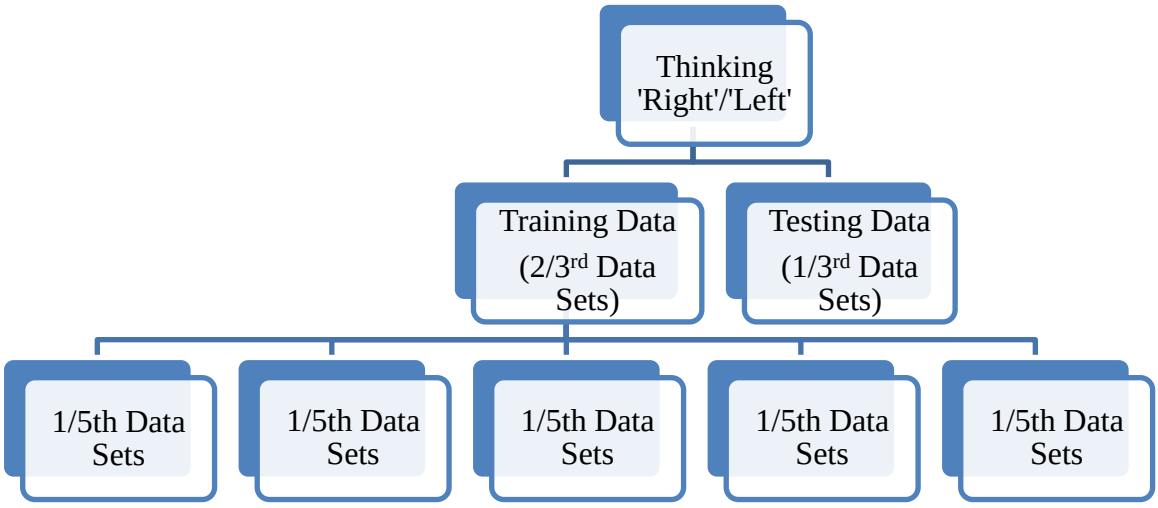


Figure 3.3: Experimental Design for 5-Fold Cross Validation.

Figure 3.3 above shows a pictorial representation of the 5-Fold Cross Validation method. In this method the training data sets are further divided into five equal parts, where one part is used as testing data set and four parts are used for training. Using this the value for C and ξ_i is determined. This step is repeated five times, making each of the $1/5^{th}$ data set the testing data set once. Hence, it will give five values for C and ξ_i. Finally, the best or most optimum value is chosen to design the SVM hyperplane. The values for C and ξ_i are substituted in equation (3.28) to find the position of the dividing hyperplane. After this, the final step is to test the design of the SVM one the data in the testing data set and find the accuracy of the design.

Chapter 4: Results and Discussion

The results obtained from the experimental procedure have been shown in this chapter. They are in the same order as the procedure. The first section has the results from the EEG circuit after the signal is collected and the second section has results of the signal after it is processed using SVM.

4.1 Results from the EEG Circuit

After constructing the circuit as instructed in section 2.1 of the chapter 2, the next step is to test it. The output at the end of each stage of the EEG circuit is verified by connecting the probes of the oscilloscope at the end of each stage. These observations have been graphed and explained further in this section.

Stage 1: Instrumentation Amplifier:

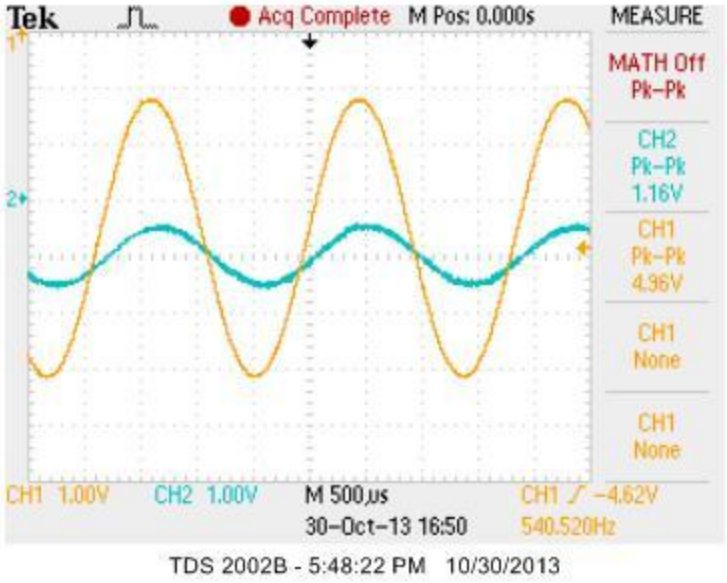


Figure 4.1: Instrumentation amplifier Gain.

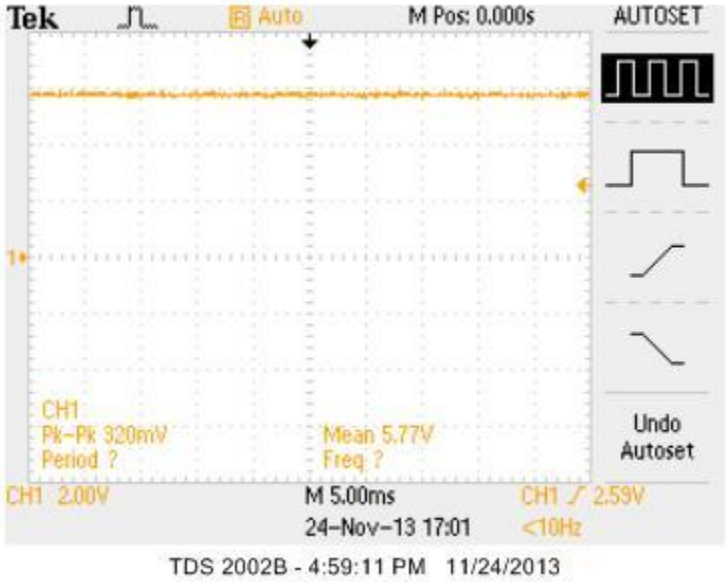


Figure 4.2: Output from the Instrumentation Amplifier

The figure 4.2 above shows the output from the instrumentation amplifier when the EEG electrodes are worn on the scalp. This signal has a small amplitude since not enough gain has been applied to the signal at this stage of the circuit. The frequency is almost 130 Hz which shows that there is a lot of noise added to the signal. This noise will be negated further in the circuit. The gain of the instrumentation amplifier is shown in figure 4.1 above. This is the screen shot from the oscilloscope when given a sine wave as input to the instrumentation amplifier. The wave in blue is the input and the wave in yellow is the amplified output.

Stage 2.3: Notch Filter and Low Pass Filter

(a) (b)

Figure 4.3: (a) Output at the end of the Notch Filter; (b) Output from the Low Pass Filter.

The out from the 60Hz notch filter and the 31Hz low pass filter is shown in figure 4.3(a) and 4.3(b) respectively. This basically shows that a great portion of the noise has been removed from the original EEG signal. In the next section of the circuit the signal will pass through the gain stage to amplify it further.

Stage 4: Gain Stage

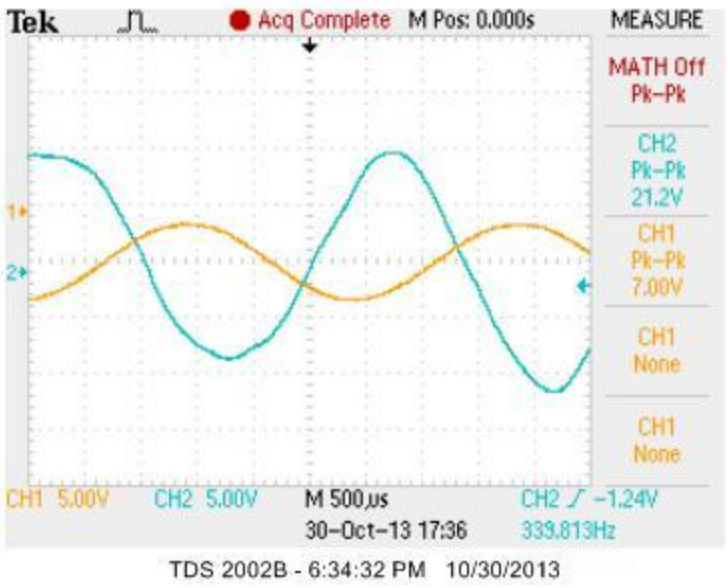
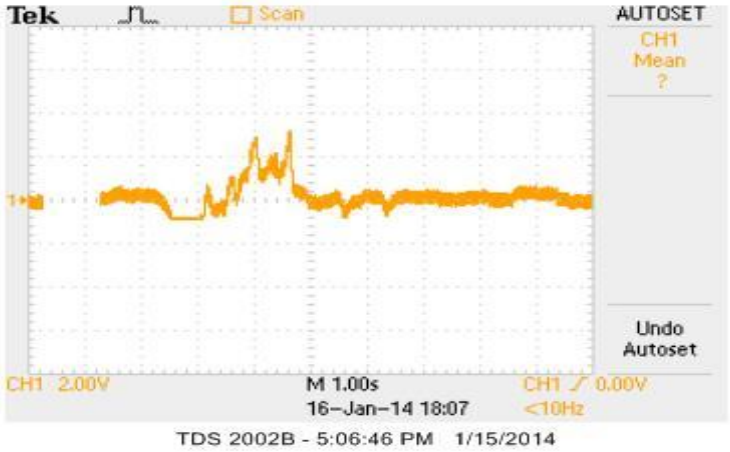


Figure 4.4: The overall gain of the EEG circuit.

The overall gain of the circuit is 5140 V/V (G= 25.7 V/V * 200 V/V) . This gain value will be multiplied to the voltage difference between the negative and positive EEG electrodes. The yellow wave in figure 4.4 is a sine wave input and the blue colored wave is the amplified version. The output at the end of the gain stage is shown below in figure 4.5(a). The slight fluctuation in the signal is due to movement of the right arm. When the arm is moved up a positive peak is observed, and when it moves down the peak is negative.



4.5(a): EEG Signal from the gain stage with arm movement.

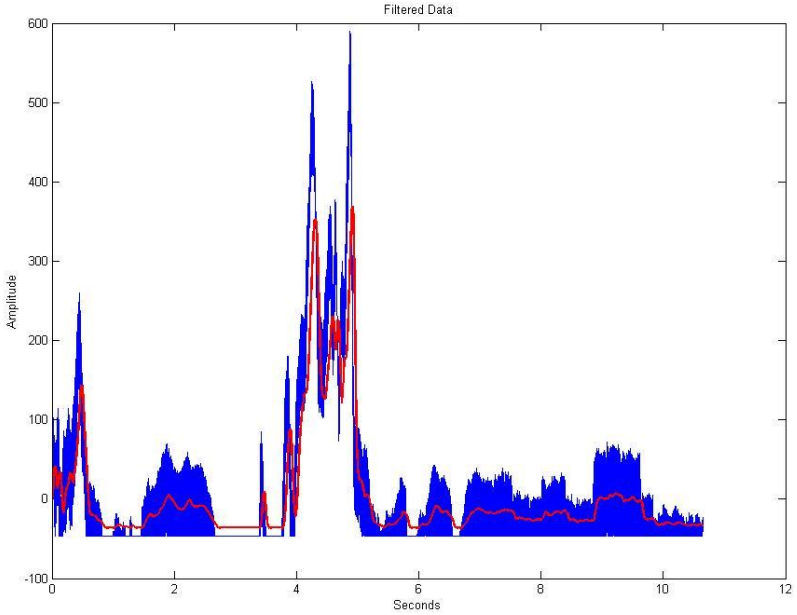


Figure 4.5 (b): Filtered EEG signal from the gain stage with arm movement.

Figure 4.5(b) is the filtered version of the signal in figure 4.5(a). The signal from the oscilloscope is passed through a digital low pass filter designed on Matlab. It is designed using the 'fdatool' on Matlab. The sampling frequency (Fs) used is 838Hz, which is the sampling

frequency at which the data is collected on the arduino. The cutoff frequency used is 31Hz, since most of the signal collected should be below 31Hz (theta, alpha, and beta waves).

It is evident from the figure that the negative part of the signal is clipped off as the arduino uno cannot read any values below zero. This causes loss of some EEG data, which is why the signal need to be offset using a clamper circuit. So, a clamper circuit is added at the end of the gain stage and the final plot of the signal is shown further in the section.

Stage 5: Clamper Circuit

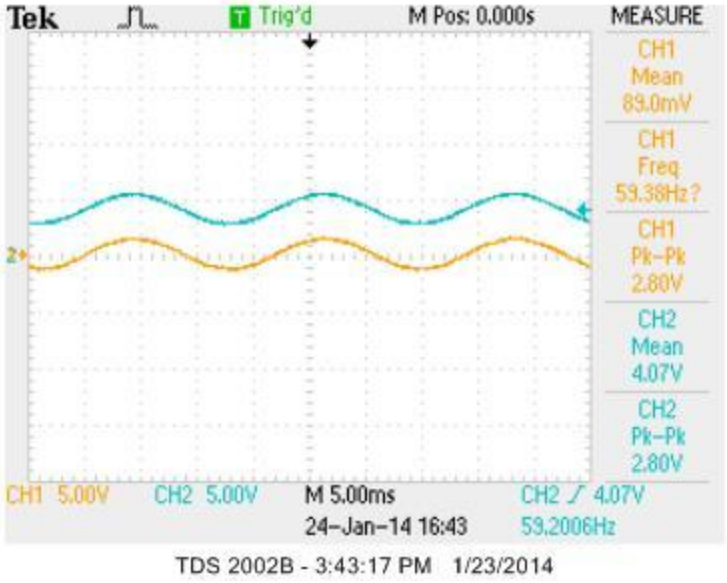


Figure 4.6: Testing the output of the clamper circuit with a sine wave.

The clamper circuit was built as per the circuit given in section 2.1(B) of chapter 2 and a sine wave it passed through it to check the output. The output of the clamper circuit is shown in figure 4.6 above. The yellow colored wave is the given input, and as observed it has a few data points that lie below zero. The blue colored wave is the output which is offset and its reference

axis has been brought up, bringing all the data points above the zero axis. Hence, the arduino can now read all the data values of the signal.

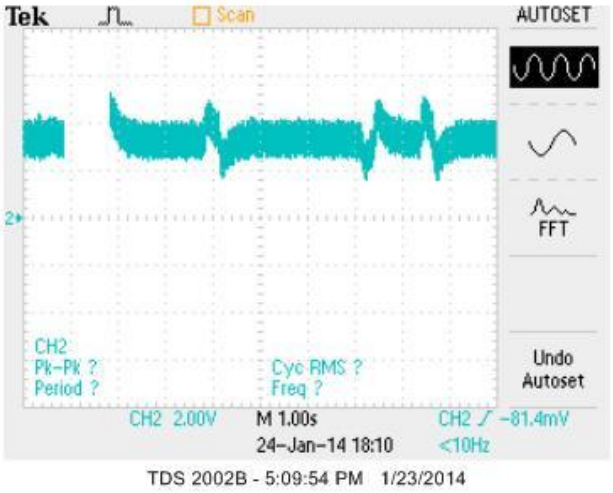


Figure 4.7: Final EEG signal output from the EEG circuit.

Figure 4.7 displays the EEG signal after being offset using the clamper circuit. As you can see in the image, now the entire EEG signal has been shifted above the zero axis. The two extreme peaks seen in the image is due to the right to left eye movements. The first peak is seen when the eye moves right and the second peak is seen when it moves left.

4.2 Results after Processing the EEG Signal using SVM

This section shows the results from processing the EEG data collected using SVM. EEG signals were collected from several people while they were asked to think about the directions ‘Right’ and ‘Left’ respectively. They had to concentrate on visualizing any object moving toward the left or right without it actually moving. Data was collected from 10 individuals at different times, each data set having 150 samples of each direction. These samples were processed individually as well as after mixing all of them together and the accuracy of the SVM model was determined.

All the values have been shown in the table 4.1 below.

Table 4.1: Results from processing the EEG data using SVM.

Sr. No.	Data Set (Thinking 'Left' and 'Right')	Unknown Parameters	Error Rate (%)	Training Accuracy (%)	Cross Validation Accuracy (%)	Test Accuracy (%)
1	300 Samples from Individual 1	Best C = 12.8071 Best ξ = 30.8766	28	100	72	83
2	300 Samples from Individual 2	Best C = 104.505 Best ξ = 9.4877	21.5	100	78.5	81
3	300 Samples from Individual 3	Best C =73.6998 Best ξ = 7.846	16	100	84	88
4	300 Samples from Individual 4	Best C = 33.1155 Best ξ = 15.7998	26	100	74	71

Table 4.1: Results from processing the EEG data using SVM Cont.

Sr. No.	Data Set (Thinking 'Left' and 'Right')	Unknown Parameters	Error Rate (%)	Training Accuracy (%)	Cross Validation Accuracy (%)	Test Accuracy (%)
5	485 Samples from Individual 5	Best C = 141.175 Best ξ = 73.6998	2.41	100	97.5192	97.5155
6	300 Samples from Individual 6	Best C = 6.3598 Best ξ = 21.1153	20.50	100	79.5	76
7	300 Samples from Individual 7	Best C = 0.522 Best ξ = 11.0232	27.5	100	72.5	75
8	300 Samples from Individual 8	Best C = 3.4903 Best ξ = 127.7404	4.5	96.125	95.5	94

Table 4.1: Results from processing the EEG data using SVM Cont.

Sr. No.	Data Set (Thinking 'Left' and 'Right')	Unknown Parameters	Error Rate (%)	Training Accuracy (%)	Cross Validation Accuracy (%)	Test Accuracy (%)
9	300 Samples from Individual 9	Best C = 15.6426 Best ξ = 11.473	30	100	70	72
10	305 Samples from Individual 10	Best C = 16.4446 Best ξ = 62.8028	18.64	100	80.9756	84
11	3190 Samples from all 10 Individuals	Best C = 2.7183 Best ξ = 11.9413	28.13	99.9061	71.9249	71.6038

The table 4.1 displays the values for the best C and ξ, error rate, training accuracy and test accuracy for the SVM model designed to process the EEG data. The C and ξ is obtained using 5-fold cross validation. The cross validation accuracy for each trial is also given, which determines

how accuracy of C and ξ. The training accuracy determines the efficiency of training the SVM model and the test accuracy determines how accurate the model is at distinguishing the new data as 'Right' or 'Left. The best test accuracy obtained from an individual is observed to be 97.5155% and the lowest is 71%. The average of all the testing accuracies from individual people is 80%. The error rate decreases as the testing accuracy increases, which means that the best result has the least error rate (0.024). The data collected from each individual was then mixed together and then processed using SVM. The testing accuracy for the mixed data is 71.6038%.

There is a great difference between the highest and the lowest test accuracy from individual people. This could be because thinking about the direction 'Right' or 'Left' is very vague. Each person has their own different way of thinking about the direction. Which is also why when the data from different people is mixed the test accuracy decreases. Also, there were external factor that could have caused error, like noise in the room, slight noise from the EEG circuit, the EEG electrodes not being placed in the perfect position on the scalp. These factors could affect the accuracy of the data collected. The number of EEG sensors used is three, which is the minimum number of sensors to be used for an EEG analysis. Hence, they are not able to sense a great amount of EEG signals. All these factors together justify the results shown in table 4.1.

Chapter 5: Conclusion and Future Work

This thesis describes the method of constructing an EEG Headband that senses EEG signals when worn on the scalp. The Headband is constructed using three EEG Ag/AgCl electrodes. An amplifying circuit is constructed in order to amplify the EEG signals and view them on a Cathode Ray Oscilloscope. These signals are then collected using a microcontroller and transmitted to another microcontroller used to control a robot. Data from the EEG signals is collected and also processed using the Machine Learning algorithm called Support Vector Machine. Data was collected from 10 individuals at different times, where they were asked to think strongly about moving an object to the 'Right' and 'Left' respectively, without actually moving it. About 150-250 data samples for each direction were collected from each individual. They were processed using SVM individually as well as after mixing all the data together. The model is designed using Matlab. The results prove to give an accuracy of 71.6038% for the mixed data and average of 82% for the individual data sets. The accuracy obtained is proved to be good which implies that the model designed is efficient.

The future work for this project can be divided into two parts. One part could be to expand the EEG headband to more than just three electrodes. Several EEG circuits will have to be constructed for that as each circuit takes in only two input electrodes and one ground electrode. All the circuits can be made on a single Printed Circuit Board (PCB) for as many EEG electrodes that one needs to use as that would reduce the noise in the signal. Having more electrodes would also help collect a stronger EEG signal which would also improve the result after processing the signal using SVM. The second part could be to increase the sampling frequency of transmitting the EEG signal from the headband to the robot. The frequency achieved now is about 20-50 Hz, which is very less for the robot to be able to detect if the signal is for moving 'Right' or 'Left'.

To overcome this problem, we used wired connection to transmit the signal to the robot instead of wireless and could achieve a sampling frequency of 837 Hz.

Bibliography

[1] H. GASTAUT, "Electrocorticographic study of the reactivity of rolandic rhythm," *Rev. Neurol. (Paris),* vol. 87, pp. 176-182, 1952. \

[2] "Arduino Project," 31 12 2013. [Online]. Available: http://www.arduino.cc/.

[3] Digi International Inc., "www.digikey.com," 2006-2011. [Online]. Available: http://www.digi.com/pdf/ds_xbeemultipointmodules.pdf

[4] "Machine Learning," Wikipedia, 04 June 2014. [Online]. Available: http://en.wikipedia.org/wiki/Machine_Learning

[5] "Support Vector Machine," Wikipedia, 20 June 2014. [Online]. Available: http://en.wikipedia.org/wiki/Support_Vector_Machine.

[6] StatSoft, Inc., "Electronic Statistics Textbook. Tulsa, OK," 2013. [Online]. Available: https://www.statsoft.com/textbook/support-vector-machines.

[7] Analog Devices, Inc., "www.digikey.com," 2003-2011. [Online]. Available: http://www.analog.com/static/imported-files/data_sheets/AD620.pdf.

[8] Texas Instruments, "www.ti.com," February 1977-Revised Januray 2014. [Online]. Available: http://www.ti.com/lit/ds/symlink/tl084m.pdf.

[9] Texas Instruments, "www.ti.com," July 1992-Revised October 2010. [Online]. Available: http://www.ti.com/lit/ds/symlink/uaf42.pdf.

[10] C. Henry, "http://www.instructables.com/," 22 June 2012. [Online]. Available: http://www.instructables.com/id/DIY-EEG-and-ECG-Circuit/.

[11] A. Ben-Hur and J. Weston, "A user's guide to support vector machines," in *Data Mining Techniques for the Life Sciences*Anonymous Springer, 2010, pp. 223-239.

[12] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers.

In D. Haussler, editor, 5th Annual ACM Workshop on COLT, pages 144–152, Pittsburgh, PA,

1992. ACM Press.

[13] B. Sch¨olkopf, K. Tsuda, and J.P. Vert, editors. Kernel Methods in Computational Biology.

MIT Press series on Computational Molecular Biology. MIT Press, 2004.

[14] J. Shawe-Taylor and N. Cristianini. Kernel Methods for Pattern Analysis. Cambridge UP, Cambridge, UK, 2004.

[15] B. Sch¨olkopf and A. Smola. Learning with Kernels. MIT Press, Cambridge, MA, 2002.

[16] T. Fletcher, "Support vector machines explained," *Tutorial Paper., Mar*, 2009.

[17] The MathWorks, Inc, "www.mathworks.com," The MathWorks, Inc, 1994-2014. [Online]. Available: http://www.mathworks.com/help/signal/ref/periodogram.html.

Appendix

Code to collect data using the Arduino UNO

```
int number = 10;
int samples = 838;
int seconds = 3.5;
int sensorPin = A0;    // select the input pin for the potentiometer
bool off = false;
void setup() {
  // declare the ledPin as an OUTPUT:
  Serial.begin(115200);
}
void loop() {
  // read the value from the sensor:
  while(off)
 {
 }
  getValues();
  off = true;
}
void getValues(){
  int time = micros();
  for(int j = 0;j<number;j++){
  Serial.print("data");
  Serial.print(j+1);
  Serial.print("=[");
    for(int i =0; i<samples*seconds-1; i++){
```

```
      Serial.print(micros()-time);
      Serial.print(", ");
      Serial.print(analogRead(sensorPin));
      Serial.print(", ");
    }
    Serial.print(micros()-time);
    time = micros()-time;
    Serial.print(", ");
    Serial.print(analogRead(sensorPin));
    Serial.println("];");
    Serial.println("");
  }
}
```

Matlab Code for Low Pass Filter:

```
Fs = 838;
number = 10;

%insert data matrices here
data1=[];
data2=[];
data3=[];
data4=[];
data5=[];
data6=[];
data7=[];
data8=[];
data9=[];
data10=[];


Y = zeros(number,3000);
G1 = 5.0/1024.0;
G2 = 25.7
G3 = 200
%G = G1*G2*G3;
G = 5.0/(1024.0*200*25.7)
%G = 200*25.7

i = 1:2:length(data1);
time = data1(i)/1e06;
i = 2:2:length(data1);
data1 = data1(i)*G;
data1 = data1 - mean(data1);
Y(1,:) = filter(Hd1,data1);

data2 = data2(i)*G;
data2 = data2 - mean(data2);
Y(2,:) = filter(Hd1,data2);

data3 = data3(i)*G;
data3 = data3 - mean(data3);
Y(3,:) = filter(Hd1,data3);

data4 = data4(i)*G;
data4 = data4 - mean(data4);
Y(4,:) = filter(Hd1,data4);

data5 = data5(i)*G;
```

```
data5 = data5 - mean(data5);
Y(5,:) = filter(Hd1,data5);

data6 = data6(i)*G;
data6 = data6 - mean(data6);
Y(6,:) = filter(Hd1,data6);

data7 = data7(i)*G;
data7 = data7 - mean(data7);
Y(7,:) = filter(Hd1,data7);

data8 = data8(i)*G;
data8 = data8 - mean(data8);
Y(8,:) = filter(Hd1,data8);

data9 = data9(i)*G;
data9 = data9 - mean(data9);
Y(9,:) = filter(Hd1,data9);

data10 = data10(i)*G;
data10 = data10 - mean(data10);
Y(10,:) = filter(Hd1,data10);

figure(1);
title('Thinking Right');
hold on
subplot(2,2,1);
plot(time,Y(1,:),'r','linewidth',2);
title('EEG Signal 1');

subplot(2,2,2);
plot(time,Y(2,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('EEG Signal 2');

subplot(2,2,3);
plot(time,Y(3,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('EEG Signal 3');

subplot(2,2,4);
plot(time,Y(4,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('EEG Signal 4');
```

```
figure(5);
plot(time,Y(5,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('Filtered Data');

figure(6);
plot(time,Y(6,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('Filtered Data');

figure(7);
plot(time,Y(7,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('Filtered Data');

figure(8);
plot(time,Y(8,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('Filtered Data');

figure(9);
plot(time,Y(9,:),'r','linewidth',2);
xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('Filtered Data');

figure(10);
plot(time,Y(10,:),'r','linewidth',2);
x xlabel('Time (Seconds)'); ylabel('Amplitude (Volts)');
title('Filtered Data');
```

Matlab Code to convert the EEG signal from Time Domain to Frequency Domain:

```
%  transfer time domin signals into frequency domin

clear all
clc

% Fs is sampling rate , N is the amount of sample
Fs = 837;   N = 3000;

powerdata_all1 = csvread('Left formatted.csv',1,0);
powerdata_all2 = csvread('Right formatted.csv',1,0);

powerdata_all1 = powerdata_all1(1:end,[1:2:end])';
powerdata_all2 = powerdata_all2(:,[1:2:end])';

original_data = [powerdata_all1;powerdata_all2];


    for k = 1:300;  % k is the number of different channel
window = hamming(N); %using hamming window to reduce energy leakage
psd_data(k,:) = 2*periodogram(original_data(k,:),window,0:Fs/2,Fs); % periodogram is build in
function to transfer time domin data into frequency domin data
    end
    str =['left and right power data .csv'];
    csvwrite(str,psd_data,0,0)
```

Matlab Function for SVM:

```
function [eeg_svmstrcture] = train_rbfsvm_eeg_Jay(eeg_training_data_x, eeg_training_data_y ,
method1 , method2,bond,alpha)
%Summary of this function goes here

x = eeg_training_data_x;
y = eeg_training_data_y;

number_of_success = sum(y);
r = length(y);
number_of_failure = r - number_of_success;

% creat a index for success and failure
success_index = find(y==1)';
failure_index = find(y==0)';

% suppose a range for C
percentage_bound = bond ; % 95
max_c=15;     %max_c=20, 5
min_c=-5;      %min_c = -12, -5, 1
C_array=exp(min_c:1:max_c);
C_length=length(C_array);

% suppose a range for sigma
max_sigma=5;   %max_sigma=5
min_sigma=-10;  %min_sigma=-10
Sigma_array=exp(min_sigma:1:max_sigma);% 1
G_length=length(Sigma_array);

% select optimal gamma using mulitiple-fold cross validation
xValidationFolds = 5;
rand('state',0); randn('state',0);

indexPermutation_success = randperm(number_of_success); % sample from this index set
setSize_success = number_of_success/xValidationFolds;

rand('state',1); randn('state',1);
indexPermutation_failure = randperm(number_of_failure); % sample from this index set
setSize_failure = number_of_failure/xValidationFolds;

% here comes my model selection loop
train_correct_rate=zeros(C_length, G_length);
test_correct_rate=zeros(C_length, G_length);
optimazation_criterion_2=zeros(C_length, G_length);
%%
```

```
%first searching
for k=1:C_length
   for j=1:G_length
      for fold=1:xValidationFolds
         testPortion_success = indexPermutation_success(round((fold-
1)*setSize_success)+1:round(fold*setSize_success));
         testPortion_failure=indexPermutation_failure(round((fold-
1)*setSize_failure)+1:round(fold*setSize_failure));
         testPortion=zeros(1, length(testPortion_success)+length(testPortion_failure));
         for n=1:length(testPortion_success)
            testPortion(1, n)=success_index(testPortion_success(n));
         end
         for
m=length(testPortion_success)+1:length(testPortion_success)+length(testPortion_failure)
            testPortion(1, m)=failure_index(testPortion_failure(m-length(testPortion_success)));
         end
         trainPortion = setdiff(1:1:r,testPortion);
         % train SVM classifier here using indices 'trainPortion'
         x_train = x(trainPortion, :);
         y_train = y(trainPortion, :);
         svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k), 'kernel_function',
'rbf', 'rbf_sigma', Sigma_array(1, j), 'method', method1);
         %svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k),  'method', 'SMO');
         group_train=svmclassify(svmstrcture, x_train);
%            %%new
train_correct_rate(k,j)=train_correct_rate(k,j)+100*sum(group_train==y_train)/size(x_train,1);
         % compute the correct rate for test data in this fold
         x_test= x(testPortion,:);
         y_test= y(testPortion,:);
         group=svmclassify(svmstrcture, x_test);
         test_correct_rate(k, j)=test_correct_rate(k,j)+100*sum(group==y_test)/size(x_test,1);
         Pro_test_1=0;
         Pro_test_2=0;
         for l=1:length(y_test)
           if (y_test(l)==1) && (group(l)==0)
              Pro_test_1=Pro_test_1+1;
           else if (y_test(l)==0) && (group(l)==1)
                 Pro_test_2=Pro_test_2+1;
              end
           end
         end
         Pro_test_1=Pro_test_1/length(testPortion_success);
         Pro_test_2=Pro_test_2/length(testPortion_failure);
         optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)+alpha*Pro_test_1+(1-
alpha)*Pro_test_2;
      end % end fold
```

```
        train_correct_rate(k,j)=train_correct_rate(k,j)/xValidationFolds;
        test_correct_rate(k,j)=test_correct_rate(k,j)/xValidationFolds;
        optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)/xValidationFolds;
        if train_correct_rate(k,j)<percentage_bound
          optimazation_criterion_2(k, j)=Inf;
        end
    end % end j
end %end k
train_correct_rate
optimazation_criterion_2
test_correct_rate
[best_rate1, index1]=min(optimazation_criterion_2);
[best_rate2, index2]=min(best_rate1);
best_C=C_array(index1(1, index2))
best_sigma=Sigma_array(index2)
best_rate2

%%
% second searching
C_range_1=max(min_c, log(best_C)-0.5*10):0.5:min(log(best_C)+0.5*10,max_c);
C_array=exp(C_range_1);
Sigma_range_1=max(min_sigma, log(best_sigma)-
0.5*10):0.5:min(log(best_sigma)+0.5*10,max_sigma);
Sigma_array=exp(Sigma_range_1);
C_length=length(C_array);
G_length=length(Sigma_array);

% select optimal gamma using mulitiple-fold cross validation
xValidationFolds = 5;
rand('state',0); randn('state',0);

indexPermutation_success = randperm(number_of_success); % sample from this index set
setSize_success = number_of_success/xValidationFolds;

rand('state',1); randn('state',1);
indexPermutation_failure = randperm(number_of_failure); % sample from this index set
setSize_failure = number_of_failure/xValidationFolds;

% here comes my model selection loop
train_correct_rate=zeros(C_length, G_length);
test_correct_rate=zeros(C_length, G_length);

optimazation_criterion_2=zeros(C_length, G_length);

for k=1:C_length
    for j=1:G_length
```

```
      for fold=1:xValidationFolds
         testPortion_success = indexPermutation_success(round((fold-
1)*setSize_success)+1:round(fold*setSize_success));
         testPortion_failure=indexPermutation_failure(round((fold-
1)*setSize_failure)+1:round(fold*setSize_failure));
         testPortion=zeros(1, length(testPortion_success)+length(testPortion_failure));
         for n=1:length(testPortion_success)
            testPortion(1, n)=success_index(testPortion_success(n));
         end
         for
m=length(testPortion_success)+1:length(testPortion_success)+length(testPortion_failure)
            testPortion(1, m)=failure_index(testPortion_failure(m-length(testPortion_success)));
         end
         trainPortion = setdiff(1:1:r,testPortion);
         % train SVM classifier here using indices 'trainPortion'
         x_train = x(trainPortion, :);
         y_train = y(trainPortion, :);
         svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k), 'kernel_function',
'rbf', 'rbf_sigma', Sigma_array(1, j), 'method', method2);
         %svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k),  'method', 'SMO');
         group_train=svmclassify(svmstrcture, x_train);
%           %%new
train_correct_rate(k,j)=train_correct_rate(k,j)+100*sum(group_train==y_train)/size(x_train,1);
         % compute the correct rate for test data in this fold
         x_test= x(testPortion,:);
         y_test= y(testPortion,:);
         group=svmclassify(svmstrcture, x_test);
         test_correct_rate(k, j)=test_correct_rate(k,j)+100*sum(group==y_test)/size(x_test,1);
         Pro_test_1=0;
         Pro_test_2=0;
         for l=1:length(y_test)
          if (y_test(l)==1) && (group(l)==0)
             Pro_test_1=Pro_test_1+1;
          else if (y_test(l)==0) && (group(l)==1)
                Pro_test_2=Pro_test_2+1;
             end
          end
         end
         Pro_test_1=Pro_test_1/length(testPortion_success);
         Pro_test_2=Pro_test_2/length(testPortion_failure);
         optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)+alpha*Pro_test_1+(1-
alpha)*Pro_test_2;

      end % end fold
      train_correct_rate(k,j)=train_correct_rate(k,j)/xValidationFolds;
      test_correct_rate(k,j)=test_correct_rate(k,j)/xValidationFolds;
```

```
      optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)/xValidationFolds;
      if train_correct_rate(k,j)<percentage_bound
        optimazation_criterion_2(k, j)=Inf;
      end
   end % end j
end %end k
train_correct_rate
optimazation_criterion_2
test_correct_rate
[best_rate1, index1]=min(optimazation_criterion_2);
[best_rate2, index2]=min(best_rate1);
best_C=C_array(index1(1, index2))
best_sigma=Sigma_array(index2)
best_rate2


%%
% third searching
C_range_1=max(min_c, log(best_C)-0.25*10):0.25:min(log(best_C)+0.25*10,max_c);
C_array=exp(C_range_1);
Sigma_range_1=max(min_sigma, log(best_sigma)-
0.25*10):0.25:min(log(best_sigma)+0.25*10,max_sigma);
Sigma_array=exp(Sigma_range_1);
C_length=length(C_array);
G_length=length(Sigma_array);

% select optimal gamma using mulitiple-fold cross validation
xValidationFolds = 5;
rand('state',0); randn('state',0);

indexPermutation_success = randperm(number_of_success); % sample from this index set
setSize_success = number_of_success/xValidationFolds;

rand('state',1); randn('state',1);
indexPermutation_failure = randperm(number_of_failure); % sample from this index set
setSize_failure = number_of_failure/xValidationFolds;

% here comes my model selection loop
train_correct_rate=zeros(C_length, G_length);
test_correct_rate=zeros(C_length, G_length);

optimazation_criterion_2=zeros(C_length, G_length);


for k=1:C_length
   for j=1:G_length
```

```
      for fold=1:xValidationFolds
         testPortion_success = indexPermutation_success(round((fold-
1)*setSize_success)+1:round(fold*setSize_success));
         testPortion_failure=indexPermutation_failure(round((fold-
1)*setSize_failure)+1:round(fold*setSize_failure));
         testPortion=zeros(1, length(testPortion_success)+length(testPortion_failure));
         for n=1:length(testPortion_success)
            testPortion(1, n)=success_index(testPortion_success(n));
         end
         for
m=length(testPortion_success)+1:length(testPortion_success)+length(testPortion_failure)
            testPortion(1, m)=failure_index(testPortion_failure(m-length(testPortion_success)));
         end
         trainPortion = setdiff(1:1:r,testPortion);
         % train SVM classifier here using indices 'trainPortion'
         x_train = x(trainPortion, :);
         y_train = y(trainPortion, :);
         svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k), 'kernel_function',
'rbf', 'rbf_sigma', Sigma_array(1, j), 'method', method2);
         %svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k),  'method', 'SMO');
         group_train=svmclassify(svmstrcture, x_train);
% %           %new
train_correct_rate(k,j)=train_correct_rate(k,j)+100*sum(group_train==y_train)/size(x_train,1);
         % compute the correct rate for test data in this fold
         x_test= x(testPortion,:);
         y_test= y(testPortion,:);
         group=svmclassify(svmstrcture, x_test);
         test_correct_rate(k, j)=test_correct_rate(k,j)+100*sum(group==y_test)/size(x_test,1);
         Pro_test_1=0;
         Pro_test_2=0;
         for l=1:length(y_test)
          if (y_test(l)==1) && (group(l)==0)
             Pro_test_1=Pro_test_1+1;
          else if (y_test(l)==0) && (group(l)==1)
                Pro_test_2=Pro_test_2+1;
             end
          end
         end
         Pro_test_1=Pro_test_1/length(testPortion_success);
         Pro_test_2=Pro_test_2/length(testPortion_failure);
         optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)+alpha*Pro_test_1+(1-
alpha)*Pro_test_2;
      end % end fold
      train_correct_rate(k,j)=train_correct_rate(k,j)/xValidationFolds;
      test_correct_rate(k,j)=test_correct_rate(k,j)/xValidationFolds;
      optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)/xValidationFolds;
```

```
      if train_correct_rate(k,j)<percentage_bound
        optimazation_criterion_2(k, j)=Inf;
      end
   end % end j
end %end k
train_correct_rate
optimazation_criterion_2
test_correct_rate
[best_rate1, index1]=min(optimazation_criterion_2);
[best_rate2, index2]=min(best_rate1);
best_C=C_array(index1(1, index2))
best_sigma=Sigma_array(index2)
best_rate2

%%
% fourth searching
C_range_1=max(min_c, log(best_C)-0.1*10):0.1:min(log(best_C)+0.1*10,max_c);
C_array=exp(C_range_1);
Sigma_range_1=max(min_sigma, log(best_sigma)-
0.01*10):0.01:min(log(best_sigma)+0.01*10,max_sigma);
Sigma_array=exp(Sigma_range_1);
C_length=length(C_array);
G_length=length(Sigma_array);

% select optimal gamma using mulitiple-fold cross validation
xValidationFolds = 5;
rand('state',0); randn('state',0);

indexPermutation_success = randperm(number_of_success); % sample from this index set
setSize_success = number_of_success/xValidationFolds;

rand('state',1); randn('state',1);
indexPermutation_failure = randperm(number_of_failure); % sample from this index set
setSize_failure = number_of_failure/xValidationFolds;

% here comes my model selection loop
train_correct_rate=zeros(C_length, G_length);
test_correct_rate=zeros(C_length, G_length);

optimazation_criterion_2=zeros(C_length, G_length);

for k=1:C_length
   for j=1:G_length
      for fold=1:xValidationFolds
         testPortion_success = indexPermutation_success(round((fold-
1)*setSize_success)+1:round(fold*setSize_success));
```

```
        testPortion_failure=indexPermutation_failure(round((fold-
1)*setSize_failure)+1:round(fold*setSize_failure));
        testPortion=zeros(1, length(testPortion_success)+length(testPortion_failure));
        for n=1:length(testPortion_success)
           testPortion(1, n)=success_index(testPortion_success(n));
        end
        for
m=length(testPortion_success)+1:length(testPortion_success)+length(testPortion_failure)
           testPortion(1, m)=failure_index(testPortion_failure(m-length(testPortion_success)));
        end
        trainPortion = setdiff(1:1:r,testPortion);
        % train SVM classifier here using indices 'trainPortion'
        x_train = x(trainPortion, :);
        y_train = y(trainPortion, :);
        svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k), 'kernel_function',
'rbf', 'rbf_sigma', Sigma_array(1, j), 'method', method2);
        group_train=svmclassify(svmstrcture, x_train);
%           %%new
train_correct_rate(k,j)=train_correct_rate(k,j)+100*sum(group_train==y_train)/size(x_train,1);
        % compute the correct rate for test data in this fold
        x_test= x(testPortion,:);
        y_test= y(testPortion,:);
        group=svmclassify(svmstrcture, x_test);
        test_correct_rate(k, j)=test_correct_rate(k,j)+100*sum(group==y_test)/size(x_test,1);
        Pro_test_1=0;
        Pro_test_2=0;
        for l=1:length(y_test)
          if (y_test(l)==1) && (group(l)==0)
             Pro_test_1=Pro_test_1+1;
          else if (y_test(l)==0) && (group(l)==1)
                Pro_test_2=Pro_test_2+1;
             end
          end
        end
        Pro_test_1=Pro_test_1/length(testPortion_success);
        Pro_test_2=Pro_test_2/length(testPortion_failure);
        optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)+alpha*Pro_test_1+(1-
alpha)*Pro_test_2;
      end % end fold
      train_correct_rate(k,j)=train_correct_rate(k,j)/xValidationFolds;
      test_correct_rate(k,j)=test_correct_rate(k,j)/xValidationFolds;
      optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)/xValidationFolds;
      if train_correct_rate(k,j)<percentage_bound
        optimazation_criterion_2(k, j)=Inf;
      end
   end % end j
```

```
end %end k
train_correct_rate
optimazation_criterion_2
test_correct_rate
[best_rate1, index1]=min(optimazation_criterion_2);
[best_rate2, index2]=min(best_rate1);
best_C=C_array(index1(1, index2))
best_sigma=Sigma_array(index2)
best_rate2

% fifth searching
C_range_1=max(min_c, log(best_C)-0.05*10):0.05:min(log(best_C)+0.05*10,max_c);
C_array=exp(C_range_1);
Sigma_range_1=max(min_sigma, log(best_sigma)-
0.05*10):0.05:min(log(best_sigma)+0.05*10,max_sigma);
Sigma_array=exp(Sigma_range_1);
C_length=length(C_array);
G_length=length(Sigma_array);

% select optimal gamma using mulitiple-fold cross validation
xValidationFolds = 5;
rand('state',0); randn('state',0);

indexPermutation_success = randperm(number_of_success); % sample from this index set
setSize_success = number_of_success/xValidationFolds;

rand('state',1); randn('state',1);
indexPermutation_failure = randperm(number_of_failure); % sample from this index set
setSize_failure = number_of_failure/xValidationFolds;

% here comes my model selection loop
train_correct_rate=zeros(C_length, G_length);
test_correct_rate=zeros(C_length, G_length);

optimazation_criterion_2=zeros(C_length, G_length);


for k=1:C_length
   for j=1:G_length
      for fold=1:xValidationFolds
         testPortion_success = indexPermutation_success(round((fold-
1)*setSize_success)+1:round(fold*setSize_success));
         testPortion_failure=indexPermutation_failure(round((fold-
1)*setSize_failure)+1:round(fold*setSize_failure));
         testPortion=zeros(1, length(testPortion_success)+length(testPortion_failure));
         for n=1:length(testPortion_success)
```

```
            testPortion(1, n)=success_index(testPortion_success(n));
        end
        for
m=length(testPortion_success)+1:length(testPortion_success)+length(testPortion_failure)
            testPortion(1, m)=failure_index(testPortion_failure(m-length(testPortion_success)));
        end
        trainPortion = setdiff(1:1:r,testPortion);
        % train SVM classifier here using indices 'trainPortion'
        x_train = x(trainPortion, :);
        y_train = y(trainPortion, :);
        svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k), 'kernel_function',
'rbf', 'rbf_sigma', Sigma_array(1, j), 'method', method2);
        %svmstrcture=svmtrain(x_train, y_train, 'boxconstraint', C_array(1,k),  'method', 'SMO');
        group_train=svmclassify(svmstrcture, x_train);
% %           %%new
train_correct_rate(k,j)=train_correct_rate(k,j)+100*sum(group_train==y_train)/size(x_train,1);
        % compute the correct rate for test data in this fold
        x_test= x(testPortion,:);
        y_test= y(testPortion,:);
        group=svmclassify(svmstrcture, x_test);
        test_correct_rate(k, j)=test_correct_rate(k,j)+100*sum(group==y_test)/size(x_test,1);
        Pro_test_1=0;
        Pro_test_2=0;
        for l=1:length(y_test)
          if (y_test(l)==1) && (group(l)==0)
             Pro_test_1=Pro_test_1+1;
          else if (y_test(l)==0) && (group(l)==1)
                Pro_test_2=Pro_test_2+1;
             end
          end
        end
        Pro_test_1=Pro_test_1/length(testPortion_success);
        Pro_test_2=Pro_test_2/length(testPortion_failure);
        optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)+alpha*Pro_test_1+(1-
alpha)*Pro_test_2;
      end % end fold
      train_correct_rate(k,j)=train_correct_rate(k,j)/xValidationFolds;
      test_correct_rate(k,j)=test_correct_rate(k,j)/xValidationFolds;
      optimazation_criterion_2(k, j)= optimazation_criterion_2(k, j)/xValidationFolds;
      if train_correct_rate(k,j)<percentage_bound
        optimazation_criterion_2(k, j)=Inf;
      end
   end % end j
end %end k
train_correct_rate
optimazation_criterion_2
```

```
test_correct_rate
[best_rate1, index1]=min(optimazation_criterion_2);
[best_rate2, index2]=min(best_rate1);
best_C=C_array(index1(1, index2))
best_sigma=Sigma_array(index2)
best_rate2
train_correct_rate =train_correct_rate(index2)
corresponding_test_correct_rate=test_correct_rate(index1(1, index2), index2)

eeg_svmstrcture=svmtrain(x, y, 'boxconstraint', best_C, 'kernel_function', 'rbf', 'rbf_sigma',
best_sigma, 'method', method2);
end
```

Matlab Code to Test the EEG Data using SVM:

```
close all;
clear all;
clc;

powerdata_all = csvread('left and right power data .csv') ;
decision_data_all = [ones(150,1);zeros(150,1)];
defalt = [1:300];
test_range = [randsample(1:150,50) randsample(151:300,50)];
x_test = powerdata_all(test_range,:);
y_test = decision_data_all(test_range);

train_range = setdiff(defalt,test_range);
training = powerdata_all(train_range,:);
decision = decision_data_all(train_range);
alpha = 0.5;
%% start calculating
method1 ='LS' ;
method2 ='LS' ;
bond    = 80;
[ eeg_svmstructure ]=train_rbfsvm_eeg_Jay(training,decision,method1,method2,bond,alpha);
group_train=svmclassify(eeg_svmstructure, training);

% train_correct_rate=100*sum(group_train==decision)/size(training,1)

group=svmclassify(eeg_svmstructure, x_test);
test_correct_rate=100*sum(group==y_test)/size(x_test,1)
Pro_test_1=0;
Pro_test_2=0;
for l=1:length(y_test)
   if (y_test(l)==1) && (group(l)==0)
      Pro_test_1=Pro_test_1+1;
   else if (y_test(l)==0) && (group(l)==1)
         Pro_test_2=Pro_test_2+1;
      end
   end
end
Pro_test_1=Pro_test_1/sum(y_test)
Pro_test_2=Pro_test_2/(length(y_test)-sum(y_test))
conMat = confusionmat(y_test, group)
```