Imperial College
London

EE2-PRJ Brain-Computer Interface
Final Technical Report

2nd Year Group Project
Low-Cost Brain Computer Interface for Motor-Impaired People

Group 8

Members:	Javier Hernandez (00943425)
	Nicolas Perez (00931920)
	Aaron Low (00920886)
	Vinay Maniam (00981045)
	Jun Chan (00861477)
	Jorn Voegtli (00958215)
	Samuel Zatland (00818847)
Supervisor:	Dr. Bruno Clerckx
Year:	2nd Year
Course:	Electrical & Electronic Engineering
Submission Date:	13 March 2016

Contents

Abstract

The aim of this report is to explain the method used to implement a Brain Computer Interface (BCI) for motor impaired patients. These patients have become incapable of moving due to various reasons such as a stroke or paralysis. This system that will aid them to communicate consists of a screen with a user-interface that will display a keyboard with letters flashing periodically. The user will be able to select the letters by simply staring at the desired letter for a short period of time. The letter will then appear on the screen and at the same time, some suggested words will be displayed. The letter detection is based on a phenomenon called P300 that consists of detecting peaks on the electroencephalogram (EEG) when the subject receives an unexpected stimulus, such as a letter flashing.

Since the main aim of this project is to determine if it is possible to implement a low-cost BCI speller, open-source material was used throughout the whole development. Effectively, two prototypes were built for testing. One of them is a headset developed to amplify, filter and record EEG signals. The other one is the software for the interface and signal processing of the EEG signal. Both are completely independent, the first one can operate without the second, and vice versa.

The headset has proven to be able to detect and amplify EEG signals when tested using the oscilloscope. Nevertheless, it still has compatibility issues with several operating systems and thus, its efficacy could not be completely measured. On the other hand, the software has been tested using a different headset available at Imperial College. The results were positive as an accuracy of 60% was achieved (with no word predictor or speller), with a margin of improvement if certain parameters were changed (the positioning of the electrodes for example).

The general conclusion is that there is no reason to believe that a full low-cost BCI speller is not feasible. There is still testing and improvements to be done, but the results so far have shown a high reliability at this stage.

1. Introduction and Background

The upcoming era is one of great technological advances that will change life as it is known. Whilst developments that will allow mankind to interact with artificial intelligence will be seen, it is also expected that the average life expectancy and life quality will also significantly increase. Nowadays however, although people can reach an older age with improved living conditions, certain medical diseases have not been cured yet. For instance, neurological conditions such as multiple sclerosis or Parkinson's disease may imply the partial or total loss of functionality of the body, and so far, no solution to these conditions has been discovered.

People that suffer from such neurological diseases may often be affected up to such a point that they suffer from motor impairment. There are different symptoms of motor impairment: weakness due to a spinal cord injury, fatigue in multiple sclerosis, impaired sensation and movement after a stroke or impaired balance from Parkinson's disease. In some cases, several of these impairments may co-exist in a single person, which might disable them from walking or talking. Motor impairment is quite prevalent around the globe. For example, in 2003, approximately 350,000 Australians experienced a stroke, from which half of these patients could not recover properly. (NeuRA, 2016)

This major health problem calls for a solution. In the special case of patients with a total loss of body functionality, technology has to aid their communication with the outside world. Such a solution is being implemented using a Brain Computer Interface (BCI). A BCI is a system that is used to recognise brain signal patterns and thus enable communication or control without movement. This device detects specific patterns of the user's brain activity that reflect a certain message that the user wants to send, such as spelling. (Guger, Vaughan & Allison, 2014) These systems normally work following five consecutive stages: signal acquisition, pre-processing or signal enhancement, feature extraction, classification and then control interface. (Nicolas-Alonso & Gomez-Gil, 2012) One of the ways that a BCI can work is via the flickering of images or letters, which will stimulate the user's response in the visual cortex. These signals will then be recorded via electrodes placed on the scalp and used to translate the user's intentions.

Different disciplines such as Medicine or Biomedical Engineering have been investigating how to create reliable BCI systems, however the current solutions are extremely expensive, from £500 to £1000 or even more. (OpenBCI, 2016) These high prices make these products unaffordable for some patients as well as the facilities where patients are treated. For this reason, the aim of this project is to develop a low-cost BCI system that will thus be affordable for these patients.

2. Design Criteria

Any product that is going to be designed and later produced needs to follow certain specifications and design criteria, mostly specified by the future user, in order to be as effective as possible. To do this, the group decided to specify these criteria in the Product Design Specification (Please refer to Appendix A).

The specifications for an electronic device that would be used on a daily basis by motor impaired people had to address the social, economic and environmental contexts in which it would be applied as a product. It was therefore decided that the most important design criteria to meet were the following: safety, performance, aesthetics and appearance, reliability, installation, product cost, packing, documentation and ergonomics.

First of all, certain safety measures have to be taken into account. Firstly, since the product is powered by electricity, an electrical barrier that isolates the user and the device needs to be present. In addition, it would not be recommended for people who suffer from epilepsy since its continuous use for long periods of time may result in unpleasant side effects. For this reason, a well-documented pack is necessary to warn about these safety measures. These documents would include an installation pack, where clear instructions on how to use and test the device would be included. This installation manual would need to be user-friendly, including many images and big text so that the utilisation of the device is as simple as possible.

Other criteria that have to be taken into account are reliability and performance, since both of them are closely linked. The reliability of the product is crucial, since the whole aim of the project is to effectively help a motor impaired person to communicate via a keyboard on a computer screen. Therefore, the output of letters that correspond to the targeted letters that the user is intending to print, would need to be at least 90% accurate for a final product as the group is aware that certain conditions, such as excessive noise or fatigue in the user may cause errors when processing the brain signal. For this reason, with a 90% accuracy, even if one letter is missed out of a 10 letter word for example, the assistant that will be aiding the user should be able to deduce the intended letter. Such reliability can only be possible if the performance of the device is near to optimum. For this reason, the project was downscaled as a whole to make it manageable. The original idea of using two signal phenomena was reduced to just focusing on making one of these signal-processing methods to work, therefore to be as reliable as possible in this first initial stage. In addition to this and in order to increase the typing speed and reliability, a word predictor system will also be added to the keyboard (similar to word predictors in mobile phone keyboards). This is a very powerful feature that has not been exploited in BCI spellers so far.

In terms of ergonomics and the aesthetics and appearance of the product, the group has decided to modify the criteria in this area from what was presented in the Interim Report due to the fact that a product that is aesthetically pleasing at first sight and is comfortable for the user will be a better overall solution. For this reason, certain components of the product have been refined. For instance, the headset must be lightweight, thus plastic printing fibre is used to make it. It could also be available in different sizes and colours depending on the user's preference and head size. Moreover, the entire electrical circuit will now be enclosed in a tightly sealed laser-cut box that could be personalized to include the user's name, as well as the option to be made from different materials such as wood or special plastic fibre. This box would be part of our first prototype solution, as the aim would be to develop a final product that incorporates all the electrodes and circuit into the headset, thus reducing space and weight. Despite that, this final headset design would allow for customisability too.

In addition, the user-interface has now been developed by the group in order for it to work both in Windows and Linux operating systems, and the colours that were being used have changed. Now, the background colour is blue, which upon testing has been determined to be the least tiring for the eyes after long periods of time (Neuroeng, 2012), and the letters are now flashed in yellow. The software for the user-interface has been programmed in a way that certain features, such as letter size or colour, could be easily modified by the user.

Lastly, one of the main objectives of the project is reducing the cost of the entire BCI. The current market options oscillate around a price of £500 for the most basic options and can rise up to £1000 (OpenBCI, 2016). For this reason, the designing of the whole device has to revolve around the idea that the total cost has to be minimal so that many facilities that take care of motor-impaired patients can buy the product. The total price is therefore estimated to be around £250 (OpenEEG, 2015), which is the budget that was allocated to the group. The total price would also include packing, which would be a tightly sealed box with fitted Styrofoam in order to prevent dust from corrupting the electronic circuit or damaging the headset with the placed electrodes.

3. Concept Designs Considered

For the design two different modules need to be considered: the hardware equipment to record EEG signals and the software to process the signals and interface with the user.

3.1. Hardware: Low Cost EEG Equipment

3.1.1. Concept 1: OpenEEG

OpenEEG is a fully open project developed in 2003 for EEG recording. It allows the construction of a headset with up to 6 differential channels (i.e. up to 12 electrodes, two per channel). Differential channels provide a higher Signal-to-Noise Ratio (SNR) and spatial resolution than the conventional channels (Sheng-Feng Yen et al. 2009). In addition, the electrodes may be placed anywhere on the scalp and thus may be placed wherever the signal is better for the product's purposes. Its cost varies depending on the type of electrodes chosen and the number of desired channels. For example, for a 4-channel device the cost would be around £250 (OpenEEG, 2015). On the other hand, OpenEEG is a project developed more than 10 years ago and although it has a moderately active community, it does not have any costumer or User Support Service and thus in case of running into unexpected problems, a solution might not be easily found.

Apart from that, this design also needs a headset structure design. This headset needs to be able to reach the targeted head positions. The headset would need to be light and easy to fit. This can be done using any of the 3D printed headsets shown on the picture below. A cabling system would also be needed along with an electrode socket array and a box containing all the electronics, all of which are fully described in the Concept Development section.

This concept should be only considered for prototyping since a final product would require a complete update of the electronics (designed in 2003) in order to make it cheaper and smaller.

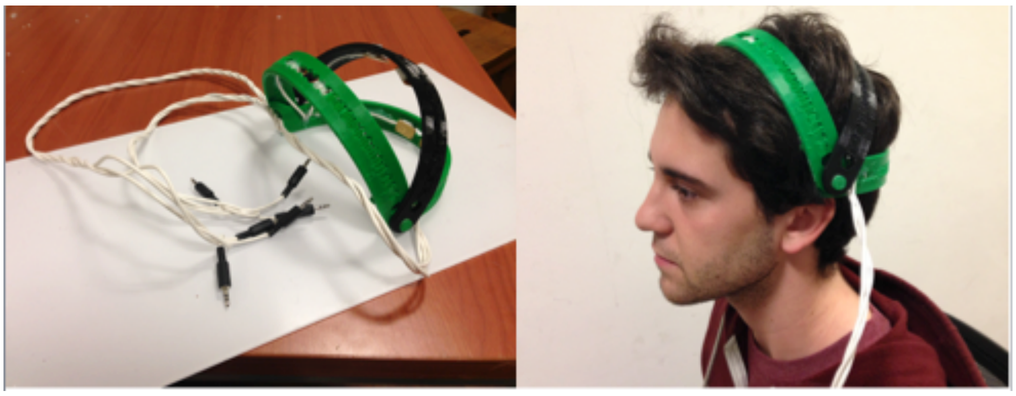


Figure 1: The group's self-made headset using OpenEEG active electrodes

3.1.2. Concept 2: Emotiv Epoc Headset

Emotiv has developed an EEG headset that allows signal monitoring using up to 14 channels. However, Emotiv Epoc's electrodes are fixed and thus they cannot be placed in the optimal head locations for P300 detection (more information about P300 in section 3.2). Besides, Emotiv Epoc channels are conventional EEG channels (i.e. one electrode per channel), not differential which means that the acquired signal is potentially worse than that achieved using a differential headset. (Sheng-Feng Yen et al. 2009) Thus, using more channels would be advised in order to get a better signal. In addition, the price of this product is £505 (Emotiv, 2015), which is above the proposed total budget for the product. On the other hand, Emotiv Epoc is fully documented and it has a Customer Service currently running which means it would be easier to deal with unexpected problems concerning the hardware. It is also a product with a large community behind it that has proved that this device works in a wide range of applications.

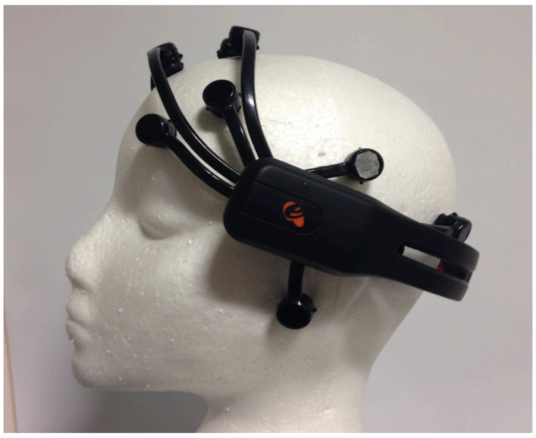


Figure 2: Emotiv Epoc Headset at Imperial College

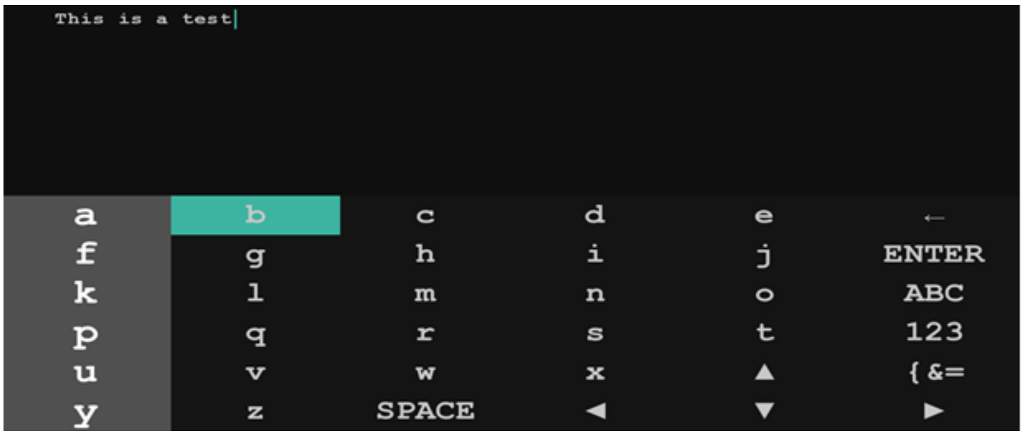
3.2. Software: Signal Processing and Interface

3.2.1. Concept 1: P300 based Interface

The P300 technique involves the use of a matrix of letters and characters on the screen, with rows and columns being flashed in a seemingly random manner. As stated on Neuroeng (2012), when a symbol that the user is focusing on flashes, the unexpected visual stimulus generates an electrophysiological response (EVP). This EVP is represented by a peak in electric potential of the brain waves over the visual cortex, roughly 300ms after it is observed, hence the name P300. After several sequences of flashes, each symbol in the matrix would have a distinct and distinguishable array of brain wave peaks that can then be used to tell the users what letter they were trying to spell. Since this technique relies only on coordinating the flash and the response to that flash, there is potentially no limit to the number of symbols that can be detected. By using this technique, a bit rate of up to 20-25 bits/min can be achieved. (Nicolas-Alonso & Gomez-Gil, 2012) Nevertheless, the group design would combine this P300 technique with a speller and word predictor that will significantly improve the bit rate.

User training requires very little skill, and takes under 20 minutes. Moreover, by varying flashing frequency and number of flashes per decision, the time taken to spell one letter lies in the range of 10<t<50 seconds. Combined with the predictive text algorithm, the average time taken to spell a word can be configured to be as little as 1 minute.

The interface itself would be an on-screen keyboard which will randomly flash letters. The user would have to focus on looking at a particular letter in order to type it on the screen. After a few letters have been entered, three predicted words would appear on the matrix.


This is a test

a	b	c	d	e	←
f	g	h	i	j	ENTER
k	l	m	n	o	ABC
p	q	r	s	t	123
u	v	w	x	▲	{&=
y	z	SPACE	◀	▼	▶

Figure 3: P300 Interface, using Pygame

3.2.2. Concept 2: Steady State VEP based Interface

Visually evoked potentials (VEPs) follow a different neurophysiological phenomenon to P300. VEPs are brain wave cadences in the visual cortex, parameterised by frequency, morphology and field. (Nicolas-Alonso & Gomez-Gil, 2012) For this BCI Speller, only frequency will be varied. Whenever the user is stimulated by a flashing image at a certain frequency (between 8 and 20 Hz) a peak on the power spectral density of the EEG signal, read on the visual cortex, appears.

In contrast, the time taken to make a decision in SSVEP is on average faster than P300, falling in the range of 10 to 20 seconds and provides up to 60-100 bits/min. (Nicolas-Alonso & Gomez-Gil, 2012) The downside to this strategy is that none of the flashing frequencies can be an integer multiple of the other (due to harmonics interference), and also needs to be a factor of 60 (computer screen refresh rate). In addition, these frequencies must be greater than 6Hz or the brainwave will be indistinguishable from regular brain activity. This limits the number of options to 12, 15 and 20Hz.

The interface will implement this procedure using three flashing boxes on a screen, each flashing at a different frequency. When the user focuses on a flashing box, a signal is observed in the visual cortex, the frequency of which corresponds to that of the flashing box. To this end, it is a simple matter to use a band-pass filter and determine which box the user is focusing on.

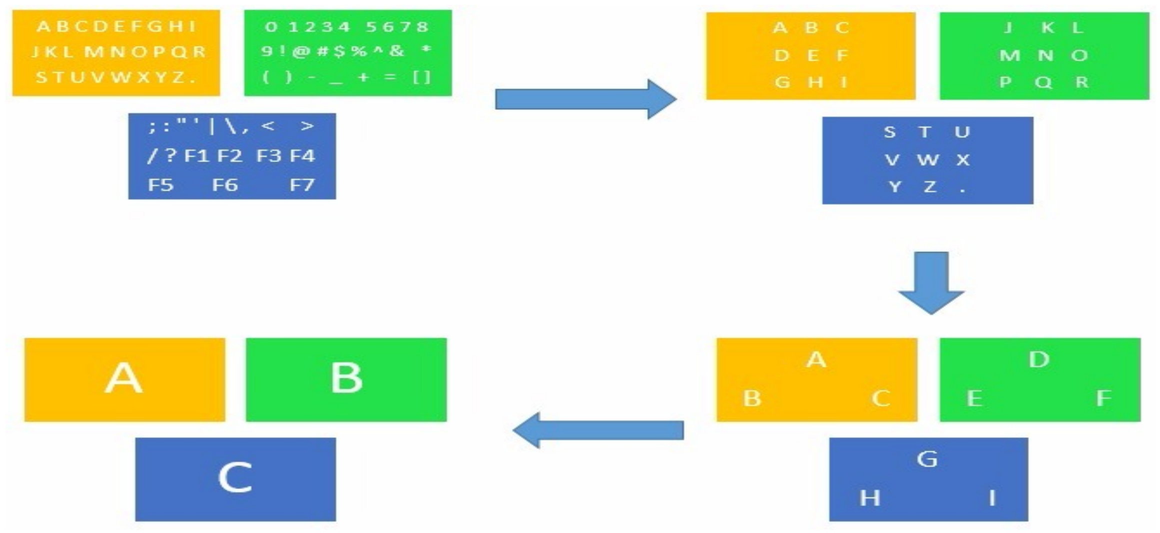
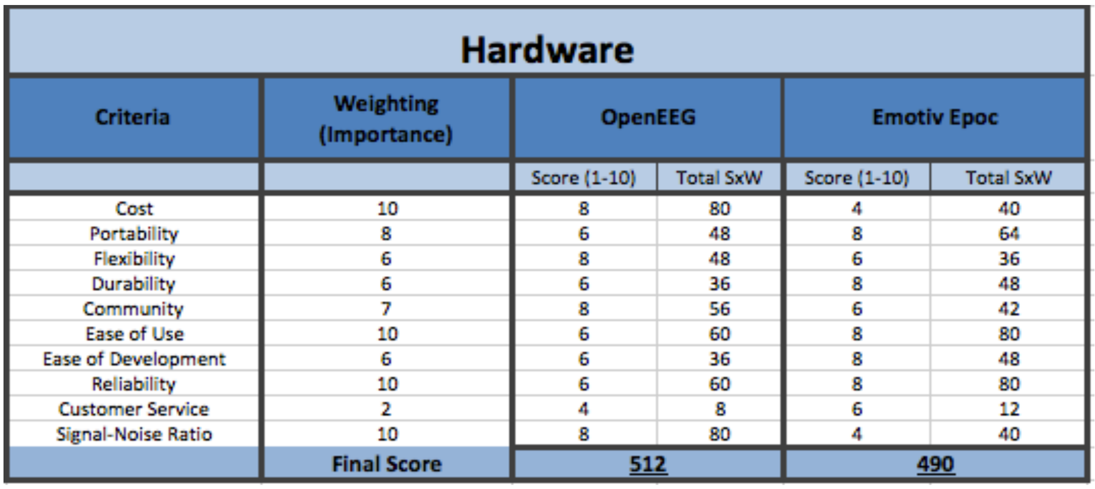


Figure 4: Four screen captures of the SSVEP based Interface running when letter A is being targeted.

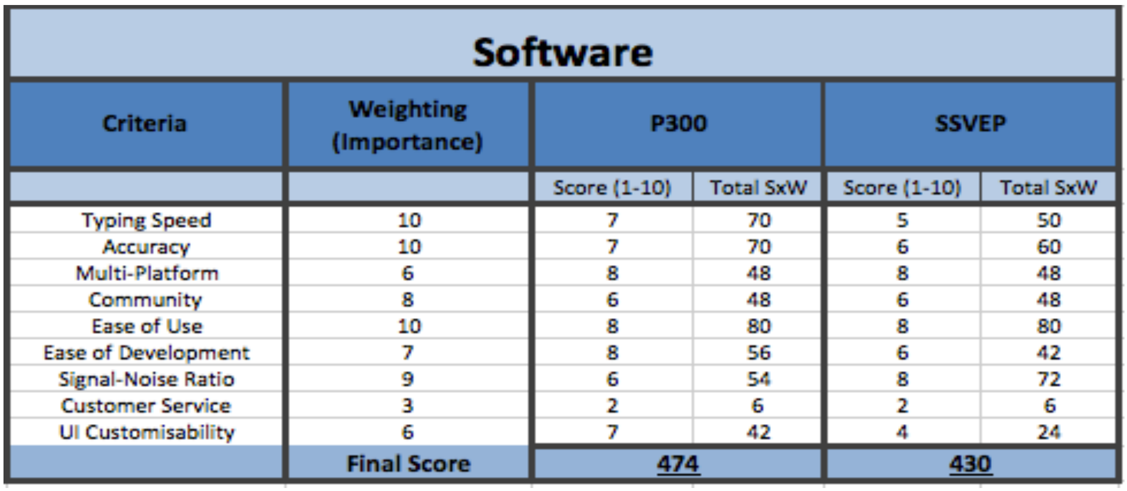
4. Concept Selection

The concept selection method was done with a weighting and rating matrix. In this matrix, the most important features to be considered are listed, along with a weighting from 1-10 on each. The different criteria will be scored in this range depending on their importance for the overall design. In the end, there will be a weighted total for each concept, and thus the concept with the most points will be the preferred choice. This method is commonly used for assessing the relative merits of a range of options. In the group's case, there will be two of these matrices, one for hardware and one for signal-acquiring software. The matrices shown below illustrate how the selection process was performed using this method.



Hardware					
Criteria	**Weighting (Importance)**	**OpenEEG**		**Emotiv Epoc**	
		Score (1-10)	Total SxW	Score (1-10)	Total SxW
Cost	10	8	80	4	40
Portability	8	6	48	8	64
Flexibility	6	8	48	6	36
Durability	6	6	36	8	48
Community	7	8	56	6	42
Ease of Use	10	6	60	8	80
Ease of Development	6	6	36	8	48
Reliability	10	6	60	8	80
Customer Service	2	4	8	6	12
Signal-Noise Ratio	10	8	80	4	40
	Final Score	**512**		**490**	

Table 1: Scoring and Weighting Matrix for Hardware



Software					
Criteria	Weighting (Importance)	P300		SSVEP	
		Score (1-10)	Total SxW	Score (1-10)	Total SxW
Typing Speed	10	7	70	5	50
Accuracy	10	7	70	6	60
Multi-Platform	6	8	48	8	48
Community	8	6	48	6	48
Ease of Use	10	8	80	8	80
Ease of Development	7	8	56	6	42
Signal-Noise Ratio	9	6	54	8	72
Customer Service	3	2	6	2	6
UI Customisability	6	7	42	4	24
	Final Score	474		430	

Table 2: Scoring and Weighting Matrix for Software

As a result of the matrix selection, OpenEEG will be used for the final prototype. However, since the group was given access to Emotiv Epoc at Imperial College, it will be used for software testing purposes while the OpenEEG headset is being built. With respect to the headset structure it was decided that the WalkEEG Headset Structure (Thingiverse, 2016) would be used since it is easier to customise and it is fully open source.

As for software, according to the matrix selection, the P300-based interface will be implemented. Despite the fact that SSVEP detection is faster than P300, in this particular application there is a trade-off between the typing speed and the number of letters that can be used. For that reason, the P300 system is faster in the end. Thus, a P300-based interface will be used. However, there is the possibility of implementing a hybrid system that uses both, P300 for the letter and SSVEP for predicted words while typing. This way the typing speed would significantly increase. This was the original idea of the project, but it had to be removed for overall feasibility.

5. Concept Development

5.1. Hardware Development

As stated in the Concept Design section, the aim is to build a 4-channel EEG Headset from the OpenEEG project. This headset will include the following parts, some of which can be acquired and some that have to be designed by the group:

Developed by the group:

- The cap or headset structure to place the electrodes
- Electrode socket array
- Low-noise cabling connecting each part
- Casing containing all the electronics and power supply

Acquired from electronics supplier Olimex:

- Two EEG Olimex Amplifier Boards
- One EEG Olimex Digital Board
- 6 Active Olimex Electrodes

A computer is also required to process the signal and interface with the user. An AVR programmer is also needed to program the ATMega-8 microcontroller on the digital board. (Refer to Appendix E for the whole code)

5.1.1. Headset Structure

As stated before, the *WalkEEG Headset* structure will be used. It can be printed at the Imperial College 3D printing facilities. Two different colours were chosen in order to make it look more appealing. The idea behind this is to allow the users to choose their preferred colours for the headset. The electrodes are attached following the most sensitive places on the scalp in order to detect P300 signals (Please refer to Appendix F for an image of the positioning of electrodes on the scalp).

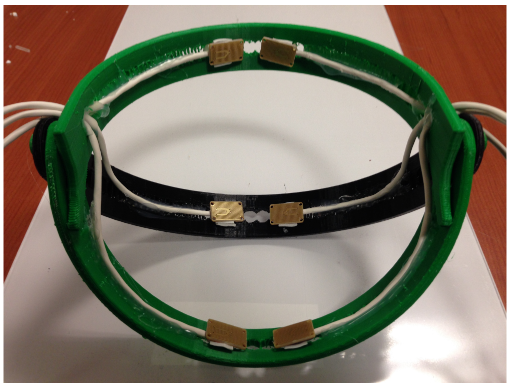


Figure 5: Differential electrode positioning in the headset for optimum P300 detection

5.1.2. Electrode Socket Array

This part of the design consists on developing an effective way of connecting the 3.5mm socket of the electrodes to the board.

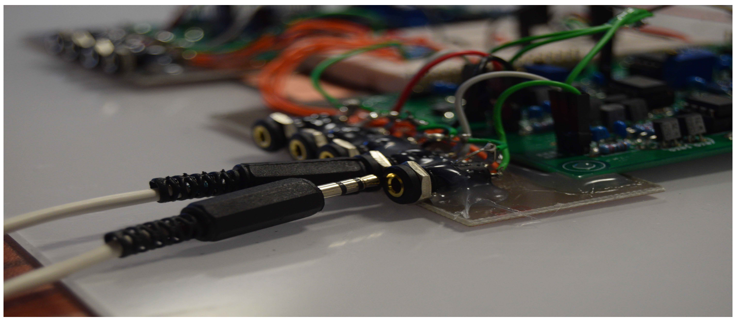


Figure 6: 3.5 mm electrode socket array, which is the input to the board

According to the electrode supplier, Olimex, each electrode must be connected in the following way:

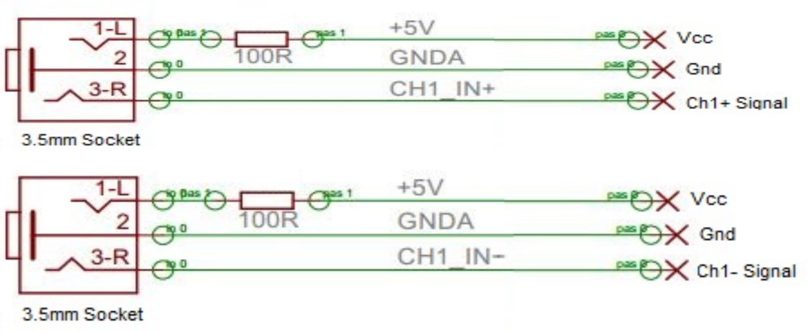


Figure 7: Schematic of Channel 1 Differential Electrodes inputs $Ch1+$ *and* $Ch1-$

This means that two of the inputs of the electrode are just for powering the device, and only the third one carries the signal. The signal is very prone to noise since it is directly taken from the scalp and moderately amplified by the electrode circuitry. This means that the cable to be used to connect the signal input must be as short and close to the board as possible. For Vcc and Gnd connection this is not so relevant since any noise on the power inputs will be present on both differential electrodes. The final signal for a channel is obtained by subtracting the signal present on each electrode, thus the noise due to the power supply will cancel.

The 100Ω resistors and the Vcc and Gnd power rails will be set on a breadboard in between each amplifier board. Despite the fact that a small noise fluctuation on the power rail is not important in terms of the acquired signal, a copper plate will be used below the breadboard to connect the Gnd connection since the voltage at which the electrodes operate must be kept constant. Finally, the electrode socket array is set as shown:

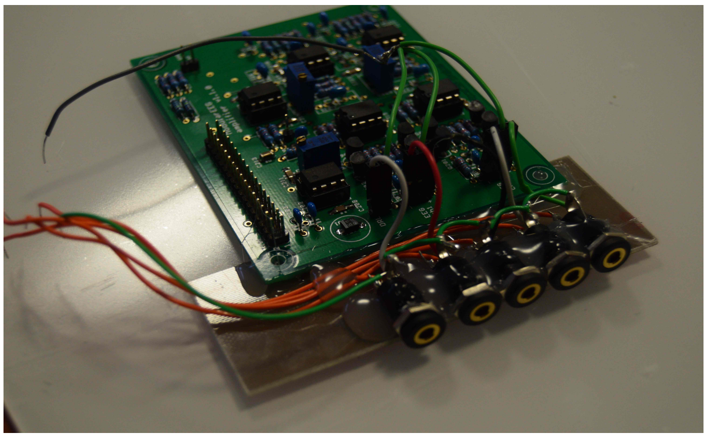


Figure 8: Electrode socket array

This design makes sure that the sockets are as close to the EEG Amplifier Boards as possible. The 3.5mm socket can be glued to a wood or plastic shaft using a glue gun. The socket can be glued to the shaft since the glue is non-conductive.

5.1.3. Casing for the Electronics and Boards Cabling

A simple box can be used to keep all the electronics of the circuit. This box can be built by using the laser cutting facilities at Imperial College. It is imperative that the box is as small as possible, and hides all the electronic components except for the electrodes, the power supply and the USB cable. The box can be made out of wood or plastic and the group's logo will be engraved on the cover, as well as labels for the electrode inputs, power supply and USB cable to facilitate ease of setup.

In order to further compact this design, the digital board will be placed above the breadboard using a shelf. There will be a small aperture in the front where the electrode socket array will be visible to plug the electrodes as well as a small aperture in the back for the power supply and USB cable. The boards will be secured to the bottom of the board and the shelf by using 4 screws and nuts at the corner of each board. The breadboard can be simply glued to the bottom of the box. Finally, a 34-pin ribbon cable will connect the boards and the breadboard, and a 9V battery will be used as the power supply.

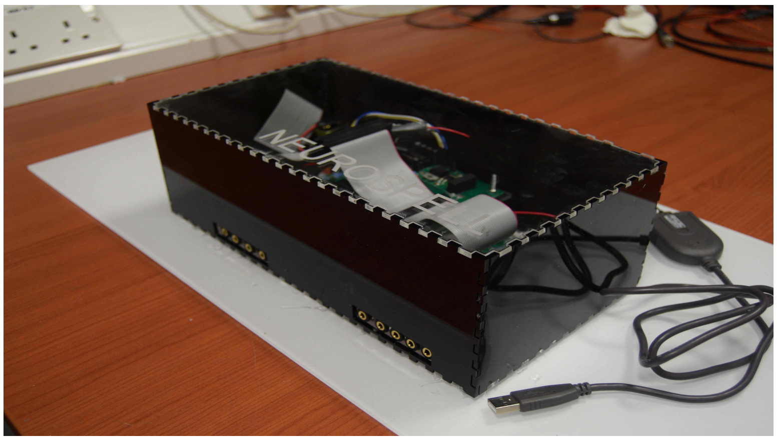


Figure 9: The group's box for the casing of all the electronic circuit

5.2. Software Development: Signal Processing

Once the signal has been acquired by the EEG hardware, it needs to be processed in order to distinguish when and what the user is trying to type. In order to do this, P300 evoked potentials will be used. P300 evoked potentials are positive peaks in the EEG recorded signal due to infrequent or unexpected auditory or visual stimuli, which appear 300ms after that stimulus has taken place. (Nicolas-Alonso & Gomez-Gil, 2012) In this particular case these stimuli will be the letters randomly flashing on the on-screen keyboard. Whenever the letter being targeted by the user is flashed a peak will appear on the EEG signal approximately 300ms after the flashing.

In order to be able to detect these peaks it is necessary to distinguish them from random noise. For that reason, a training session would be required before using this equipment on its intended regular use. The training consists of the following four steps before the online use is ready. These steps will be called *Scenarios*:

I. Scenario 1: Recording of the training session
II. Scenario 2: Training a spatial filter (using the signal recorded on Scenario 1)
III. Scenario 3: Training a classifier (using the signal recorded on Scenario 1)
IV. Scenario 4: Regular online use

This whole process can be done using the *OpenVibe Designer* software (OpenVibe, 2015) which includes examples on how to perform the previous steps and has a user-friendly box-based user interface (similar to Simulink and Labview) as well as allowing the use of Python scripts (more information on section 5.3). The EEG signal in OpenVibe is acquired through an Acquisition Client that allows the use of OpenEEG hardware as well as Emotiv Epoc. All these scenarios can be found in Appendix D.

5.2.1. Recording of the Training Session Scenario

This scenario will do the following:

- Play the On-screen keyboard interface

- Display a set of Target Letters that must be looked at by the user
- Record when the Randomly Flashed Letters flash.
- Record the raw EEG signal

A full picture with explanations of the program can be found in Appendix D.

5.2.2. Training a Spatial Filter Scenario

This scenario will take the signal previously recorded and the sets of Flashed and Target Letters. This scenario can be run without any user intervention. The raw EEG signal will be filtered through a 4th order pass-band Butterworth filter with cut-off frequencies of 1 and 20 Hz, which is the band of the most relevant brain waves (delta (1-4 Hz), theta (4-7 Hz), alpha (8-12), beta (12-20). (Nicolas-Alonso & Gomez-Gil, 2012)

After this, the filtered signal will be epoched (divided into time windows). The epoching will trigger whenever a target letter is flashed. The epoch duration will last 0.6s since the P300 spike occurs approximately 300ms after the flashing, therefore a wider window of time is taken in case it triggers before or after.

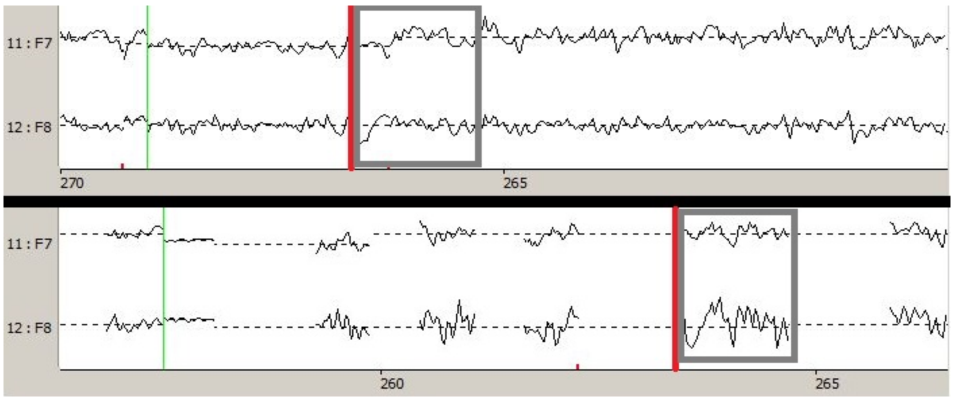
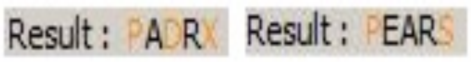


Figure 10: Two EEG signals on channels F7 and F8. Above the EEG signals and below the epoched signal. At the red line a Targeted Letter was flashed and thus, a time window of 0.6s of the EEG signals above is passed.

The purpose of this is to compare which channels react more to the flashing, so they can be used as signal references, which react less, so it can be used as noise references. This is done by generating a number of output channels from the input channel, each output channel being a linear combination of the input channels, so that the channels with a greater Signal to Noise Ratio are boosted while the ones with a lower SNR are weakened. The number of outputs can vary, however, after testing a range of 1 to 12 output channels from 12 inputs, it was found that the best results were achieved when using 3 to 5 outputs (refer to section 6.2: Testing the software for more information).



Result : PADRX Result : PEARS

Figure 11: Results of a user trying to spell PEARS without the filter (left) and with the filter (right)

5.2.3. Training Classifier Scenario

This scenario will take the signal recorded and filtered through the Butterworth filter described above and through a *Spatial Filter* using the configuration trained in the previous scenario. This configuration is basically the number of output and input channels and the coefficients of the linear combination of channels described above.

This signal then follows the next process:

- If there is a **match between the Target Letter and the Flashed Letter** an epoch of 0.6s is triggered, its average value is calculated and then it is stored in a vector (left branch of *figure 12)*.
- If there is **no match between the Target Letter and the Flashed Letter** the same process is followed but it is stored in a different vector.

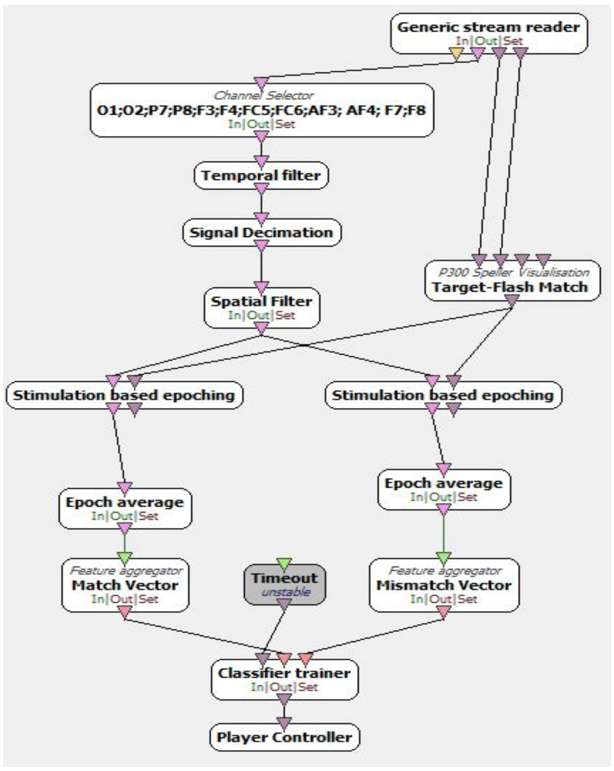


Figure 12: Training Classifier Scenario

After reading the whole file, there will be 2 vectors, one with the average values of a match (i.e. the signal received when the user was looking at a letter being flashed) and the values of a mismatch (i.e. the signal received when the user was looking at a letter not being flashed). Then, these two vectors are taken and a Linear Discriminant Analysis (LDA) is performed in order to recognise a pattern between matched and mismatched letters. This is done in the Classifier trainer box.

The result of this is a file that will be used in the On-line Scenario to tell whether and input $\vec{x}$ (an epoch average of the signal) belongs to the *match* or *mismatch class*. Although the LDA is processed by OpenVibe, it is worth pointing out that this is done by doing the dot product between the vector $\vec{x}$ and a vector $\vec{w}$ which is defined as $\overrightarrow{W} = \frac{\overrightarrow{\mu_1}-\overrightarrow{\mu_2}}{\Sigma}$, with $\overrightarrow{\mu_1}\ and\ \overrightarrow{\mu_2}$ being the mean of each class and Σ being the covariance matrix. If this dot product is greater than a threshold, then $\vec{x}$ belongs to the first class. (Wikipedia, 2016)

5.2.4. Online Scenario

This scenario is the one to be used on a regular basis once the training is finished. Its process is similar to the previous one. The signal passes through the Butterworth and Spatial Filter. Then, an epoch is triggered each time a row or column of the keyboard flashes. After this, the average of the epoch is taken and then it is passed as a vector $\vec{x}$ to a *Classifier Processor* that will use the information from the previous scenario to compute the dot product described above (called Hyperplane Distance). Finally, a *Voting Classifier* box decides if the distance received is above or below the threshold. It waits a number of n flashes to decide which one of them was more likely to be a match (the one with the highest Hyperplane Distance). Once a match is decided it is sent to the Python UI box that will display the decided letter.

Since there are 6 rows and 6 columns, 12 branches are needed in order to do the epoch-average-classification process, since each row and column flashes independently. The number n can be changed. The larger it is, the most accurate the result will be, but the longer it will take to get the result. The time that the user has to wait before getting a result is $T = 2t_f \cdot n$, where t_f is the time between two flashes. The factor 2 is present because two coordinates (row and column) are needed in order to get a letter (rows and columns flashing will always alternate).

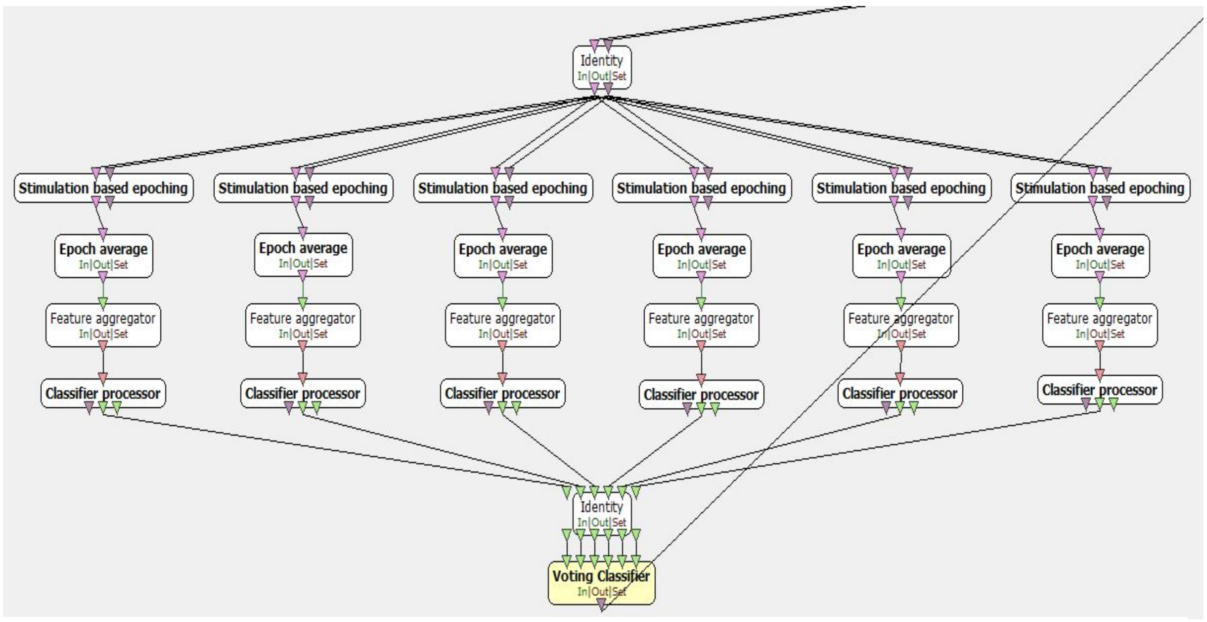


Figure 13: Row detection branch of the Online Scenario

5.3. Software Development: Interface

The P300 user interface is an on-screen keyboard that will allow users to choose what to type just by looking at the characters displayed. When running, row and column segments on the keyboard will flash randomly to stimulate signals from the user's brain that can be detected via the headset and then send to be processed. Since the P300 interface developed by OpenVibe provided limited options for customisation, the group decided to design its own user interface for the P300 speller. Moreover, additional functionalities were added, including text prediction and features to make the interface double as a text editor.

The programming language chosen to develop the interface was Python, as it is a robust, cross-platform programming language that is compatible with OpenVibe. Two versions of the user interface were developed.

The first version was developed using Pygame, a multimedia library for Python. As the hardware was still being developed, another version of the software that was compatible with the Emotiv Epoc was needed (as Emotiv was used for testing). From this, a second version using Pyglet was developed, another multimedia library for Python. All the code can be found in Appendix G.

5.3.1. Display design

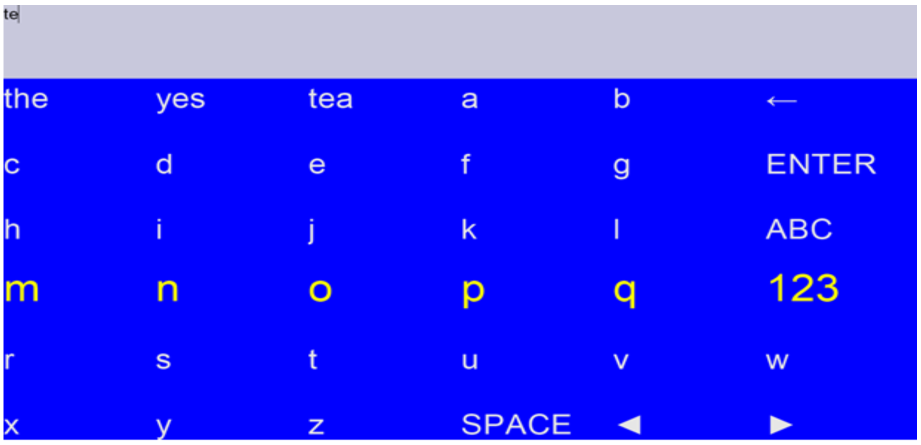
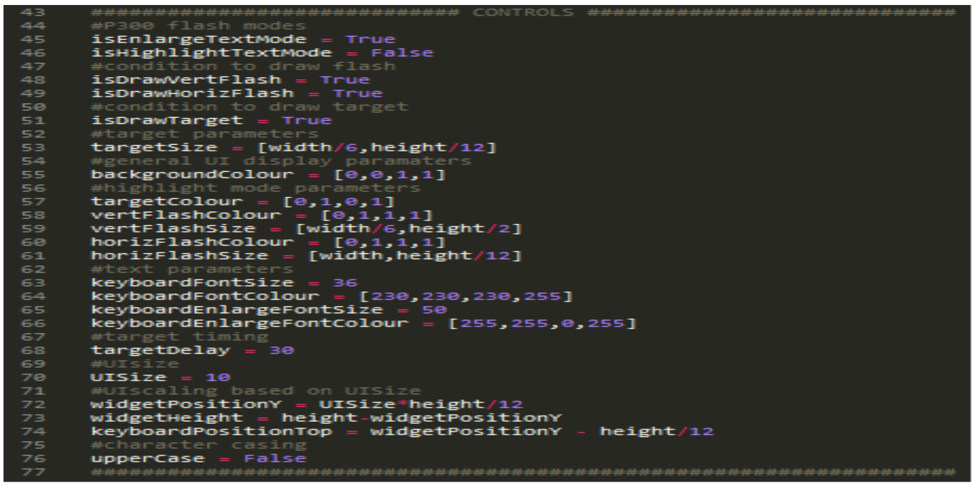


Figure 14: The second version of the user-interface using Pyglet. The first version was shown in section 3.2.1

The general interface design includes a user input display on top, and a keyboard display at the bottom. After some research (Neuroeng, 2012) and experimenting, the group realised that using highlights for flashes was not as effective as enlarging the characters in yellow. A blue background also gave a better response and thus was preferred over a black one. The group also decided to use P300 for the predictive text. This was because the original idea was to use the SSVEP signal to perform this operation of choosing the predicted word. However, due to time constraints of the project, SSVEP was discarded and therefore the predictive text choice was developed within the P300 matrix.

5.3.2. Customisability



```
############################ CONTROLS ############################
#P300 flash modes
isEnlargeTextMode = True
isHighlightTextMode = False
#condition to draw flash
isDrawVertFlash = True
isDrawHorizFlash = True
#condition to draw target
isDrawTarget = True
#target parameters
targetSize = [width/6,height/12]
#general UI display paramaters
backgroundColour = [0,0,1,1]
#highlight mode parameters
targetColour = [0,1,0,1]
vertFlashColour = [0,1,1,1]
vertFlashSize = [width/6,height/2]
horizFlashColour = [0,1,1,1]
horizFlashSize = [width,height/12]
#text parameters
keyboardFontSize = 36
keyboardFontColour = [230,230,230,255]
keyboardEnlargeFontSize = 50
keyboardEnlargeFontColour = [255,255,0,255]
#target timing
targetDelay = 30
#UIsize
UISize = 10
#UIscaling based on UISize
widgetPositionY = UISize*height/12
widgetHeight = height-widgetPositionY
keyboardPositionTop = widgetPositionY - height/12
#character casing
upperCase = False
###################################################################
```

Figure 15: Screenshot of Pyglet program where the parameters of the user-interface can be customised

In terms of customisability, there were a few useful parameters that needed to be varied: background colour, flashing colours, flashing frequency, text sizes, and highlight dimensions. The option of using highlights or not was added as well. With such a flexible way of changing parameters, it is easy to experiment with various different display styles to find out which scheme works best and gives the most accurate and reliable results. In the future, a UI will be created in a way that users can modify these settings without needing to refer to the code.

5.3.3. OpenVibe compatibility

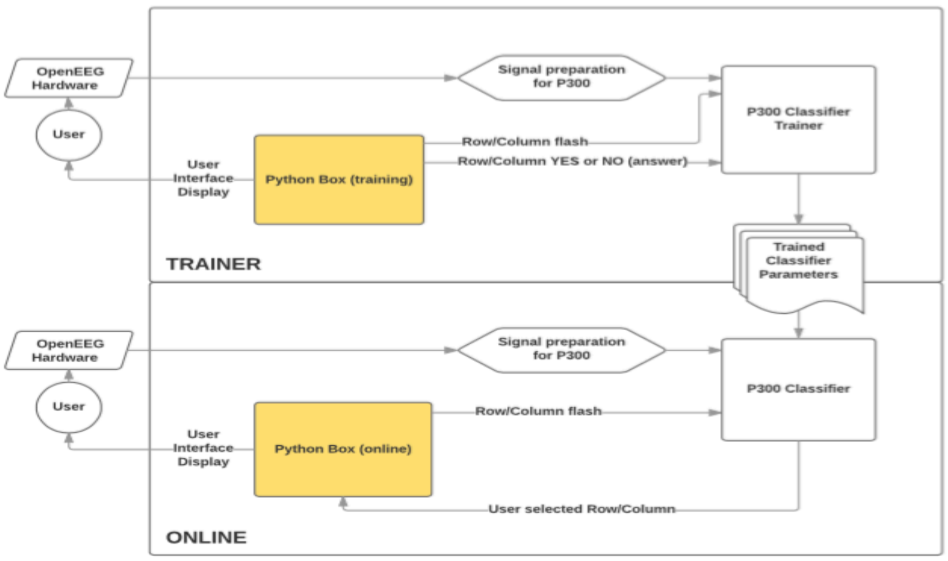


Figure 16: Program flowchart describes connection between Pyglet UI and OpenVibe

OpenVibe supports the addition of Python scripts into its program flow, such that the script is run within the scenario that the group set up. By instantiating the provided OVBox class in the script, the group was able to access attributes and methods of that class, which provide ways to handle input and output between the Python user interface and the OpenVibe stimulation streams. However, the current architecture of OpenVibe does not fully support all Python libraries, which has placed constraints on the ease of development within the environment.

A further complication that was faced while developing the user interface for OpenVibe was real time performance. The group's OpenVibe scenario depends on tight timing requirements to function correctly, which means that the Python script had to be fast enough to not slow down the real-time performance of OpenVibe. Despite these constraints, the group was able to integrate its own software into OpenVibe to obtain the desired functionality.

5.3.4. Predictive Text

To implement the predictive text, an algorithm found in Norvig (2013) was used to provide the user with three predictive text choices. The algorithm works by first taking in a large text file containing about a million English words. These words are used as data to train a probability model i.e. count the number of times the word occurs in the file. Words not provided in the text file are accounted by setting all unseen words to seen once. Then, all possible corrections of a given word are enumerated. With this, the set of words can be compared with the input word by the edit distance (or the number of edits required to transform the word into the other). In this algorithm, it is assumed that the lower the edit distance the better. Once the set of words with the shortest edit distance to the word is acquired, the word with the highest occurrence, based on the training done previously, is chosen. **This predictive text together with a BCI speller is completely innovative and it has not been seen in any speller found so far.**

6. Testing the prototype

6.1. Hardware testing

Tests: Once the hardware had been built the following tests were performed:

- At the signal output of the Amplifier Boards (Connection SJ201 and SJ204 in Appendix C): Test that the signal reacts to the electrodes being touched using an oscilloscope.
- Test that the microcontroller on the digital board ATMega8 is well programmed. For that, Atmel Studio and Avrdude software will be used.
- Test that signal is received from the electrodes on OpenVibe and that once the headset is on its place a spike can be seen at each channel waveform when the user clenches the jaw.

Results: The first two tests were completely successful although minor modifications to the code provided by the EEG project should be made in order to update the code for the ATmega8, since it was out-dated. (Please refer to Appendix E) The third testing point is currently under compatibility issues with Windows 8.1 and Windows 10, which stop the signal from being received after a few seconds of recording.

Solution: There are two options to solve the compatibility issue.

- Use a Computer with Windows XP or 7 to run OpenVibe.
- Create a new driver or use an alternative driver (there are generic drivers for development applications on OpenVibe). In order to use an alternative driver, a new protocol of communication would be most likely required. This basically means that the code on the ATmega8 microprocessor on the digital board would need to be rewritten.

The first solution was tested on a Windows 7 laptop with the same results, and it was also tested using the compatibility mode of Windows XP with no improvement. That leaves only the second proposed solution, which has not been implemented yet.

6.2. Software testing

Tests: The software testing consists of the following tests:

- Test that a signal is being received and recorded from Emotiv Epoc headset at the Communications Department Laboratory at Imperial College London. The Emotiv Epoc headset provides software that can be used to detect whether the signal is good enough for recording or not.
- Record real signals from several users in order to determine:
  - How many outputs the Spatial Filter should have

- Which flashing frequency is most appropriate
- How many letters flashings are necessary to have a consistently correct output (number n on section 5.2.4)
- Which colours are both effective and less tiring for the user

Results:

- The signal is properly received and recorded.
- Results from real signals recorded (these results have not been obtained using the speller or word predictor):
  - After a preliminary trial it was found that the optimal number of outputs for the spatial filter is between 3 and 4, which confirms the recommendations in the OpenVibe documentation (OpenVibe, 2015).

Number of Spatial filter outputs	Number of correctly spelled letters out of 5 (%success)		Number of Spatial filter outputs	Number of correctly spelled letters out of 5 (%success)
1	1 (20%)		8	1 (20%)
2	2 (40%)		9	1 (20%)
3	5 (100%)		10	2 (40%)
4	5 (100%)		11	2 (40%)
5	5 (80%)		12	2 (40%)
6	4 (80%)		No spatial Filter	2 (40%)
7	3 (60%)			

Table 3: Comparison of the number of spatial filters used during different tests, and the success percentage with each number (results obtained with n=48 and flashing frequency = 3.5Hz)

- 8 training sessions and 8 corresponding attempts to spell the word "brain" were recorded. Each training and attempt was recorded at a different flashing frequency. The tests were randomly presented, i.e. the user did not know what was going to be the next frequency. The following results were obtained:

Flashing frequency (Hz)	Percentage of Success
1.5	0%
2	20%
2.5	20%
3	20%
3.5	60%
4	60%
4.5	0%
5	0%

Table 4: Comparison of the flashing frequency used during 8 different tests, and the success percentage (results obtained with n=48 and number of spatial filter outputs = 3)

This shows how at frequencies around 3-3.5 Hz the results seem to be significantly better. Nevertheless, more testing is required in order to confirm that this frequency is the optimal one since all these tests were done by the same user. Longer randomly picked words should also be tested. The spatial filter number of outputs was varied from 1 to 12 for the most successful recorded sessions and it was confirmed that a filter with 3 or 4 outputs tends to work better.

Number of spatial filter outputs	1	2	3	4	5	6	7	8	9	10	11	12
Success at Session at 3.5 Hz (%)	20	40	60	60	60	40	40	40	60	40	20	20
Success at Session at 4 Hz (%)	40	60	60	40	40	40	60	40	40	20	20	20

Table 5: Comparison of the number of spatial filters used during different tests at frequencies 3.5 and 4 Hz, and the success percentage (results obtained with n=48)

- By using 5 different users' feedback who took training and spelling sessions of 40 min on a black and white and a blue and yellow interface, it has been determined that the second one is less tiring to look at for long periods of time as suggested in *"P300 brain computer interface: current challenges and emerging trends"* Neuroeng (2012).
- It is still needed to test the optimal number of flashes to average over on the *Voting Classifier Box.* An arbitrarily large number n=48 was chosen.

There is still room for improvement. Results would be significantly better if the speller and word predictor were used. Besides, as discussed earlier, Emotiv Epoc electrodes are not placed in the optimal areas for P300 detection and it uses conventional EEG, not differential. Thus, it is expected that using a headset with the characteristics as the one the group has developed, the results should notably improve.

7. Project Management

Initial meetings were held at the beginning of the Autumn term to discuss the work required to complete this project. Subsequently, the project was compartmentalised into three sub-projects: hardware, signal processing and user interface. The latter two were done entirely utilising software. To this end, the group was subdivided into three sub groups, each responsible for one of the aforementioned categories. Delegation of group members to specific tasks was done based on their knowledge and capabilities in the particular field. Personal preference was also taken into account. The table below illustrates the division of work amongst the group members:

Category	**Subgroup**
Hardware	Samuel, Jorn, Javier
User Interface	Aaron, Jun
Signal Processing	Nicolas, Vinay

Table 6: Distribution of workload

Following this was the scheduling of meetings for the Autumn and Spring terms. Small deliverables were established at the end of every meeting, to be presented at the beginning of the next meeting. This was done to streamline the project's advancement and frequently monitor the group's progress relative to the long-term schedule. This ensured that if any division was falling behind the prescribed timeline, it would be highlighted before the situation became severe. As seen in the Gantt chart below, deliverables were met at the prescribed times for the most part, with no task other than the delivery of components taking more than 1 week longer than expected.

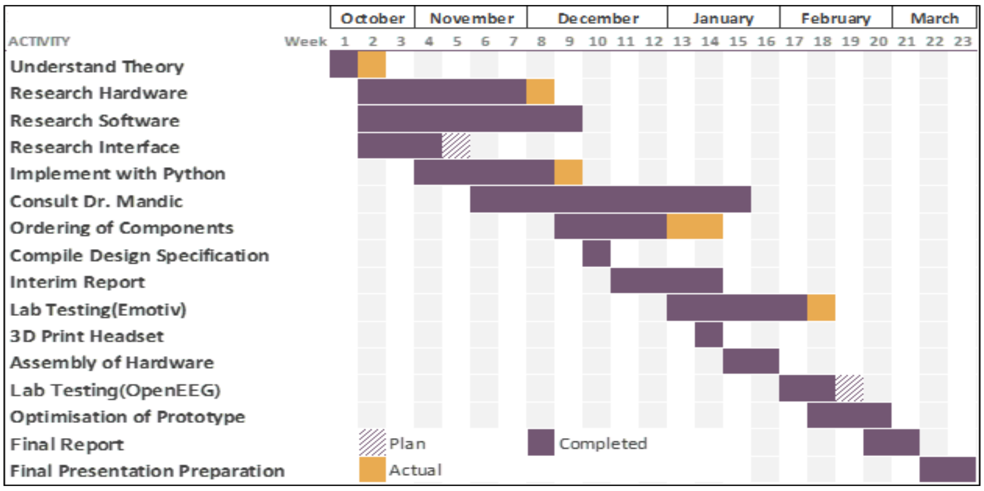


Figure 17: Gantt Chart of the group's expected and actual completion of deliverables

In addition, precautionary measures were formulated in the event that this project proved to be unfeasible. With respect to hardware, if OpenEEG could not be assembled and tested within the allotted time frame, permission was granted by the EEE department's BCI research group to use their Emotiv Epoc headset. Similarly, if an entire keyboard was demonstrably implausible, the magnitude of the experiment would be reduced to less than 10 possible characters, facilitating simple operations such as controlling a wheelchair and dialling a telephone number.

Finally, rules to handle inappropriate conduct and other red flag issues were established at the end of the first meeting. It was decided that problems of this nature would first be reported to the group leader, and in the event that it could not be resolved in this way, then the project supervisor would be notified. In retrospect, there have not been any conflicts in need of action from the group leader or the project supervisor.

7.1. Costs

Component	Cost Per Unit(£)	Quantity	Total(£)
EEG-Analogue-Asm	58.25	2	116.50
EEG-Digital-Asm	38.83	1	38.83
EEG-Active-Electrode	6.99	8	55,92
Perspex Sheet	3.00	2	6.00
9V Battery	3.50	1	3.50
34 Pin Ribbon Cable	5.00	1	5.00
3.5mm Female Connector	0.50	10	5.00
Serial Port Adapter	10.00	1	10.00
Total Cost	-	-	**240.75**

Table 7: Total device cost in case of final production

8. Conclusion and Future Work

The work developed during the last five months has shown that it is possible to build a low-cost EEG headset, as well as developing a software solution to implement a speller with it. Despite this, the project is still in its early development and a large amount of work would need to be done until a final product could be released.

It has been proved that the speller can be implemented using Emotiv Epoc, but it is still necessary to test if it works with OpenEEG. In order to do that, the compatibility problems must be solved. This may require modifying the current drivers to interface OpenEEG with Windows 8.1 and 10. After this, it is necessary to perform a full analysis on reliability and its dependence in the following parameters: flashing frequency, interface colours, spatial filter configuration and number of flashes for an optimal detection. This analysis should be done by testing the prototype with randomly selected users, both motor impaired and non-motor impaired. This step would give a good insight into whether this project would succeed or not.

The next and final step would be optimisation. It will entail updating the electronics as well as trying to include them in the headset itself if possible (some headsets already do this but for a higher price: OpenBCI, Emotiv Epoc, etc). In addition, the same reliability analysis stated before should take place after the final design is made. This analysis and testing should also include how long the device is operational without having to retrain the On-line Scenario. Besides it should also be tested on several randomly selected people. Despite the fact that the word predictor has been implemented and has worked correctly, it is still based on the P300 phenomenon. Thus, the hybrid P300-SSVEP system discussed in Section 4 (P300 for letters, SSVEP for predicted words) would also be implemented and tested at this point. The group would expect to have a very reliable and fast BCI speller after this point.

In conclusion, the full process for developing a reliable BCI system would take between 2 to 4 years. Nevertheless, it has been proved that a huge investment would not be necessary in order to do it, and thus the final product cost might be well in the anticipated margin (around £200). In addition to having a price that would be less than other similar products, innovation is present in different aspects of the device. For instance, in terms of customisability, the user would be able to choose the colour of the headset and the box, as well as adjusting the size of the headset to fit tightly on the head. Furthermore, the user-interface can be easily modified to change letter size and colour, as well as having a predictive text that could largely improve the typing speed.

To sum up, it cannot be confirmed that the aim of this project –developing a low-cost BCI speller- is a hundred percent viable at this stage. However, the work done so far provides a solid ground towards a successful final product in a medium-range term, and all the evidence so far has shown that this would be feasible.

9. References

Emotiv (2015) *Emotiv EPOC/EPOC+.* [Online] Available from: https://emotiv.com/epoc.php. [Accessed: 20th November 2015]

Galindo Merchan, D. (2008) *Brain Computer Interface*. University of Valladolid Telecommunication Engineering Department. (July). P.1-34.

Guger, Christopher, Vaughan, Theresa & Brendan Allison (2014) *Brain-Computer Interface Research.* London: Springer.

Guger Technologies (2012) *Publications.* [Online] Available from: http://www.gtec.at/Research/Publications. [Accessed: 26th December 2015]

Lee, J., Shin, S.H. & Istook, C.L. (2001) Analysis of Human Head Shapes in the United States. *2006 IJHE Journal. (2006).*

NeuRA. (2016) *Motor Impairment Blog.* [Online] Available from: http://motorimpairment.neura.edu.au/about/ [Accessed: 4 March 2016]

Neuroeng (2012) P300 brain computer interface: current challenges and emerging trends. *Frontiers in Neuroengineering.* PMC3398470 [PMC free article] [PubMed]

Nicolas-Alonso, L.F. and Gomez-Gil, J. (2012). Brain Computer Interfaces, a Review. *Sensors*. PMC3304110. [PMC free article] [PubMed]

Norvig, Peter (2013) *How to Write a Spelling Corrector.* [Online]. Available from: http://norvig.com/spell-correct.html. [Accessed: 1st December 2015]

Olimex (2015) *EEG-ANALOG-ASM.* [Online] Available from: https://www.olimex.com/Products/EEG/OpenEEG/EEG-ANALOG-ASM/open-source-hardware. [Accessed: 10th November 2015]

OpenBCI (2016) *OpenBCI.* [Online] Available from: http://www.openbci.com/. [Accessed: 1st November 2015]

OpenEEG (2015) *Hardware: Modular EEG.* [Online] Available from: http://openeeg.sourceforge.net/doc/index.html. [Accessed: 19th October 2015]

OpenVibe (2015) *Software for Brain Computer Interfaces and Real Time Neurosciences.* [Online] Available from: http://openvibe.inria.fr/. [Accessed: 1st November 2015]

Peyton R. (2014) *Selecting an EEG Device.* [Online] Available from: http://scienceforthemasses.org/2014/04/11/selecting-an-eeg-device/. [Accessed: 26th December 2015]

R. S. Leow, F. Ibrahim, Member, IEEE & M. Moghavvemi, 2007*.* In: University of Malaya (Department of Biomedical Engineering, *International Conference on Intelligent and Advanced Systems 2007*. Kuala Lumpur, Malaysia. 25 November-28 November 2007.

Sheng-Feng Yen et al. (2009) *Differential EEG. Proceedings of the 4th International IEEE EMBS Conference on Neural Engineering*. Antalya, Turkey

Thingiverse (2016) *MakerBot Thingiverse.* [Online] Available from: https://www.thingiverse.com/ [Accessed 20th December 2015]

Wikipedia (2016) *Linear discriminant analysis.* [Online] Available from: https://en.wikipedia.org/wiki/Linear_discriminant_analysis . [Accessed: 10th February 2016]

DIYtDCS (2015)*EEG Standard* [Online] Available from: www.diytdcs.com/media/010_EEG_standard.gif [Accessed: 10th January 2016]

Appendix A: Product Design Specification

<u>Product Design Specification</u>

Project : BCI **Date: 13 March 2016**

Author: Group 8 **Version: 3**

1. <u>Performance</u>

   This device is intended to be used to spell letters on a screen by just looking at an on-screen keyboard. EEG data will be recorded from the subject via electrodes. Then a screen will flash letters on the keyboard. If the targeted letter is flashed then it will be displayed on the screen. This should produce a letter each 15 seconds. The keyboard will include a word predictor system so that it corrects spelling mistakes and makes typing faster.

2. <u>Environment</u>

   The product would need to operate indoors since it is mainly composed of electronic components that could be damaged if exposed to weather conditions.

3. <u>Life in Service</u>

   Life of the entire device should be around 3-5 years. This is due to the fact that we expect to get more reliable, economic and updated versions of this product in that time, therefore during that period, the product needs to remain reliable and in good use.

4. <u>Maintenance</u>

   Occasional replacement of electrodes, around 6 months. The box where the boards are placed is designed to that it can be easy to access the different parts in case replacing some components is needed. However, most times this should be done by qualified personnel.

5. <u>Target Product Cost</u>

   £ 250. This amount was agreed between the group and the staff in charge of funding the project. The total cost is a sum of different components: PCB Amplifier boards, PCB Digital board, electrodes, cables, connections and casing. This does not include the cost of a required PC with Windows or Linux installed in order to use this device.

6. <u>Competition</u>

   This project is not meant as a product to be launched into the market, but more focused into expanding the research on the BCI area and explore more cost effective solutions. Nevertheless, a company named Emotiv is currently selling full-mounted headsets, with their own user interface for a price of around £500, although their products are not directly intended to be used for this kind of application. There is another company called OpenBCI which has a wide range of products related to Brain Computer Interface applications. Nevertheless, the price of a full headset is above £450.

7. Shipping

There are two parts of the product: software and hardware. Software can be downloaded from our website. Hardware will be sent via any regular delivery service.

8. Packing

A box with fitted Styrofoam. Preferably inside a box that is tightly sealed to prevent dust corrupting the electronic components.

9. Quantity

The product will be created on demand as we do not expect to have a production process longer than 2 weeks. The group would need to contact facilities aimed at treating motor impaired patients to determine the quantity required and needed.

10. Manufacturing Facility

An environment with soldering tools, laser cutters and 3-D printers. A place similar to the 5th floor electronics lab in the Electrical and Electronic Engineering Department at Imperial College.

11. Customer

The main customer for which this product is intended would be a motor-impaired person with high difficulty or impossibility to speak or type. However, the product would be sold to specialized facilities in which patients that suffer of this disease are being treated. The total product cost would make the product's purchase affordable for these facilities.

12. Size

Two parameters to consider: headset and the box containing the electronic components.

Headset: Different headsets would be built to adjust to different head geometries. However, as a prototype, the group decided to build a headset of approximately 24cmx14.5cm (Lee, J., Shin, S.H. & Istook, C.L., 2001)

Box: The boards will be placed in a box that should be able to be placed in a wheel chair. Due to the size of the current PCB boards and the large amount of connections, the box has been calculated to have dimensions of 40cmx30cmx20cm. However, for future versions of this product we expect it to be at least one third of this size if not smaller.

13. Weight

Headset: less than 0.25 kg as more weight could be uncomfortable for the user

Box: less than 1 kg as more weight placed on the lap could affect the user if the product is used for long periods of time.

14. Materials

Our product consists of 3D printed polylactic acid (PLA) plastic for the headset prototype and laser-cut acrylic for the box where the electronic components will be placed. The rest is made out of standard electronic components and saline gel for proper conductivity of the electrodes.

15. Product Life Span

Costumer-feedback dependent although the technology used for the product is expected to be relevant for the next 3-5 years.

16. Aesthetics, Appearance and Finish

The aesthetics, appearance and finish of the products are divided into three sub-parts of the whole product: headset, box containing the electronic components and user-interface.

Headset: It will be built out of lightweight plastic printing fiber, therefore it will be comfortable to wear for long periods of time. Moreover, there could be a wide range of color offer for the user to choose from.

Box: The box will be laser-cut wood or plastic that will make all the components to fit tightly inside. It can also be added the user's name into the box to add personalization as well as color preference.

Interface: The code used to program the entire interface is designed so that it is easy to change the color of the interface as well as the size of the letters and keyboard. This would allow further user personalization.

17. Ergonomics

The ergonomic design is one of the most important aspects of our product. The headset should fit the user's head comfortably, thus different headset sizes should be produced once the product is finalized. The interface has to be completely intuitive and minimal training would be required for day-to-day use (training would take 20-30 minutes, and ideally will not be necessary more than once a week).

18. Standards and Specifications

The product should be designed to fit current international standards regarding safety and guarantee. This is the case since the user is a very specific sector of the population, thus, the product has to be ready to be used anywhere in the world.

19. Quality and Reliability

Crucial for our product. The whole aim revolves around the idea that a person that is not able to communicate will be able to do so with our product. For this reason, a reliability of at least 90% is expected so that communication can be meaningful. In addition, regular feedback has to be sent by the user in order to improve any imperfections. This would help to troubleshoot certain aspects in the user-interface that can be modified in order to improve reliability.

20. Shelf Life (storage)

Approximately 10 years as electronic components can be used for many years without malfunctioning and software will not deteriorate. Only electrodes could be somewhat corrupted at an earlier time.

21. Testing

After manufacture, each product should be tested before delivery to check that it works. The testing would require certain standard procedures:

-Check that all the electrical connections between the electronic components are working
-Check that the box and headset do not have any faults that could cause an early malfunction
-Test the whole product to ensure that it effectively is able to produce letters and words in the screen with the software being used

22. Processes

N/A.

23. Time Scale

From design planning to the building of a prototype: 5 months as this is the time given before the project has to be presented.

From a prototype to final product: It would take at least a year to build a final product that would be sufficiently reliable that it could be sent to the market. Therefore, it could take up to 2 years.

24. Safety

Maintain an electrical isolation barrier between a user connected to the product and the device to which the EEG device is connected.

The product should not be used during a lightning storm or whenever the electrical power grid in unstable.

Could cause unpleasant side-effects for a small number of people. Usage of our product could lead to anxiety and/or leading to tics, insomnia or panic attacks. Stimulation of latent seizure activity to full (epileptic) seizure activity. Mood changes, such as depression or anger outbursts.

25. Company Constraints

Limited budget and man-power as well as supplier reliance.

26. Market Constraints

The main market constraint is that the costumer is a very specific type of user, thus expansion of the product could not be large scale.

27. Patents, Literature and Product Data

The whole project has been developed based on an open source project called Open EEG where information on how to build a BCI system was published in their website. In addition, an open source software called OpenVibe was used to do all the signal processing.

28. Legal

N/A.

29. Political and Social Implications

Since our target costumer is very specific we do not plan do make a major social or political impact. It could however have a large impact on the facilities where these patients are being treated since communication with them will be greatly improved.

30. Installation

Our product requires a fully working Windows or Linux computer with an installed OpenVibe program. The computer should have Python as well as a monitor to display the interface with the user.

31. Documentation

Our documentation will be open source. Will provide a user manual. Any non-trained user should be able to use the product with ease if all the installation requirements have been met.

32. Disposal

Disposal as any electronic component, such as a WEEE directive. They could as well be sent to us so that we could make use of the components that still work and therefore recycle the products.

Appendix B: Minutes of Meetings

- **Meeting 1**: 18 January 2016

  Present: All members
  Time: 2 hours

-Issues from previous meeting:
- Check the report overall before submission
- Testing of the software
- Start looking at how to connect the PCB boards

-Action points and decisions:
- Worked on trying to implement the software on OpenVibe but had compatibility problems with Windows
-Started to check what cables would be needed to connect all the hardware parts

-Tasks for next meeting:
- Advance with software development
- Try to book Level 8 lab to start doing tests with the Emotiv Epoc

-Scheduled next meeting:
- TBC

- **Meeting 2**: 26 January 2016

  Present: All members
  Time: 2 hours

-Issues from previous meeting:
- Advance with software development
- Try to book Level 8 lab to start doing tests with the Emotiv Epoc

-Action points and decisions:
- Group in charge of developing software were programming
- Cables and connections were determined and were order via de EEE Online Stores
- Introduced on how to use the Emotiv Epoc so that we would be able to start using it for testing
- Some initial tests with Emotiv headset to see how it worked

-Tasks for next meeting:
- Figure out how to use Emotiv properly with OpenVibe
- Continue developing software
- Start testing PCB boards once connections arrive

-Scheduled next meeting:
- 1 February 2016

- **Meeting 3**: 1 February 2016

  Present: All members
  Time: 2:30 hours

-Issues from previous meeting:
  - Figure out how to use Emotiv properly with OpenVibe
  - Continue developing software
  - Start testing PCB boards once connections arrive

-Action points and decisions:
  - Work on the development of the OpenEEG hardware
  - Started to connect PCB board to a small breadboard so that the necessary resistors could be placed on the breadboard and therefore the connections made better

-Tasks for next meeting:
  - Continue with software development
  - Book Level 8 Labs to get more testing with Emotiv done

-Scheduled next meeting:
  - 4 February 2016

- **Meeting 4**: 4 February 2016

  Present: All members
  Time: 2:30 hours

-Issues from previous meeting:
  - Continue with software development
  - Book Level 8 Labs to get more testing with Emotiv done

-Action points and decisions:
  - Continued testing with Emotiv Epoc
  - Could get the raw date and saw that brain signals worked but no output file with the data was created
  - Software was developed during the session as the group realized that the incompatibility problems with Windows required a new software development to make it compatible with Windows

-Tasks for next meeting:
  - Progress on new software for Windows
  - Build hardware and 3-D print headset

-Scheduled next meeting:
  - 9 February 2016

- **Meeting 5**: 9 February 2016

  Present: All members
  Time: 1:30 hours

-Issues from previous meeting:
  - Progress on new software for Windows
  - Build hardware and 3-D print headset

-Action points and decisions:
  - Connections for hardware arrived
  - Started to connect hardware together, which included soldering and pasting the electrode sockets to the PCBs
  - Had to check whether the connections were properly made

-Tasks for next meeting:
  - Advance on software
  - Book Level 8 labs to work on further testing with Emotiv

-Scheduled next meeting:

- 15 February 2016

- **Meeting 6**: 15 February 2016

Present: All members
Time: 3 hours

-<u>Issues from previous meeting:</u>
- Advance on software
- Book Level 8 labs to work on further testing with Emotiv

-<u>Action points and decisions:</u>
- Further testing with Emotiv
- Changed several signal processing filters in OpenVibe to make sure that the signal was processed correctly
- First tests made outputs of 1 letter correct out of 5 or even no correct letter
- Added a Spatial Filter to the signal processor which apparently would make a big difference

.- After adding the Spatial Filter, one of the tests done worked. The user's intention was to write Pears and the program effectively processed the signals and spelled Pears

-<u>Tasks for next meeting:</u>
- Continue building hardware and connect everything
- Laser-cut a box so that all the electronic components can be placed inside
- Finish 3-D printing headset

-<u>Scheduled next meeting:</u>
- 24 February 2016

- **Meeting 7**: 24 February 2016

Present: All members
Time: 3 hours

-<u>Issues from previous meeting:</u>
- Continue building hardware and connect everything
- Laser-cut a box so that all the electronic components can be placed inside
- Finish 3-D printing headset

-<u>Action points and decisions:</u>
- Successfully connected the entire electrical circuit
- The group was able to power up the device
- Having problems with connecting to computer
- Box was laser-cut, and all the hardware was placed inside
- 3rd rim out of 4 was 3-D printed and mounted on existing headset

-<u>Tasks for next meeting:</u>
- Book level 8 lab to work on further testing with Emotiv
- Have software basically finished to start testing with it

-<u>Scheduled next meeting:</u>
- 26 February 2016

- **Meeting 8**: 26 February 2016

  Present: All members
  Time: 3 hours

  -Issues from previous meeting:
  - Book level 8 lab to work on further testing with Emotiv
  - Have software basically finished to start testing with it

  -Action points and decisions:
  - Worked on software to solve minor issues
  - Further testing was done in labs
  - This time, tests were not successful. No word chosen from the user to spell was written correctly. At most, 2 out of 4 letters were correct.
  - Determined that it was due to the pre-built user interface in OpenVibe, thus will have to start testing with own software

  -Tasks for next meeting:
  - Finish software and test with it to see if it works with Emotiv

  -Scheduled next meeting:
  - 2 March 2016

- **Meeting 9**: 2 March 2016

  Present: All members
  Time: 3 hours

  -Issues from previous meeting:
  - Book Level 8 Lab to work on testing with our own software

  -Action points and decisions:
  - Tried OpenVibe with our own software
  - Did testing with different flashing frequencies to determine what frequency would give the optimum results
  - Checked how our software compared to the one already present in Open Vibe

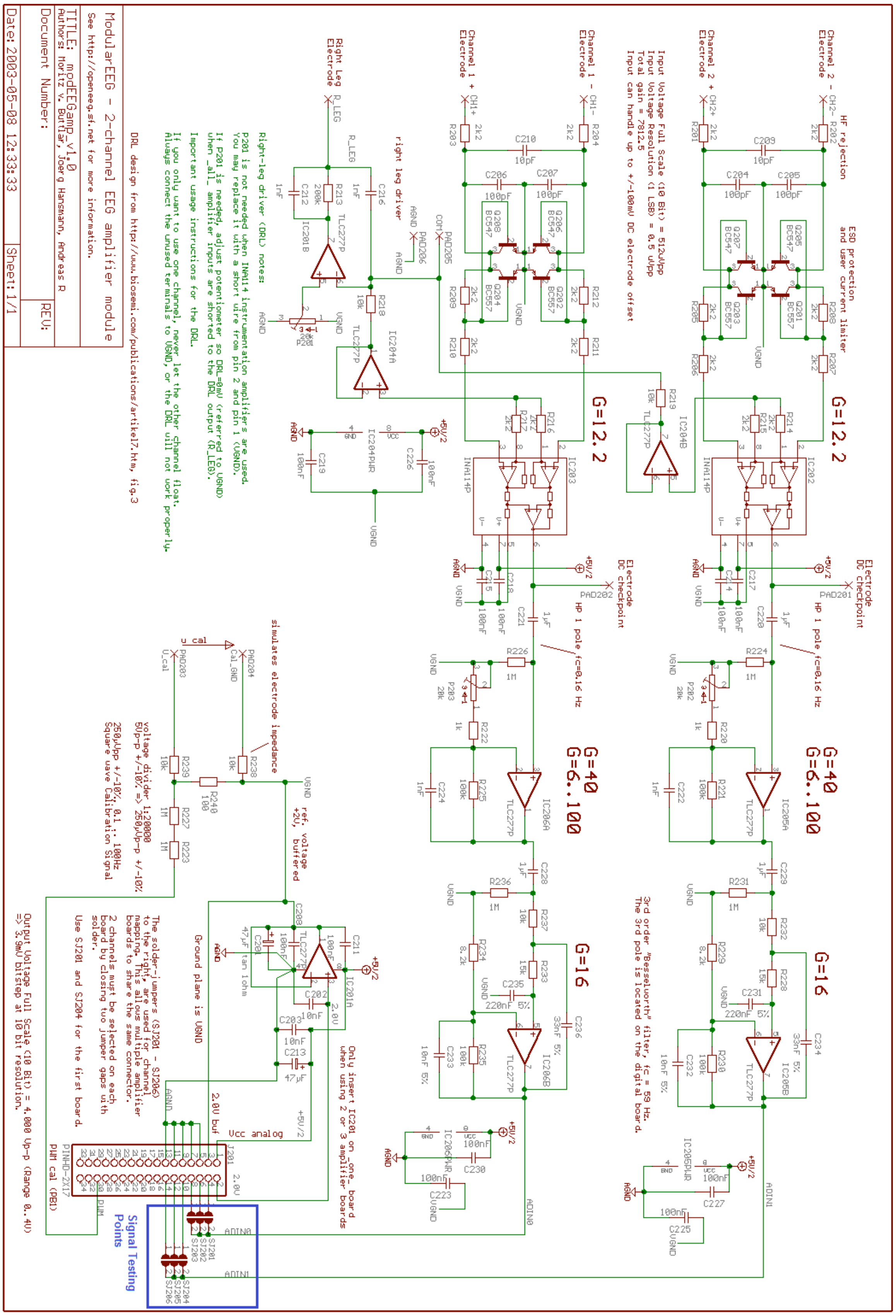
  -Tasks for next meeting:
  - Finish testing and building minor details on hardware

  -Scheduled next meeting:
  - 4 March 2016

Appendix C: OpenEEG Schematics

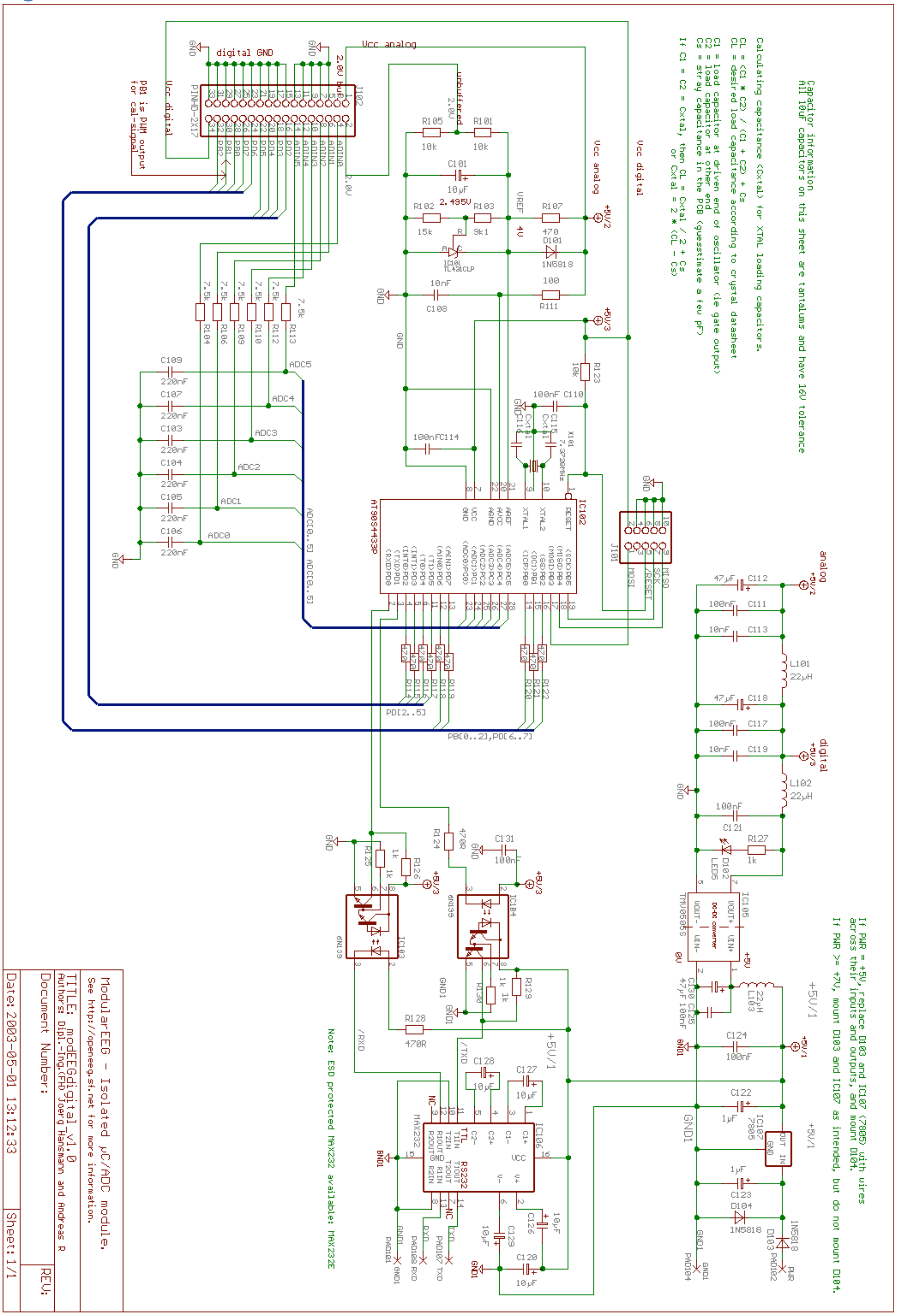
Obtained from OpenEEG webpage (OpenEEG 2015)

Amplifier Schematic:



ModularEEG - 2-channel EEG amplifier module
See http://openeeg.sf.net for more information.
TITLE: modEEGamp_v1.0
Authors: Moritz v. Buttlar, Joerg Hansmann, Andreas R
Document Number:
REV:
Date: 2003-05-08 12:33:33
Sheet: 1/1
DRL design from http://www.biosemi.com/publications/artikel7.htm, fig.3
Right-leg driver (DRL) notes:
P201 is not needed when INA114 instrumentation amplifiers are used.
You may replace it with a short wire from pin 2 and pin 1 (UGND).
If P201 is needed, adjust potentiometer so DRL=0mV (referred to UGND)
when _all_ amplifier inputs are shorted to the DRL output (R_LEG).
Important usage instructions for the DRL:
If you only want to use one channel, never let the other channel float.
Always connect the unused terminals to UGND, or the DRL will not work properly.
Right Leg Electrode
Channel 1 + Electrode
Channel 1 - Electrode
Channel 2 + Electrode
Channel 2 - Electrode
Input Voltage Full Scale (10 Bit) = 512uVpp
Input Voltage Resolution (1 LSB) = 0.5 uVpp
Total gain = 7812.5
Input can handle up to +/-100mV DC electrode offset
HF rejection
ESD protection and user current limiter
right leg driver
G=12.2
G=12.2
Electrode DC checkpoint
Electrode DC checkpoint
HP 1 pole fc=0.16 Hz
HP 1 pole fc=0.16 Hz
G=40
G=6..100
G=40
G=6..100
G=16
G=16
3rd order "Besselworth" filter, fc = 59 Hz.
The 3rd pole is located on the digital board.
simulates electrode impedance
voltage divider 1:20000
50Vp-p +/-10% => 250uVp-p +/-10%
250uVpp +/-10%, 0.1...100Hz
Square wave Calibration Signal
ref. voltage +2V, buffered
Ground plane is UGND
The solder-jumpers (SJ201 - SJ206) to the right, are used for channel mapping. This allows multiple amplifier boards to share the same connector.
2 channels must be selected on each board by closing two jumper gaps with solder.
Use SJ201 and SJ204 for the first board.
Only insert IC201 on one board when using 2 or 3 amplifier boards
Output Voltage Full Scale (10 Bit) = 4.000 Vp-p (Range 0..4V)
=> 3.9mV bitstep at 10 bit resolution.
Vcc analog
2.0V buf
PWM cal (PB1)
ADIN0
ADIN1
Signal Testing Points

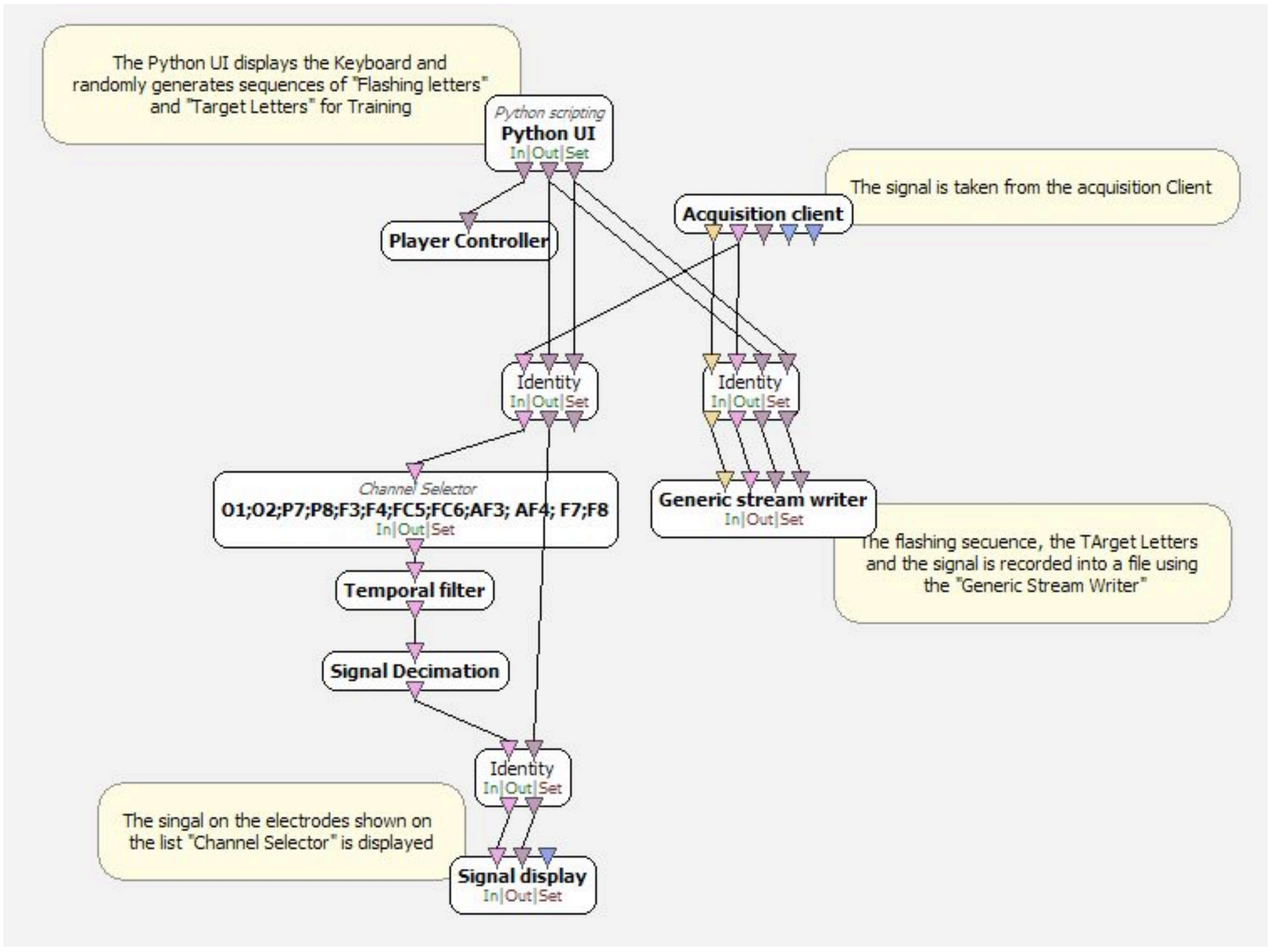
Digital Board Schematic:



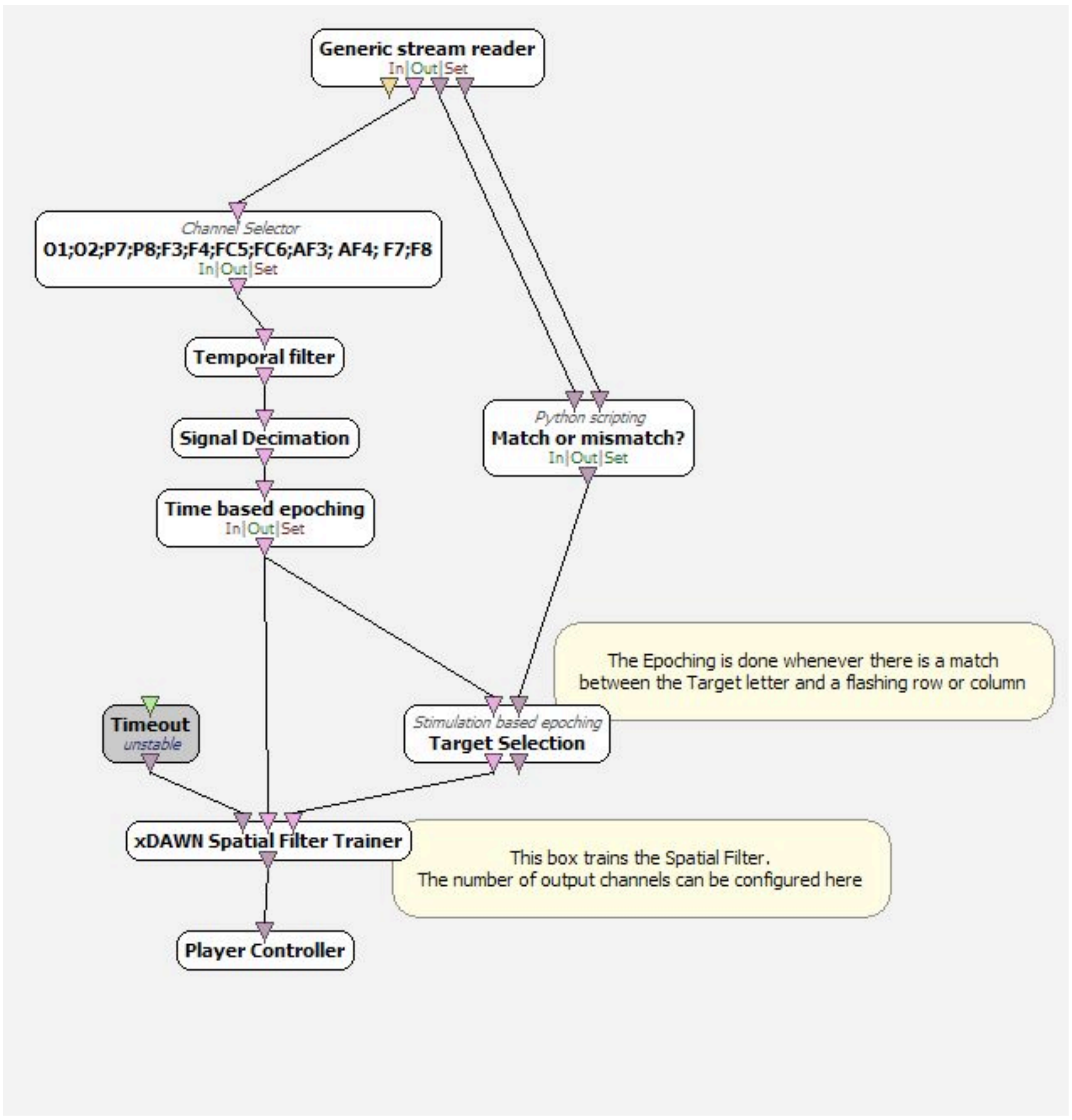
Capacitor information
All 10uF capacitors on this sheet are tantalums and have 16V tolerance
Calculating capacitance (Cxtal) for XTAL loading capacitors.
CL = (C1 * C2) / (C1 + C2) + Cs
CL = desired load capacitance according to crystal datasheet
C1 = load capacitor at driven end of oscillator (ie gate output)
C2 = load capacitor at other end
Cs = stray capacitance in the PCB (guesstimate a few pF)
If C1 = C2 = Cxtal, then CL = Cxtal / 2 + Cs
or Cxtal = 2 * (CL - Cs)
If PWR = +5V, replace D103 and IC107 (7805) with wires across their inputs and outputs, and mount D104.
If PWR >= +7V, mount D103 and IC107 as intended, but do not mount D104.
Note: ESD protected MAX232 available: MAX232E
ModularEEG - Isolated µC/ADC module.
See http://openeeg.sf.net for more information.
TITLE: modEEGdigital_v1.0
Authors: Dipl.-Ing.(FH) Joerg Hansmann and Andreas R
Document Number:
REV:
Date: 2003-05-01 13:12:33
Sheet: 1/1
PB1 is PWM output for cal-signal
Vcc digital
Vcc analog
digital GND
unbuffered 2.0V
AT90S4433P
IC102
J101
JP102
PINHD-2X17
MAX232
IC106
6N139
IC103
IC104
TMV0505S
IC105
IC107 7805
TL431CLP
X101 7.3728MHz

Appendix D: OpenVibe Scenarios

1. Scenario 1: Signal Acquisition


The Python UI displays the Keyboard and randomly generates sequences of "Flashing letters" and "Target Letters" for Training
Python scripting
Python UI
In|Out|Set
Player Controller
Acquisition client
The signal is taken from the acquisition Client
Identity
In|Out|Set
Identity
In|Out|Set
Channel Selector
O1;O2;P7;P8;F3;F4;FC5;FC6;AF3; AF4; F7;F8
In|Out|Set
Generic stream writer
In|Out|Set
The flashing secuence, the TArget Letters and the signal is recorded into a file using the "Generic Stream Writer"
Temporal filter
Signal Decimation
Identity
In|Out|Set
The singal on the electrodes shown on the list "Channel Selector" is displayed
Signal display
In|Out|Set

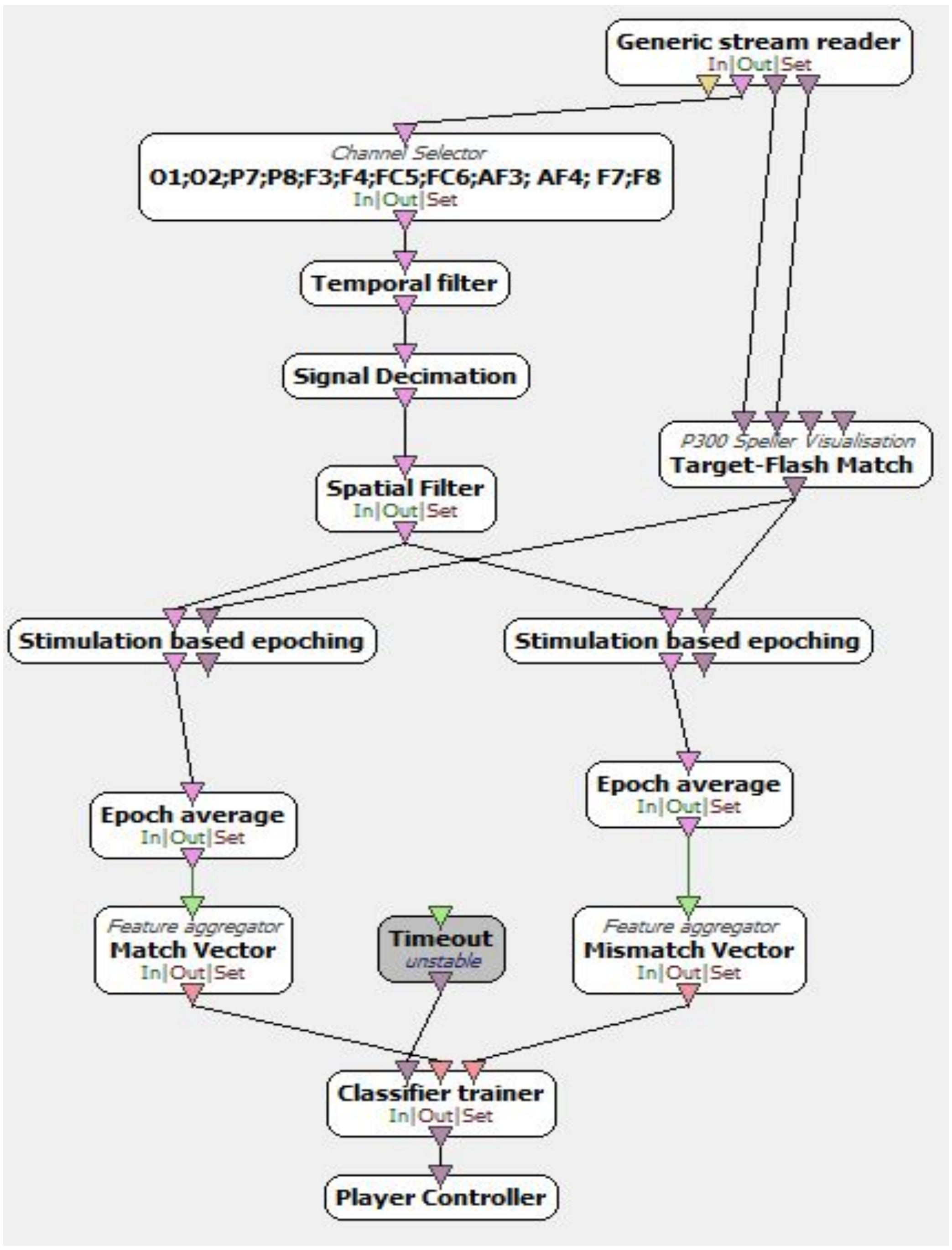
2. Scenario 2: Spatial Filter trainer



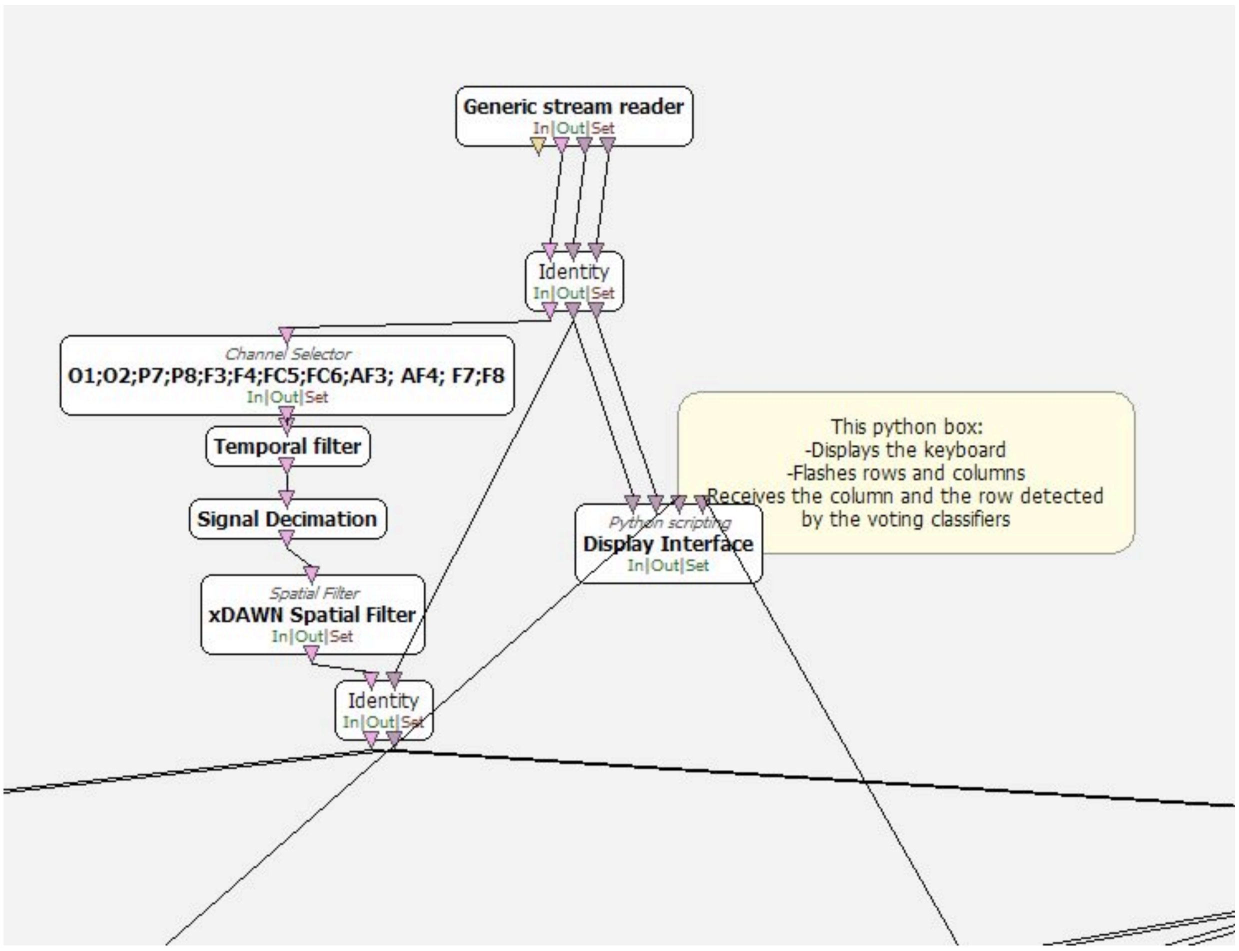
The Epoching is done whenever there is a match between the Target letter and a flashing row or column

This box trains the Spatial Filter.
The number of output channels can be configured here

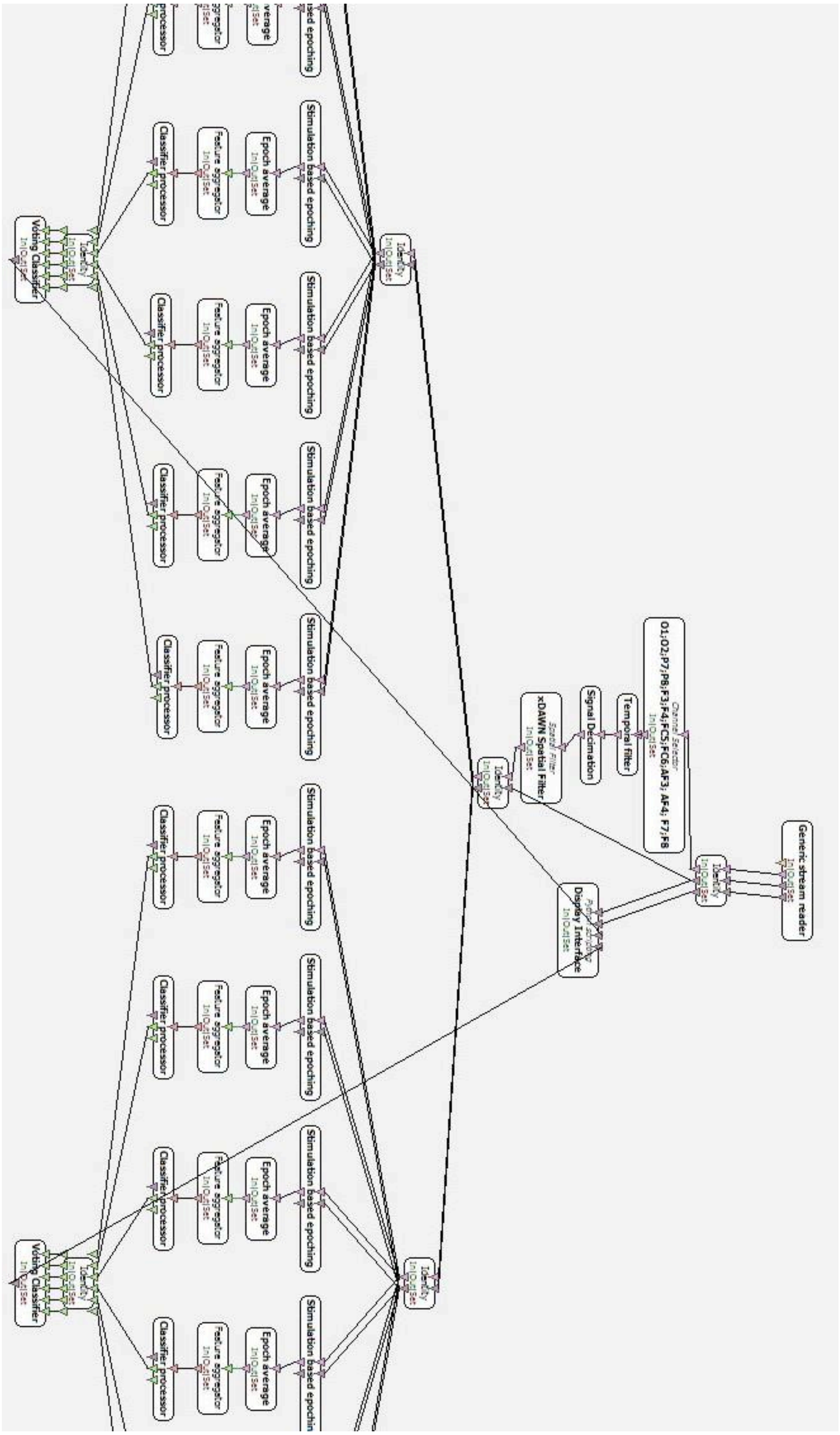
3. Scenario 3: Training Classifier



4. Scenario 4: Online Scenario



This python box:
-Displays the keyboard
-Flashes rows and columns
-Receives the column and the row detected by the voting classifiers



Generic stream reader
Identity
Channel Selector
O1;O2;P7;P8;F3;F4;FC5;FC6;AF3; AF4; F7;F8
Temporal filter
Signal Decimation
Spatial Filter
xDAWN Spatial Filter
Python scripting
Display Interface
Stimulation based epoching
Epoch average
Feature aggregator
Classifier processor
Voting Classifier
In|Out|Set

Appendix E: Microcontroller Code

The original code has been obtained from OpenEEG webpage (OpenEEG 2015) and then slightly modified to be up to the last AVR modifications. Modifications are in red.

```
/*
 * ModularEEG firmware for one-way transmission, v0.5.4-p2
 * Copyright (c) 2002-2003, Joerg Hansmann, Jim Peters, Andreas Robinson
 * Modification for Atmega48 (c) 2005, Yuri Smolyakov
 * License: GNU General Public License (GPL) v2
 * Compiles with AVR-GCC v3.3.
 *
 * Note: -p2 in the version number means this firmware is for packet version
2.
 */

//////////////////////////////////////////////////////////////
/*
////////// Packet Format Version 2 ////////////
// 17-byte packets are transmitted from the ModularEEG at 256Hz,
// using 1 start bit, 8 data bits, 1 stop bit, no parity, 57600 bits per
second.
// Minimial transmission speed is 256Hz * sizeof(modeeg_packet) * 10 = 43520
bps.

struct modeeg_packet
{
    uint8_t     sync0;      // = 0xA5
    uint8_t     sync1;      // = 0x5A
    uint8_t     version;    // = 2
    uint8_t     count;      // packet counter. Increases by 1 each packet
    uint16_t    data[6];    // 10-bit sample (= 0 - 1023) in big endian
(Motorola) format
    uint8_t     switches;   // State of PD5 to PD2, in bits 3 to 0
};

// Note that data is transmitted in big-endian format.
// By this measure together with the unique pattern in sync0 and sync1 it is
guaranteed,
// that re-sync (i.e after disconnecting the data line) is always safe.

// At the moment communication direction is only from Atmel-processor to PC.
// The hardware however supports full duplex communication. This feature
// will be used in later firmware releases to support the PWM-output and
// LED-Goggles.
*/

//////////////////////////////////////////////////////////////
/*
 * Program flow:
 *
 * When 256Hz timer expires: goto SIGNAL(SIG_OVERFLOW0)
 * SIGNAL(SIG_OVERFLOW0) enables the ADC
 *
 * Repeat for each channel in the ADC:
 * Sampling starts. When it completes: goto SIGNAL(SIG_ADC)
 * SIGNAL(SIG_ADC) reads the sample and restarts the ADC.
 *
 * SIGNAL(SIG_ADC) writes first byte to UART data register
 * (UDR) which starts the transmission over the serial port.
 *
 * Repeat for each byte in packet:
 * When transmission begins and UDR empties: goto SIGNAL(SIG_UART_DATA)
 *
 * Start over from beginning.
 */

#include <avr/io.h>
#include <inttypes.h>
#include <avr/interrupt.h>
#include <avr/wdt.h>

#define outb(port,val) \
(port) = (val)
#define inp(port) \
(port)
#define sbi(port,bit) \
(port) |= (1 << (bit))
#define cbi(port,bit) \
(port) &= ~(1 << (bit))
#define BV(bit) \
(1 << (bit))

#define NUMCHANNELS 6
#define HEADERLEN 4
#define PACKETLEN (HEADERLEN + NUMCHANNELS * 2 + 1)

#define FOSC 7372800        // Clock Speed
#define SAMPFREQ 256
#define TIMER0VAL 256 - ((FOSC / 256) / SAMPFREQ)

#define BAUD 57600
#define BAUDL FOSC/16/BAUD - 1
#define BAUDH (BAUDL)>>8


#if defined (__AVR_ATmega48__) || defined (__AVR_ATmega88__)
    #define SIG_UART_DATA SIG_USART_DATA
    #define ADCSR   ADCSRA
    #define UCSRB   UCSR0B
    #define UDRIE   UDRIE0
    #define UDR     UDR0
    #define TIMSK   TIMSK0
    #define TCCR0   TCCR0B
#endif

//char const channel_order[] = {0, 3, 1, 4, 2, 5};
char const channel_order[] = {0, 1, 2, 3, 4, 5};

// The transmission packet
volatile uint8_t TXBuf[PACKETLEN];

// Next byte to read or write in the transmission packet
volatile uint8_t TXIndex;

// Current channel being sampled
volatile uint8_t CurrentCh;

/////////////////////////////////////////////
// Sampling timer (timer 0) interrupt handler
SIGNAL(TIMER0_OVF_vect){
    outb(TCNT0, TIMER0VAL); //Reset timer to get correct sampling frequency
    CurrentCh = 0;
    // Write header and footer:
    // Increase packet counter (fourth byte in header)
    TXBuf[3]++;

    //Get state of switches on PD2..5, if any (last byte in packet)
    TXBuf[2 * NUMCHANNELS + HEADERLEN] = (inp(PIND) >> 2) & 0x0F;

    cbi(UCSRB, UDRIE);  //Ensure UART IRQ's are disabled
    sbi(ADCSR, ADIF);   //Reset any pending ADC interrupts
    sbi(ADCSR, ADIE);   //Enable ADC interrupts

    //The ADC will start sampling automatically as soon
    //as sleep is executed in the main-loop
}

////////////////////////////////////////////
// AD-conversion-complete interrupt handler
SIGNAL(ADC_vect) {
    volatile uint8_t i;

    i = 2 * CurrentCh + HEADERLEN;

    TXBuf[i+1]  = inp(ADCL);    // Fill buffer from ADC
    TXBuf[i]    = inp(ADCH);

    CurrentCh++;
    if (CurrentCh < NUMCHANNELS) {
        outb(ADMUX, channel_order[CurrentCh]);  //Select the next channel
        //The next sampling is started automatically
    } else {
        outb(ADMUX, channel_order[0]);  //Prepare next conversion, on
channel 0

            // Disable ADC interrupts to prevent further calls to SIG_ADC
        cbi(ADCSR,  ADIE);

        // Hand over to SIG_UART_DATA, by starting
        // the UART transfer and enabling UDR IRQ's
        outb(UDR,  TXBuf[0]);
        sbi(UCSRB, UDRIE);

        TXIndex = 1;
    }
}

//////////////////////////////////////////////////////////
// UART data transmission register-empty interrupt handler
SIGNAL(USART_UDRE_vect) {
    outb(UDR, TXBuf[TXIndex]);  //Send next byte

    TXIndex++;
    if (TXIndex == PACKETLEN){  //See if we're done with this packet
        cbi(UCSRB, UDRIE);      //Disable SIG_UART_DATA interrupts
        //Next interrupt will be a SIG_OVERFLOW0
    }
}

////////////////////////////////////////////////////////
// Initialize PWM output (PB1 = 14Hz square wave signal)
void pwm_init(void) {
    // Set timer/counter 1 to use 10-bit PWM mode.
    // The counter counts from zero to 1023 and then back down
    // again. Each time the counter value equals the value
    // of OCR1(A), the output pin is toggled.
    // The counter speed is set in TCCR1B, to clk / 256 = 28800Hz.
    // Effective frequency is then clk / 256 / 2046 = 14 Hz

#if defined (__AVR_ATmega8__) || defined (__AVR_ATmega48__) || defined
(__AVR_ATmega88__)
    outb(OCR1AH, 2);        // Set OCR1A = 512
    outb(OCR1AL, 0);
    outb(TCCR1A, BV(COM1A1) + BV(WGM11) + BV(WGM10)); // Set 10-bit PWM mode
    outb(TCCR1B, (1 << CS12));  // Start and let run at clk / 256 Hz
#else   //__AVR_AT90S4433
    outb(OCR1H, 2);      // Set OCR1 = 512
    outb(OCR1L, 0);
    outb(TCCR1A, BV(COM11) + BV(PWM11) + BV(PWM10)); // Set 10-bit PWM mode
    outb(TCCR1B, (1 << CS12));  // Start and let run at clk / 256 Hz
#endif
}

////////////////
// Initialize uC
void init(void) {
    //Set up the ports
    outb(DDRD,  0xC2);
    outb(DDRB,  0x07);
    outb(PORTD, 0xFF);
    outb(PORTB, 0xFF);

    //Select sleep mode = idle
#if defined (__AVR_ATmega48__) || defined (__AVR_ATmega88__)
    outb(SMCR, (inp(SMCR)  | BV(SE)) & (~BV(SM0) | ~BV(SM1) | ~BV(SM2)));
#elif defined (__AVR_ATmega8__)
    outb(MCUCR,(inp(MCUCR) | BV(SE)) & (~BV(SM0) | ~BV(SM1) | ~BV(SM2)));
#else // __AVR_AT90S4433__
    outb(MCUCR,(inp(MCUCR) | BV(SE)) & (~BV(SM)));
#endif

    // Initialize the ADC
    // Timings for sampling of one 10-bit AD-value:
    // prescaler > ((XTAL / 200kHz) = 36.8 =>
    // prescaler = 64 (ADPS2 = 1, ADPS1 = 1, ADPS0 = 0)
    // ADCYCLE = XTAL / prescaler = 115200Hz or 8.68 us/cycle
    // 14 (single conversion) cycles = 121.5 us (8230 samples/sec)
    // 26 (1st conversion) cycles = 225.69 us
    outb(ADMUX, 0);      //Select channel 0

    //Prescaler = 64, free running mode = off, interrupts off
    outb(ADCSR, BV(ADPS2) | BV(ADPS1));
    sbi(ADCSR, ADIF);    //Reset any pending ADC interrupts
    sbi(ADCSR, ADEN);    //Enable the ADC

    // Analog comparator OFF
    outb(ACSR, BV(ACD));

    //Initialize the UART
#if defined (__AVR_ATmega48__) || defined (__AVR_ATmega88__)
    outb(UBRR0H, BAUDH);         //Set speed to BAUD bps
    outb(UBRR0L, BAUDL);
    outb(UCSR0A, 0);
    outb(UCSR0C, BV(UCSZ01) | BV(UCSZ00));
    outb(UCSR0B, BV(TXEN0));     //Enable transmitter
#elif defined (__AVR_ATmega8__)
    outb(UBRRH, BAUDH);          //Set speed to BAUD bps
    outb(UBRRL, BAUDL);
    outb(UCSRA, 0);
    outb(UCSRC, BV(URSEL) | BV(UCSZ1) | BV(UCSZ0));
    outb(UCSRB, BV(TXEN));
#else // __AVR_AT90S4433__
    outb(UBRR, BAUDL);          //Set speed to BAUD bps
    outb(UCSRB, BV(TXEN));
#endif

    // Initialize timer 0
    outb(TCNT0, 0);         //Clear it
    outb(TCCR0, 4);         //Start it. Frequency = clk / 256
    outb(TIMSK, BV(TOIE0));     //Enable the interrupts
}

//////////////////////////////////////////////////////////
int main(void) {

    // Write packet header
    TXBuf[0] = 0xA5;    //Sync 0
    TXBuf[1] = 0x5A;    //Sync 1
    TXBuf[2] = 2;       //Protocol version
    TXBuf[3] = 0;       //Packet counter

    // Initialize
    init();

    // Initialize PWM (optional)
    pwm_init();

    sei();
    // Now, we wait
    // This is an event-driven program,
    // so nothing much happens here
    while (1) {
        __asm__ __volatile__ ("sleep"); // sleep until something happens
    }
}
```

Appendix F: Optimal Electrode Positioning for P300 Detection

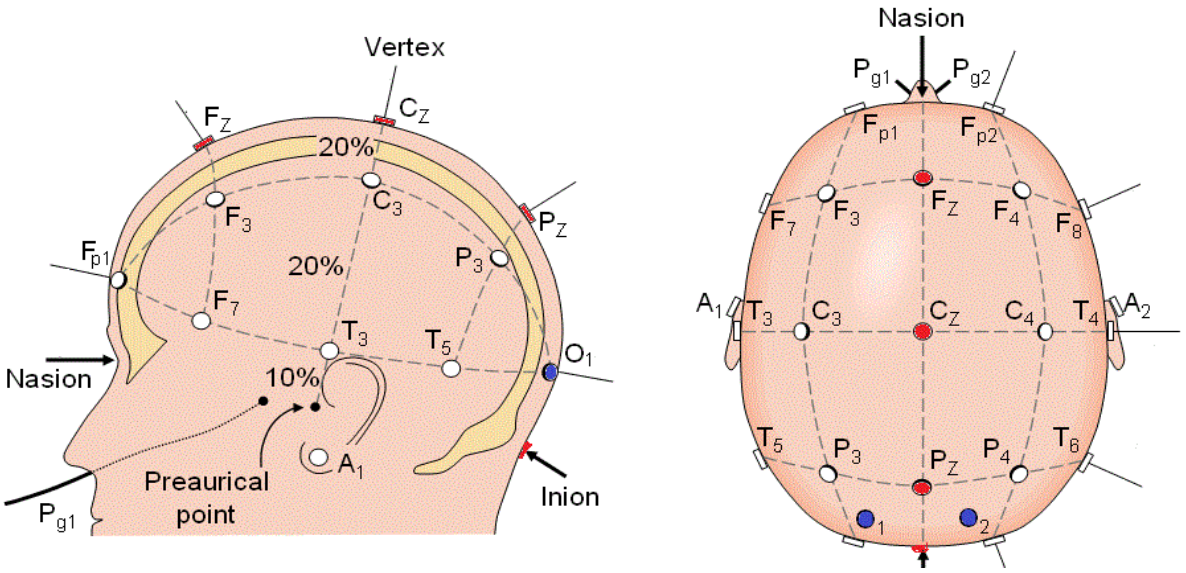


Figure 18: Optimal electrode positioning for P300 detection (Red circles) according to Nicolas-Alonso & Gomez-Gil (2012)

Appendix G: Python Code

The following is the Python code used to implement the signal acquisition:

```
#dependencies
from __future__ import division, print_function, unicode_literals
import os, sys
from pyglet.gl import *
from pyglet import *
from pyglet.window import *
import primitives
import user_input
import word_predictor
import random
import string
import ctypes
#set stimulation ids
OVTK_StimulationId_Target = 33285
OVTK_StimulationId_Label_00 = 33024
OVTK_StimulationId_Label_01 = 33025
OVTK_StimulationId_Label_07 = 33031
# Keyboard matrix
text = [ ['predtxt1','predtxt2','predtxt3','A','B',u"\u2190"],
                ['C','D','E','F','G','ENTER'],
                ['H','I','J','K','L','abc'],
                ['M','N','O','P','Q','123'],
                ['R','S','T','U','V','W'],
                ['X','Y','Z','SPACE',u"\u25C4", u"\u25BA"] ]
# Get window parameters
user32 = ctypes.windll.user32
width = user32.GetSystemMetrics(0)
height = user32.GetSystemMetrics(1)
width = width
height = height
############################# CONTROLS #############################
#P300 flash modes
```

```python
isEnlargeTextMode = True
isHighlightTextMode = False
#condition to draw flash
isDrawVertFlash = True
isDrawHorizFlash = True
#condition to draw target
isDrawTarget = True
#target parameters
targetSize = [width/6,height/12]
#general UI display paramaters
backgroundColour = [0,0,1,1] #1 corresponds to 255 last value
is alpha
#highlight mode parameters
targetColour = [0,1,0,1]
vertFlashColour = [0,1,1,1]
vertFlashSize = [width/6,height/2]
horizFlashColour = [0,1,1,1]
horizFlashSize = [width,height/12]
#text parameters
keyboardFontSize = 36
keyboardFontColour = [230,230,230,255]
keyboardEnlargeFontSize = 50
keyboardEnlargeFontColour = [255,255,0,255]
#timing
targetDelay = 30
#UIsuze
UISize = 10
#UIscaling based on UISize
widgetPositionY = UISize*height/12
widgetHeight = height-widgetPositionY
keyboardPositionTop = widgetPositionY - height/12
###############################################################
#####
#OpenVibe class
class MyOVBox(OVBox):
    def __init__(self):
        OVBox.__init__(self)

    def initialize(self):
        # Called once when starting the scenario
        self.loopCounter = 0
        self.target = [0,0]
        self.hashTable = {12:5,11:4,10:3,9:2,8:1,7:0,6:6,5:7,4:8,3:9,2:10,1:11}
        self.current_text = ""
        # Read files into lists for flashes and targets, and convert strings to ints
        with open('dep_files/flash_stims.txt') as f:
            self.flashes = f.read().splitlines()
            self.flashes = [int(x) for x in self.flashes]
        with open('dep_files/target_stims.txt') as f:
            self.targets = f.read().splitlines()
            self.targets = [int(x) for x in self.targets]
        # I/O
        self.initOutputs() # Set output stimulation headers
        # Pyglet
        #create window
        self.win = window.Window(fullscreen = True)
        #colour background and set up openGL rendering
        glClearColor(backgroundColour[0], backgroundColour[1], backgroundColour[2], backgroundColour[3])
        glEnable(GL_BLEND)
        glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA)
        #set up keyboard input
        self.keys = key.KeyStateHandler()
        self.win.push_handlers(self.keys)
        #set up user input text display
        self.batch = pyglet.graphics.Batch()
        self.widget = user_input.TextWidget('', 0, int(widgetPositionY), int(width),int(widgetHeight), self.batch)
        self.text_input = ""
        self.widget.caret.on_text(self.text_input)
        #set up user keyboard matrix
        self.matrix = []
        for j in range(0, len(text) ):
            row = []
            for i in range(0, len(text[j]) ):
                line = text[j][i]
                ypos = keyboardPositionTop -(j)*(keyboardPositionTop/5)
                xpos = i*width/6
                temp = pyglet.text.Label(line,
                    font_name='Courier New',
                    font_size=keyboardFontSize,
                    color=(keyboardFontColour[0],keyboardFontColour[1],keyboardFontColour[2],keyboardFontColour[3]),
                    x=xpos, y=ypos,
                    anchor_x='left', anchor_y='bottom')
                row.append(temp)
            self.matrix.append(row)
        #set up window rendering
        self.win.dispatch_events()
        self.win.flip()
        return
    def endExperiment(self):
        print("Quitting experiment.")
        self.closeOutputs()
        self.sendOutput(0, OVTK_StimulationId_Label_00)
        self.win.close()
        return
    def readFlash(self):
        if len(self.flashes) < 1:
            print("Reached end of flashes file.")
            self.endExperiment()
        else:
            return self.flashes.pop(0)
        return None
    def getNextTarget(self):
        if len(self.targets) < 1:
            print("Reached end of target file.")
            self.endExperiment()
        else:
            self.target[0] = self.targets.pop(0)
            self.target[1] = self.targets.pop(0)
        return
    def drawTarget(self, rowStim, colStim):
        if (isDrawTarget):
            rowNum = rowStim - OVTK_StimulationId_Label_01
            colNum = colStim - OVTK_StimulationId_Label_07
            x = colNum * width / 6
            y = (5-rowNum) * (keyboardPositionTop/5)
            primitives.drawRect(x, y, targetSize[0], targetSize[1], targetColour[0],targetColour[1],targetColour[2],targetColour[3])
        return
    def startFlash(self, rowcol):
        # If column
        if (rowcol <= 5 and isDrawVertFlash):
            if (isEnlargeTextMode):
                c = rowcol
                for r in range(0, len(self.matrix)):
                    self.matrix[r][c].font_size = keyboardEnlargeFontSize
                    self.matrix[r][c].color = (keyboardEnlargeFontColour[0],keyboardEnlargeFontColour[1],keyboardEnlargeFontColour[2],keyboardEnlargeFontColour[3])
            if (isHighlightTextMode):
                primitives.drawRect(rowcol*width/6, 0, vertFlashSize[0], vertFlashSize[1], vertFlashColour[0],vertFlashColour[1],vertFlashColour[2],vertFlashColour[3])
        # If row
        elif (rowcol <= 11 and isDrawHorizFlash):
            if (isEnlargeTextMode):
                r = rowcol%6
                for c in range(0, len(self.matrix[r])):
                    self.matrix[r][c].font_size = keyboardEnlargeFontSize
                    self.matrix[r][c].color = (keyboardEnlargeFontColour[0],keyboardEnlargeFontColour[1],keyboardEnlargeFontColour[2],keyboardEnlargeFontColour[3])
            if (isHighlightTextMode):
                primitives.drawRect(0, rowcol%6*height/12,
horizFlashSize[0], horizFlashSize[1], horizFlashColour[0], horizFlashColour[1], horizFlashColour[2], horizFlashColour[3])
        return
    def stopFlash(self, rowcol):
        if (isEnlargeTextMode == False):
            return # No need to stopFlash if text is not enlarged
        # If column
        if (rowcol <= 5 and isDrawVertFlash):
            c = rowcol
            for r in range(0, len(self.matrix)):
                self.matrix[r][c].font_size = keyboardFontSize
                self.matrix[r][c].color = (keyboardFontColour[0],keyboardFontColour[1],keyboardFontColour[2],keyboardFontColour[3])
        # If row
        elif (rowcol <= 11 and isDrawHorizFlash):
            r = rowcol%6
            for c in range(0, len(self.matrix[r])):
                self.matrix[r][c].font_size = keyboardFontSize
                self.matrix[r][c].color = (keyboardFontColour[0],keyboardFontColour[1],keyboardFontColour[2],keyboardFontColour[3])
        return
    def process(self): # Called on each box clock tick (this can be configured by right-clicking the box)
        self.win.dispatch_events()
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT) # Set up background
        if self.win.has_exit:
            self.endExperiment()
        else:
            # Show a target for the first target
            if (self.loopCounter <= targetDelay):
                if (self.loopCounter == 0):
                    self.getNextTarget()
                    self.sendOutput(2, self.target[0])
                    self.sendOutput(2, self.target[1])
                self.drawTarget(self.target[0], self.target[1])
            # Flash for the next 50 loops
```

```python
            elif (self.loopCounter <= targetDelay + 50):
                newStim = self.readFlash()
                # Aim row/column flash
                if (33025 <= newStim and newStim <= 33036):
                    self.flash = self.hashTable[newStim - OVTK_StimulationId_Label_00]
                # Start flash
                elif (newStim == 32779):
                    self.startFlash(self.flash)
                # Stop flash
                elif (newStim == 32780):
                    self.stopFlash(self.flash)
                self.sendOutput(1, newStim)
                # Reset counter on last loop
                if (self.loopCounter == targetDelay + 50):
                    self.loopCounter = -1
            #get input from user
            if (self.keys[key.A]):
                self.text_input = "temp"
                self.current_text = self.current_text + self.text_input
                self.widget.caret.on_text(self.text_input)
                self.text_input = ""
            ## Predictive text
            corrected_text = word_predictor.correct(self.current_text)
            ypos = keyboardPositionTop - (0)*(keyboardPositionTop/5)
            xpos  = 0*width/6
            temp0 = pyglet.text.Label(corrected_text[0],
                    font_name='Courier New',
                    font_size=keyboardFontSize,
                    color=(keyboardFontColour[0],keyboardFontColour[1],keyboardFontColour[2],keyboardFontColour[3]),
                    x=xpos, y=ypos,
                    anchor_x='left', anchor_y='bottom')

            ypos = keyboardPositionTop - (0)*(keyboardPositionTop/5)
            xpos  = 1*width/6
            temp1 = pyglet.text.Label(corrected_text[1],
                    font_name='Courier New',
                    font_size=keyboardFontSize,
                    color=(keyboardFontColour[0],keyboardFontColour[1],keyboardFontColour[2],keyboardFontColour[3]),
                    x=xpos, y=ypos,
                    anchor_x='left', anchor_y='bottom')

            ypos = keyboardPositionTop - (0)*(keyboardPositionTop/5)
            xpos  = 2*width/6
            temp2 = pyglet.text.Label(corrected_text[2],
                    font_name='Courier New',
                    font_size=keyboardFontSize,
                    color=(keyboardFontColour[0],keyboardFontColour[1],keyboardFontColour[2],keyboardFontColour[3]),
                    x=xpos, y=ypos,
                    anchor_x='left', anchor_y='bottom')
            self.matrix[0][0] = temp0
            self.matrix[0][1] = temp1
            self.matrix[0][2] = temp2
            # Draw keyboard matrix
            for r in range (0,len(self.matrix)):
                for c in range(0, len(self.matrix[r])):
                    self.matrix[r][c].draw()
            # Pyglet/GL updates
            self.batch.draw()
            self.win.flip()
            self.loopCounter += 1
        return
    def uninitialize(self): # Called once when stopping the scenario
        return
    def initOutputs(self):
        print(len(self.output))
        for index in range(len(self.output)):
            # OV protocol requires an output stim header; dates are 0
            self.output[index].append(OVStimulationHeader(0., 0.))
        return
    def sendOutput(self, index, stimLabel):
        # A stimulation set is a chunk which starts at current time and end time is the time step between two calls
        stimSet = OVStimulationSet(self.getCurrentTime(), self.getCurrentTime()+1./self.getClock())
        stimSet.append( OVStimulation(stimLabel, self.getCurrentTime(), 0.) )
        self.output[index].append(stimSet)
        return
    def closeOutputs(self):
        for index in range(len(self.output)):
            # OV protocol requires an output stim end
            end = self.getCurrentTime()
            self.output[index].append(OVStimulationEnd(end, end))
        return
box = MyOVBox()
```