Brain Computer Interface

María Madrid Sobrino
Telecommunication Engineering (Visiting Student)

Abstract - The paper presents an EEG-based brain-computer interface (BCI) in which subjects could select a picture from a set on a computer screen. The application is centred on detecting steady-state visual evoked potentials (SSVEP) in EEG signals recorded on the scalp of the subject. BCI2000 software platform is used in this project as a basis for the whole system. The platform will link its modules and the developed ones needed to achieve the closed-loop BCI system.
In this context, a C++ computer application with 16 targets and a MATLAB signal processing module were then implemented using the proposed method. In offline tests for a set of frequencies with differences of amplitude up to 15 dB, detection was achieved. Detection was also achieved in online tests.

1. Introduction

1.1 BCI concepts

A brain-computer interface (BCI) is a communication system in which the user's intention is conveyed to the external world without involving the normal output pathways of peripheral nerves and muscles [1]. The concept of a brain-computer interface stems from a need for alternative, augmentative communication, and control options for individuals with severe disabilities, though its potential uses extend to rehabilitation of neurological disorders, brain-state monitoring, and gaming [2].

Nowadays, there are two approaches to carry a BCI system out. A BCI is called invasive if the signal acquisition system (i.e. electrodes) needs to be implanted directly into the brain. In another case, the system could be developed without surgery and the signal acquisition system would be placed over the scalp. In this case, the BCI is called non-invasive. The most practical and widely applicable are those based on non-invasive electroencephalogram (EEG) measurements recorded from the scalp. These EEG measurements provide information from the brainwaves by recording the electrical activity along the scalp. However, Magnetoencephalography (MEG) and functional magnetic resonance imaging (fMRI) have both been used successfully as non-invasive BCIs [3, 4].

Non-invasive BCI's based on EEG generally utilize either event-related potentials (ERPs) such as P300, visual evoked potentials (VEP) measures or steady-state visual evoked potentials (SSVEP) [2]. ERPs are electrocortical potentials generated in the brain during the presentation of stimulus. The stimulus could be generated by a sensor or a psychological event. It generates a time delay wave in EEG that can be detected after processing EEG signals [5]. On the other hand, VEP consist of fewer waves, namely, those deriving from the activity of the cerebral cortex [6]. Different wave forms generated by visual stimuli can be distinguished on the basis of the latency of their appearance. VEP potentials are called “transient” because the slow rate of stimulation allows the sensory pathways to recover or “reset” before the next stimulus appears. When visual stimuli are presented at a constant rate that is rapid enough to prevent the evoked neural activity from returning to base line state, the elicited response becomes continuous and is called the steady-state visual evoked potential (SSVEP). At rapid stimulation rates, the brain response to the stimulus becomes sinusoidal [6].

The implemented BCI presented here uses the SSVEP paradigm. SSVEP-based BCI’s rely on the psychophysiological properties of EEG brain responses produced during the periodic presentation of a flickering stimulus [7]. When the stimulus is being presented the subject’s brain produces EEG signals that resonate at the stimulus rate and its multipliers. In some research works (such as [8]) phase has also been use as a distinctive parameter for a particular RVS. The effectiveness of SSVEP-based BCI designs is due to several factors like the high signal-to-noise ratios that could be achieved [2] and the short training required. However, certain stimulation frequencies can provoke epileptic seizures and induce fatigue [9].

1.2. Introduction to the approach

In order to increase the usability and the possibilities of a SSEVP-based BCI, a classification application is proposed. Each command or target is associated with a repetitive visual stimulus (RVS) that has a distinctive frequency [9]. A total of 4 stimuli are simultaneously presented to the user who selects a target by focusing his/her attention on the corresponding stimulus.

The application presented allows the user to select within 16 choices. The choices are shown into 4 expandable groups. This way, 16 choices are available by using only 4 commands. However, selecting one picture entail a double selection. First, the user selects one of the 4 groups and once the group is expanded he/she selects the desired picture. Screenshots of the application are shown below in Figure 1 and Figure 2.

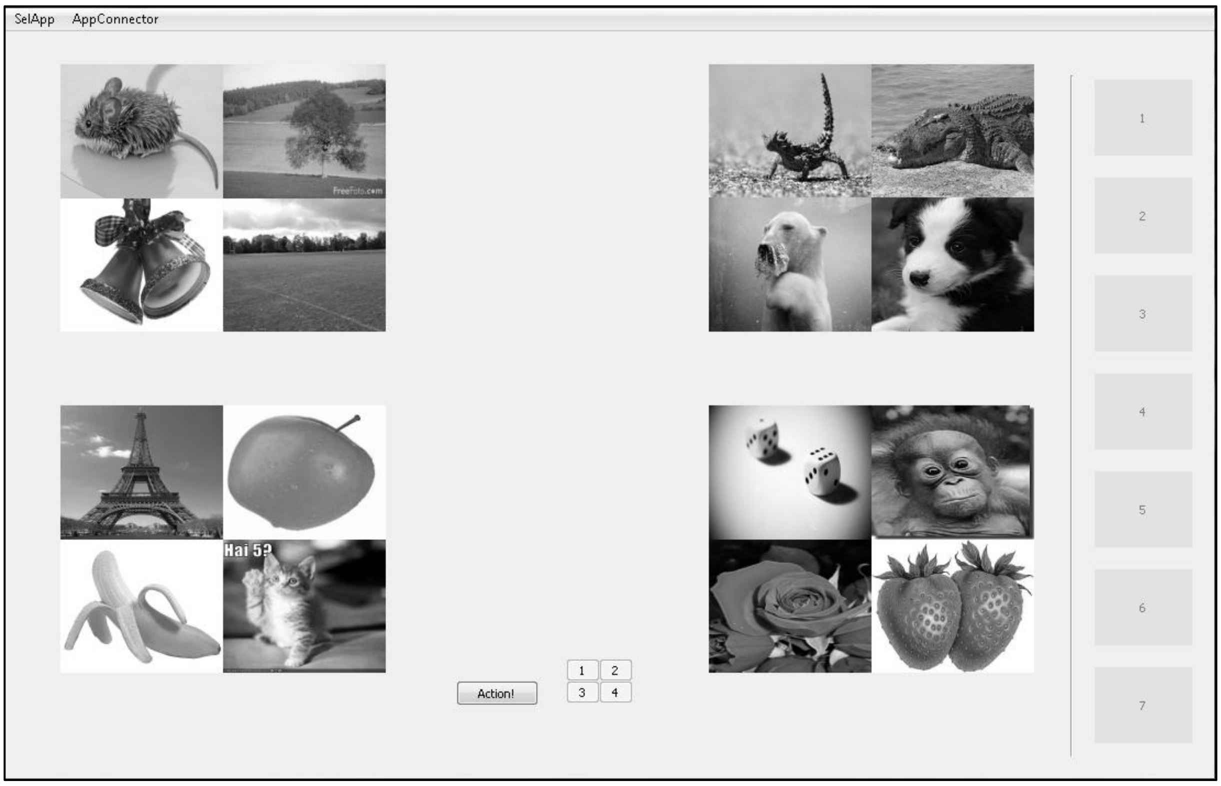
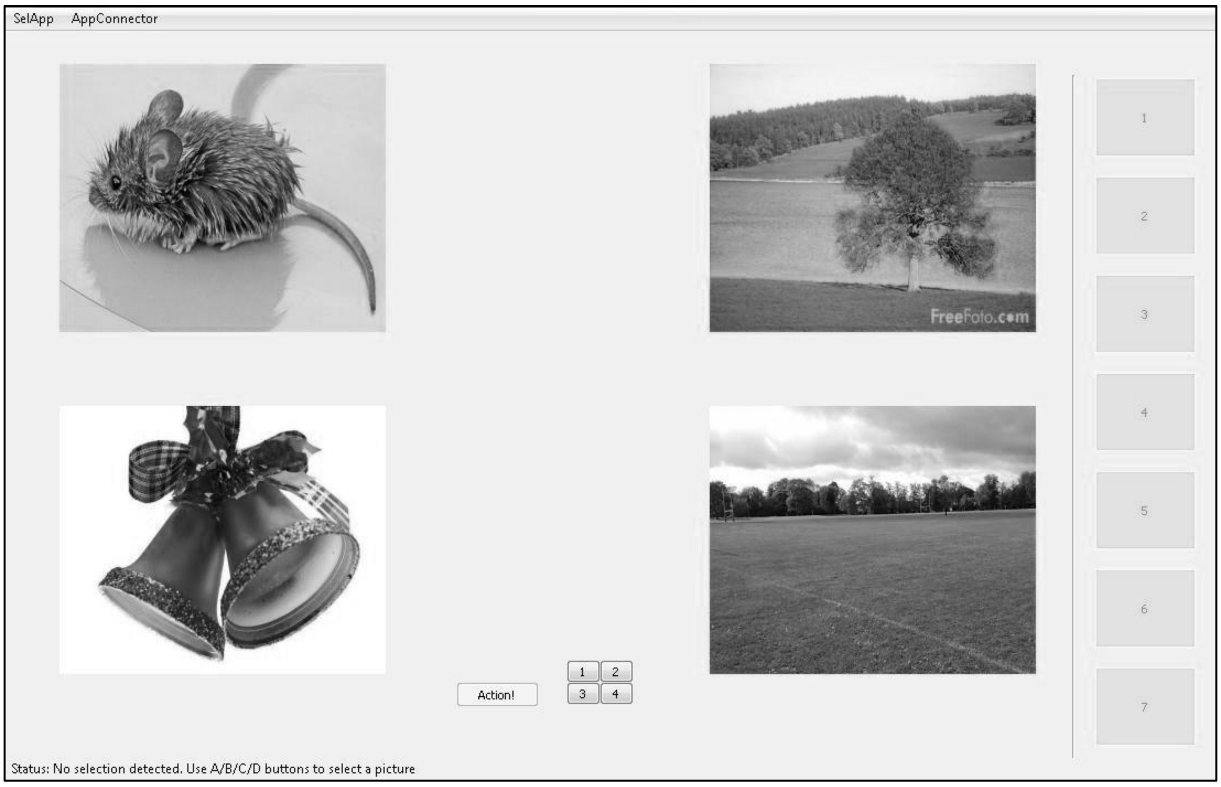


Figure 1: Application overview before selection



Status: No selection detected. Use A/B/C/D buttons to select a picture

Figure 2: Application overview after the first selection

To fully build a BCI system from scratch developers need to care about the data acquisition, the processing of the obtained data and the stimuli presentation. By grouping these modules, a BCI closed loop is constituted. This loop can be sustained by open source software platforms-such as BCI2000 [10] or OpenViBE [11], which provide a variety of data acquisition systems, brain signals, and study/feedback paradigms.

The mentioned general-purpose systems enable to design, test and use BCIs. The present project uses BCI2000 as basis for the development. Both signal processing module and application module were built and then integrated into the BCI2000 platform.

2. Background

The major part of the background of this project is presented along the Introduction section; however, there is some specific research about particular features that is presented below.

A comparison between different performances was carried out in order to see what system achieved the best operation. Table 1 shows the results. Note: ITR stands for Iteration Transfer Rate.

Table 1: Performance overview

Source	Method	ITR [bits/min]	Freqs. [Hz]	Targets	Time per selection [s]
Jia et al. [8]	Fourier coefficient projections	60	10, 12, 15	15	2.5*
Wang et al. [12]	Power spectrum analysis	74.5	10, 11, 12	16	3.08
Parini et al. [13]	Spatial Filtering and Channel Combining	51.47	6, 7, …, 17	4	2
Bin et al. [14]	Canonical Correlation analysis (CCA)	58	6.7, 7.5, 8.6, 10, 12, 15	6	2
*Data length of each trial is 2s and there is 0.5 rest time after each selection.					

When looking for information about electrodes positioning, 3 papers were examined. These papers presented SSVEP based BCI's and different electrodes configuration according to the international 10–20 system. Table 2 shows the positioning they applied.

Table 2: Electrode positioning overview

Source	Number of electrodes	Electrode positions
Lalor et al. [2]	2	O1, O2
Martinez et al. [15]	5	CPz, Pz, POz, P1, P2, Fz
JJ Vidal et al. [16]	6	Pz-Oz, O1-Oz, O2-Oz, I-Oz, Oz-A, Fz-Pz

3. Methods

3.1. BCI2000 and system overall

BCI2000 software platform was chosen as basis for the developed Brain Computer Interface. BCI2000, as seen in its website [10], is a general-purpose system for brain-computer interface research and it is free for academic and research institutions. It can also be used for data acquisition, stimulus presentation, and brain monitoring applications. BCI2000 supports a variety of data acquisition systems, brain signals, and study/feedback paradigms. BCI2000 also facilitates interactions with other software such as MATLAB.

BCI2000 comes out of the box with proven support for different data acquisition hardware, signal processing routines, and experimental paradigms [10]. So, the developer could just have his/her own BCI built system by simply installing the provided software and connecting one of the supported acquisition hardware.

The BCI2000 platform is composed by four modules. These modules handle acquisition of brain signals (i.e., Source module), processing of these brain signals (Signal Processing module), user feedback (i.e., User Application module), and the interface to the investigator (i.e., Operator module), respectively. The Operator module is just the GUI (Graphical User Interface) seen by the researcher. One of the goals for BCI2000 is for each module to be as independent of the others as possible. As mentioned before, there is no need to replace or build any of those modules in order to achieve a working system; however, in order to achieve an SSVEP-based BCI, the application module was replaced. The Signal Processing module was also replaced in order to test the viability and the performance when using the MATLAB engine. Figure 3 shows an overall of the developed system, including the relevant modules of BCI2000 and their interactions with the designed approach.

To set the BCI2000 environment, the latest binary release can be downloaded and later installed. The source code could also be downloaded and then built by following the guide provided by BCI2000 team available at [17].

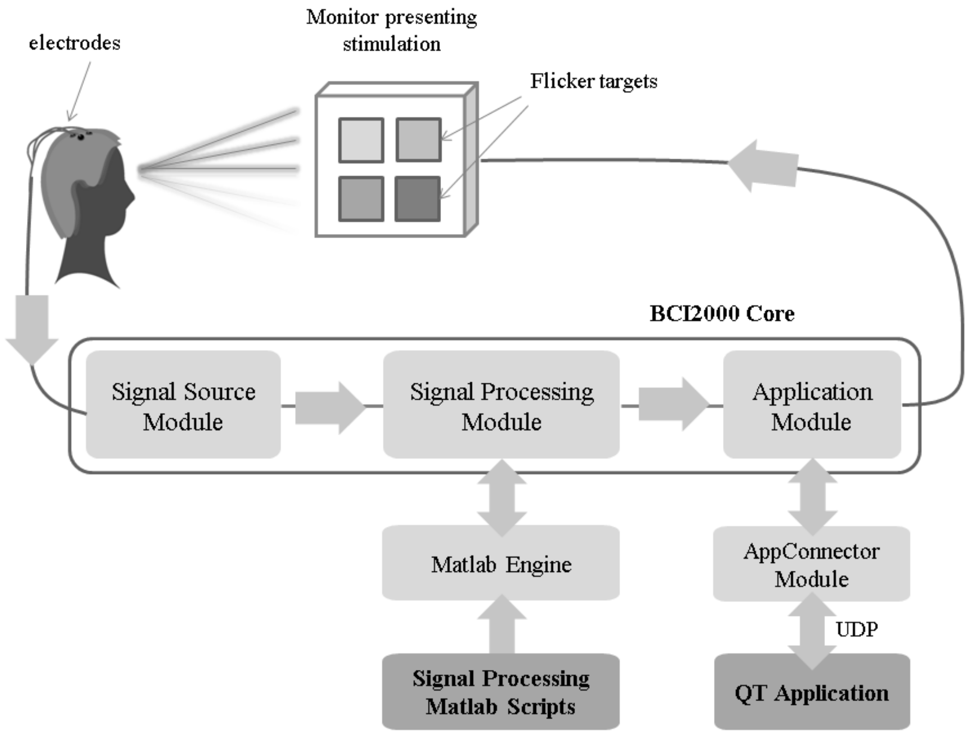


Figure 3: System overall. BCI2000 platform and MATLAB engine in light grey; Developed parts in dark grey.

3.2. Signal Processing module

BCI2000 Signal Processing module acts like a black box to the rest of the system - it receives brain signals from the Signal Source and sends control signals on to the Application. BCI2000 allows developers to use several ready-to-use signal processing modules integrated in the platform but it also allows producing our own module. There are two possibilities when developing a signal processing module for BCI2000; either building a C++ module integrated in BCI2000 or performing the signal processing in MATLAB which will interact with

BCI2000, receiving input data and returning the output of the processing back to it as shown above in Figure 3.

In the case described here, the MATLAB signal processing was used rather than building a C++ integrated module. For this purpose BCI2000 provides a convenient and simple programming interface detailed now. The signal processing component is implemented as a set of MATLAB scripts in the format described by BCI2000. The *MatlabSignalProcessing* module implements a mechanism for using the MATLAB engine within the BCI2000 pipeline. While BCI2000 is running, each block of data is pushed to the MATLAB engine and a well-specified MATLAB function (i.e. *.m file) is executed. The most relevant MATLAB scripts that could be implemented are listed and detailed below.

- bci_Construct: Performs any states and parameters initialization. Requests BCI2000 parameters and states by returning parameter and state definition lines.
- bci_Preflight: Used to check parameters for consistency. Also, it reports output signal dimensions.
- bci_Initialize: Determination of filter coefficients.
- bci_StartRun: Used to reset filter state at the beginning of each run.
- bci_Process: Process a signal input (a single data block) according to the signal processing chain, and return the result of processing in a signal output variable.

Parameters are used by BCI2000 to configure the signal source, signal processing and application modules. States refer to the states of operation that apply to a BCI2000 system as a whole.

If either of the bci_Preflight, bci_Initialize or bci_Process function is not available, a warning will be displayed to the user. Therefore, these functions must be implemented for the system to work. More information about the functions listed above can be found in [18, 19].

Still, it is easy to underestimate the effort required to transform an existing offline implementation of a signal processing algorithm into a functional online implementation. While BCI2000 tries to make the transformation as simple as possible, it cannot remove the effort required to deal with chunks of data, which implies the need of buffering - rather than having immediate access to a continuous data set, it may be necessary to maintain an additional data buffer – and also the need of dealing with the MATLAB interface, maintaining a consistent state between subsequent calls to the processing script [20]. This need of buffering is present in the bci_Process script. It sustains this effort by using a scrolling buffer. Once a new block of data is received from the previous module the oldest received block of data is dumped and replaced by the new one.

The simple implemented signal processing method applies the following processes chain to the data obtained in the acquisition module. There is a band-pass filter which isolates the frequencies of the stimuli presented by the SSVEP application. Then, after performing the power spectrum of the filter output, the power at each stimulus frequency is used as output. This operation is executed for each channel, producing a matrix output with dimensions *channels x number of stimuli* which will later adapted to the format required by BCI2000. Figure 4 shows below a graphical description of the signal processing chain for each channel.

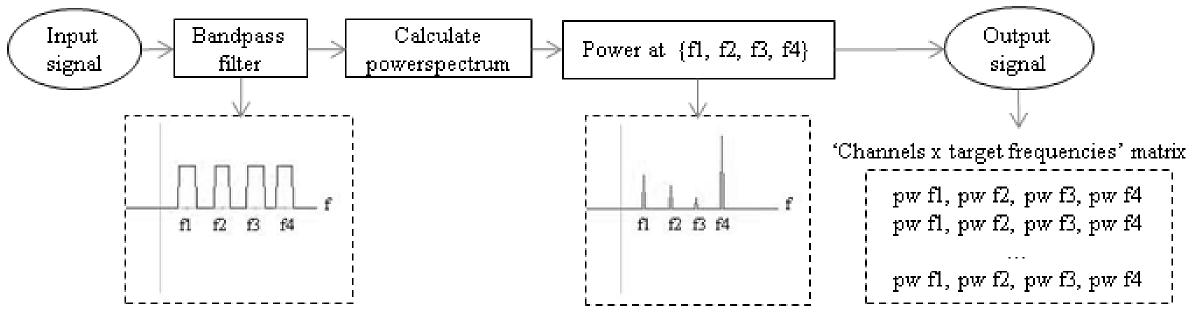

Input signal
Bandpass filter
Calculate powerspectrum
Power at {f1, f2, f3, f4}
Output signal
f1 f2 f3 f4 f
f1 f2 f3 f4 f
'Channels x target frequencies' matrix
pw f1, pw f2, pw f3, pw f4
pw f1, pw f2, pw f3, pw f4
...
pw f1, pw f2, pw f3, pw f4

Figure 4: Signal processing scheme

As the size of the blocks passing between BCI2000 is uniform, the output signal dimensions must be the same as input signal dimensions, which are *channels x blockSize*. The signal outputted by the described signal processing has different dimensions so, before send it to the next module, it is expanded to the right dimensions by adding *0* padding. The block size is freely configurable by the user. The number of channels is typically fixed by the acquisition module. Nevertheless, the user/developer can later choose how many channels to use in the *Config* option provided in the *Operator* module.

Once BCI2000 is running with the MATLAB filter, a MATLAB command line window will open. In that command line window the developer can type commands that will show the variables that BCI2000 communicates to the MATLAB engine [21]. It is important to make sure that BCI2000 connects with the MATLAB engine. It's necessary to verify that the BCI2000 files are placed in the C/ directory and check the PATH variable as mentioned in [22]. Administrator privileges are also necessary.

Detailed information about the developed MATLAB scripts and code can be found in Appendix 1 at page 20.

3.3. Application Module

The application was built in C++ using the Qt Creator IDE [23]. Qt Creator is a cross-platform integrated development environment (IDE). Qt Creator runs on Windows, Linux/X11 and Mac OS X desktop operating systems, and allows developers to create applications for multiple desktop and mobile device platforms. Qt Creator is freely accessible for download, either alone or as part of the Qt SDK. By simply installing the downloaded package from its webpage the development tool will be available.

Qt Creator comes with a GUI designer which provides an easy way to create the graphical user interface needed for the SSVEP application. To start a new project like the present one the developer needs just to click the *Create project* option and selects then *Qt Gui Application*. After that, refer to the *Design* option to begin the GUI design. For code writing, refer to the option *Edit*.

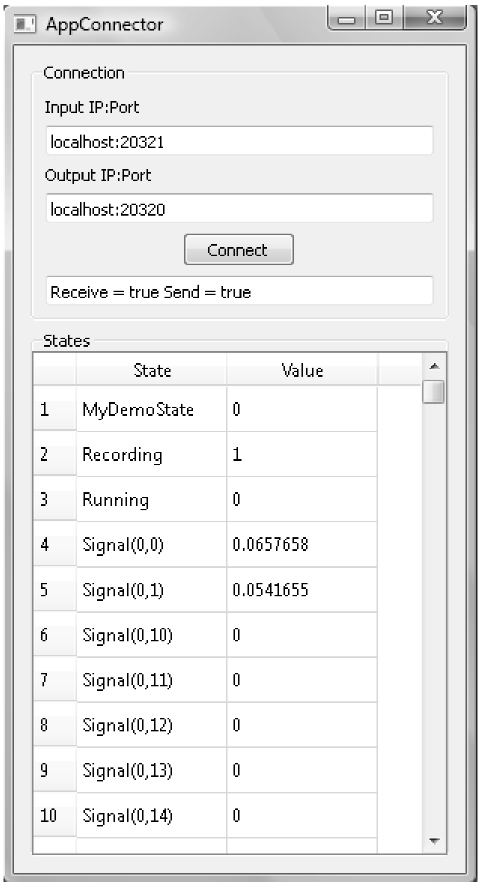
When building an application for SSVEP stimulation, is important to take into account the number of targets or commands. Increasing the number of targets offers a higher number of possible commands but can decrease classification accuracy and speed [9]. The application developed in this project allows the user to select within 16 choices as explained before in the introduction. The main window, which presents the SSVEP stimulation, is shown in Figure 1. The *MainWindow* class constitutes the main graphical interface of the SSVEP application. It presents the four targets in the screen and provides a graphical list of the selected pictures. It also has a start button and four selection buttons in case the selection via BCI2000 fails and shows an upper menu that allows the user the configuration of the application.

When developing the four flicker targets, multithreading programming was needed in order to guarantee the correct flicker. If only timers were used the flicker would be incorrect and faulty, so for each target a thread containing a timer became mandatory. This way, every timer is executed autonomously. For this purpose a class called *TimerThread* was created. *TimerThread* class holds a thread which handles a timer. This way, timers do not interrupt themselves. The timer handles the flicker frequency of a particular picture (or group of pictures).

The number of stimuli is always limited by the refresh rate of the monitor [12], so the 4 frequencies used for the flicker objects were selected taking this into account. As the refresh rate of the monitor used is 60 Hz, the available frequencies become {60, 30, 20, 15, 10, 8.57, 7.5, 6.66, 6, … , 1 Hz}, which correspond to the refreshRate/frame, being frame = {1, 2, 3,…, 60}. In order to avoid close frequencies, the used set was {6, 10, 15, 20 Hz}.

The connection between the application and BCI2000 is established via the external application interface included in BCI2000. The *AppConnector* interface provides a bidirectional link to exchange information with external processes running on the same machine, or on a different machine over a local network [24]. As shown in Figure 3, the designed application uses the *AppConnector*, which connects with BCI2000 using UDP protocol. For each block of data processed by the BCI2000 system, two types of information are sent out and may be received from the external application interface; the BCI2000 internal state and the BCI2000 control signal. The internal states are variables that represent if the system is either running or suspended, the time when a block of data was recorded, etc… The control signal is the output of the signal processing and is presented in the following format: *signal(<channel>,<element>) = float value in decimal ASCII representation*, where <*channel*> is the channel index and <*element*> is the sample. <*channel*> and <*element*> are given in zero-based form. In this case, the relevant values of <*channel*> will be {0, 1, 2, 3}, because of the expected output signal format described in the previous section. *AppConnector* messages format mentioned is completely unrelated to the binary message format BCI2000 uses for communication between its modules.

A developed example is included in the BCI2000 source code; it is called *AppConnectorExample* and it allows catching UDP messages from BCI2000. This example has been developed by the BCI2000 team using QT and comes with all the source code, so its integration with the SSVEP application means just adding those files into the project and then linking them to feed the data post-processing functionality. For this purpose the new class *AppConnector* was added to the project. Figure 5 shows a screenshot of the *AppConnector* example integrated into the developed application. Some messages as defined before are visible.



Input IP:Port
localhost:20321
Output IP:Port
localhost:20320
Receive = true Send = true

	State	Value
1	MyDemoState	0
2	Recording	1
3	Running	0
4	Signal(0,0)	0.0657658
5	Signal(0,1)	0.0541655
6	Signal(0,10)	0
7	Signal(0,11)	0
8	Signal(0,12)	0
9	Signal(0,13)	0
10	Signal(0,14)	0

Figure 5: AppConnector receiving messages

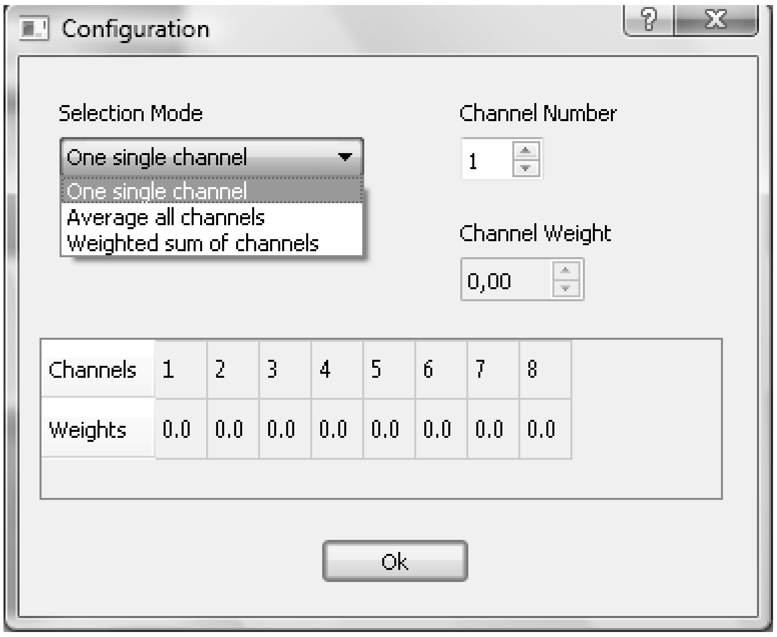
The created application includes a post-processing feature for the incoming data. The application allows getting the classification result by averaging all the received channels, getting the info from one single channel or making a weighted sum of the channels. The processing is done by the *MainWindow* class. The process is explained below in pseudocode.

```
//SignalValues is the array where signal values
//from BCI2000 are stored.
signalValues[numChannels][samples];
//ChannelWeights is where weights for each channel are stored.
channelWeights[channels];
//For singleChannel case, chosenChannel is the selected channel.
chosenChannel;

//Auxiliar variables
aux;
auxArray[samples];

    switch(configurationOption)
    {
    case singleChannel:
        for each relevant sample at the chosen channel
        {
           //Find the high value and keep the iteration index
           if(signalValues[chosenChannel][sample]>aux)
           {
               update aux;
               selection = indexIteration;
```

```
                }
            }
        case averageChannels:
            for each relevant sample
            {
                for each channel
                {
                   //Add all the channel values for each sample
                   auxArray[sample] += signalValues[channel][sample];
                }
            }
            for each sample
            {
               //Find the high value and keep the iteration index
               if(auxArray[sample]>aux)
               {
                   aux = auxArray[sample];
                   selection = indexIteration;
               }
            }
        case weightedSumChannels:
            for each relevant sample
            {
                for each channel
                {
                   //Add all the channel weighted values for each sample
                   auxArray[sample] += signalValues[channel][sample]
                                       *channelWeights[channel];
                }
            }
            for each sample
            {
               //Find the high value and keep the iteration index
               if(auxArray[sample]>aux)
               {
                   aux = auxArray[sample];
                   selection = indexIteration;
               }
            }
        } //End switch
    return selection;
```

Relevant samples mentioned in the code above stand for those samples which are non-zero. Remember the output signal served by the MATLAB signal processing has been zero padded, so only the first four samples, (which correspond to the four targets), contain relevant information.

To choose one of these options, a configuration tool has been created and it is accessible by clicking in the main windows' upper menu. The class added for this feature is called *Configuration* and it handles the selection of one of the three options mentioned before. This class modify the variables *configurationOption, chosenChannel* and *channelWeights* presented in the code before. Then, variables are used by the *MainWindow* class to perform the post-processing algorithm. A screenshot of the configuration tool graphical interface is available in Figure 6.



Selection Mode: One single channel

Channel Number: 1

Channel Weight: 0,00

Channels	1	2	3	4	5	6	7	8
Weights	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

Figure 6: Configuration graphical user interface

After getting the information from the *AppConnector* and performing the post-processing operations on the data, the classification result is used to store a picture in the right margin of the application's main window. The program will wait a short time -about 5 seconds- to receive the data and then it will process it. A classification result is produced after the processing and it is used to select a group of pictures or a specific picture if a group is already expanded. The selected picture will be placed into the reserved spaces in the right margin. If the waiting time expires and the classification result has not been received, the user will have to click one of the four buttons placed nearby the *Action!* button to be able to perform a selection. This button was added to assure the selection option even though the BCI2000 and the application were not able to connect with each other. Figure 1 in the Introduction section shows the mentioned buttons and the selected pictures area.

3.4. EEG data acquisition

The EEG recordings were performed using the Deymed TruScan 32 acquisition hardware. Recordings were made with eight electrodes located on the central, parietal, occipital and temporal lobes, namely in positions Cz, P3, Pz, P4, O1, O2, T5 and T6 according to the international 10–20 system. This electrodes position was carefully chosen nearby parietal and occipital regions because attention increases SSVEP power at electrodes over both occipital and parietal cortex [25]. The data was sampled with sampling rate of 1024 Hz. Figure 7 shows the electrodes configuration.

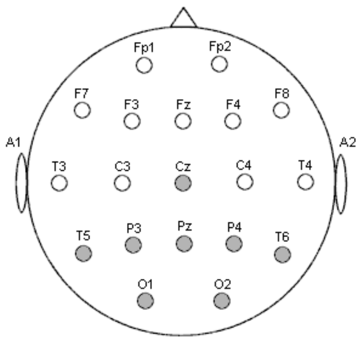


Figure 7: Electrodes configuration

One set of recording was made using frequencies {6, 10, 15, 20 Hz}. The subject was asked to stare at each one of the pictures for making the double selection. For each picture 20 seconds of data were recorded: 10s for the group selection and 10s for the final picture selection.

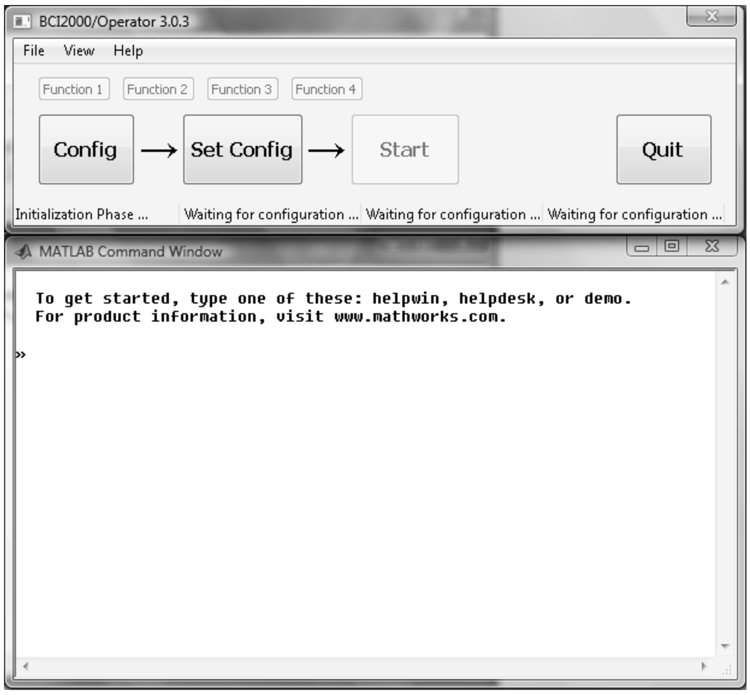
4. Results

There are two main possibilities when testing SSVEP-based BCI systems, offline analysis and online analysis. The most likely and appropriate choice for the developed application was online analysis. In an early stage online analysis was attempted to carry out but the system failed when connecting BCI2000 with the MATLAB engine. After several unsuccessful trials to fix the connection error and having in mind the time constraint, it was decided to switch to offline analysis. Offline analysis was performed by simulating in MATLAB the BCI2000 behaviour. To this purpose, a replacement MATLAB script was written. This script generates dummy signals with some specific strong frequency components. So a dummy signal is basically constituted by a linear combination of different amplitude sine waves. These generated signals pass then through the BCI2000-oriented scripts. Replacement script and MATLAB signal processing scripts are available in Appendix 1 at page 20. As the amplitude of the mentioned frequency components is freely configurable, it provides an easy way to test the signal processing scripts. Results when varying the amplitude are shown in Table 3. Difference of amplitude stated in Table 3 indicates how much higher the tested frequency is, compared with the other targets frequency. For example, when testing frequency 15 Hz with difference of amplitude 15 dB, the amplitude for {6, 10, 15 20 Hz} will be {1, 1, 31.6, 1}.

Table 3: Detection of target frequencies

		Difference of amplitude (dB)		
		10	15	20
Frequency (Hz)	6	not detected	not detected	detected
	10	detected	detected	detected
	15	not detected	detected	detected
	20	not detected	detected	detected

Once the connexion between BCI2000 and MATLAB was set (by taking care about the BCI2000 path as mentioned in Methods section), online analysis was possible. To carry the analysis out, the system was established as follows; First BCI2000 was started using the *BCI2000Launcher* tool provided by the platform, then *Signal Source*, *Signal Processing* and *Application* modules were selected. To run the MATLAB processing, *MatlabSignalProcessing* module has to be selected as *Signal Processing* module. Be aware that the system expects to find the developed scripts at the MATLAB path. To run the developed SSVEP application, *DummyApplication* has to be selected as *Application* module because the application has been created as an external component and it is not a BCI2000 integrated module. The dummy module does nothing, so the only application for the stimulation will be the application started later. As these tests were performed without any EEG acquisition hardware, the *SignalGenerator* module has to be selected as a *Signal Source* module. After launching the modules as mentioned, MATLAB *Command Window* will open and the *Operator* module will show up and will be waiting for configuration. Figure 7 shows a screenshot of the expected scheme.



```
  To get started, type one of these: helpwin, helpdesk, or demo.
  For product information, visit www.mathworks.com.

>>
```

Figure 7: Operator Module waiting for configuration and MATLAB Command Window

Now, once BCI2000 is set up the SSVEP application must be opened. After doing so, the configuration for the tests needs to be done. In order to achieve this, click the *Config* option of the *Operator* module. In the last tab into the *Config* option should appear the customized parameters created in the bci_Construct script. In our case the settable parameters are the set of frequencies and the time per selection. In the *Source* tab the developer can configure the signal generator parameters. To constitute the connexion between BCI2000 and the application, the parameters in the *Connector* tab must be stay as follows;
ConnectorInputAddress: localhost:20320, *ConnectorOutputAddress*: localhost:20321. Also, the AppConnector configuration has to be compatible by setting its parameters as these; *Input IP:Port*: localhost:20321, *Output IP:Port*: localhost:20320. After setting this configuration by clicking the *Set Config* option in the *Operator* module, the tests can be performed. It is important to start the SSVEP's AppConnector module by opening it and clicking the *Connect* button.

In order to check if the SSVEP application was receiving and classifying the results in the right way, the signal generator module was configured to produce signals with frequencies 6, 10 15 and 20 Hz. Then, the application was expected to selects targets 1, 2, 3 and 4 respectively. Named targets 1, 2, 3 and 4 stand for pictures or group of pictures at positions up left, up right, down left and down right. This means that, when the signal generator is producing i.e. a 5 Hz sine wave, the MATLAB signal processing will expectedly process it and it will produce an output signal with higher power for this frequency component than for the others. Then the AppConnector will serve this information to the application which, after processing it, will detect the up left picture or group of pictures. Table 4 shows the results obtained.

Table 4: Application classification results

SineFrequency	SineAmplitude	(White) NoiseAmplitude	Target detected
6	100muV	30muV	1 (up left picture)
10	100muV	30muV	2 (up right picture)
15	100muV	30muV	3 (down left picture)
20	100muV	30muV	4 (down left picture)

5. Discussion

5.1. SSVEP paradigm

The SSVEP offers certain advantages over the transient VEP for the study of sensory and cognitive processes in that its signal is easily recorded and quantified and can be rapidly extracted from background noise. Because of the high rate of stimulus presentation (4–20 times faster than for transient VEPs), it is possible to obtain reliable wave forms more rapidly. Also, SSVEP measurements can reveal how attention is allocated within a complex, multielement stimulus array, because the visual response to each element can be measured individually by examining the SSVEP at its specific flicker frequency [6]. However, the SSVEP can be elicited by an irrelevant background flicker, which could compromise the operation of the application. This possible problem has not been tested in the presented approach because of the inability to perform EEG online analysis.

SSVEP paradigm was selected as the BCI solution because of its effectiveness. SSVEP-based BCI designs achieve high signal-to-noise ratios [2] and require short training. The signal produced by SSVEP is measurable in as large a population as the transient VEP. The task of feature extraction is reduced to simple frequency component extraction, as there are only a certain number of separate target frequencies, usually one for each choice offered in the BCI. On the other hand repetitive visual stimuli modulated at certain frequencies can provoke epileptic seizures and flashes that are excessively bright may impair the user’s vision. Furthermore, certain stimulation frequencies can induce fatigue [9].

5.2. Analysis and results

The online analysis was attempted in an early stage, but the connexion between BCI2000 and MATLAB was unsuccessful despite the efforts carried out by researching widely the possible solutions to the problem. After performing the tests using the offline analysis, a solution to the persistent problem was found just by chance and the online analysis was accomplished. This solution that has been already mentioned before consists on locate the BCI2000 files into the C/ directory and open its tools with administrator privileges.

The results from the offline study indicate that the diverse SSVEP responses can be used to make decisions when difference of amplitude between frequency components is high enough (i.e. more than 15 dB). For the online case, the signal being receiving and processed is just a sine wave at an specific frequency, so theoretically there is no others frequency components except the frequency of the sine wave, which makes easier the classification.

Although it cannot be appreciated in Table 3 or Table 4, higher target frequencies will produce better and easy classifications. This means that difference between the greatest output power and the second greatest output power is higher. When testing lower frequencies this difference become smaller. However, as the presented classification method just looks at the maximum power output, it does not provide any information about how reliable has been the classification.

Tests performed for 20 Hz frequency and 15 dB difference of amplitude during offline analysis shown high power values for 10 Hz frequency. This means that, when detecting a stimulus flickering at the frequency f_1, also the harmonics $2f_1$, $3f_1$… can be detected by the signal processing implemented. Thus, when considering a small recording interval, it is possible to erroneously detect frequency f_1 instead of $2f_1$ or $3f_1$ [7].

When executing the online analysis some interesting performances were observed. Looking at the *Timing* BCI2000’s visualization tool, it can be observed that when the signal processing is running, a peak in the roundtrip shows up every *TimeSel* seconds. This parameter has been created in the bci_Construct MATLAB script and can be consulted in Apenndix 1 at page 20. Figure 8 shows the *Timing* tool when the signal processing module is operating.

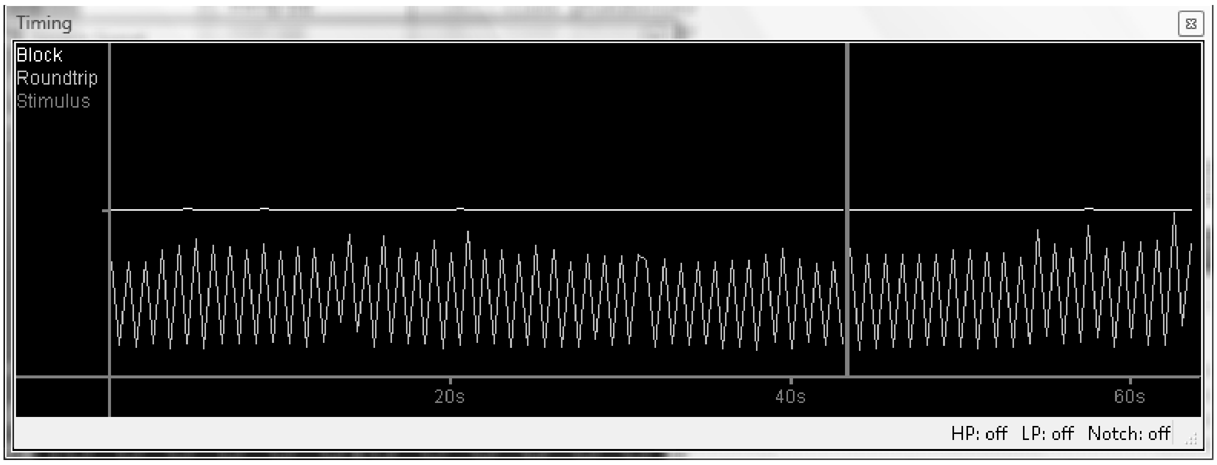


Figure 8: Timing visualization tool when signal processing module is operating. Timing is a critical issue in a system that processes data in real time. In BCI2000, data is acquired and processed in sample blocks and ideally, these are acquired in regular intervals. To work in real-time, the system needs to finish processing, user display, and data storage within a block duration. Roundtrip time is the time needed for a sample block to traverse the core modules. Starting with the acquisition of a sample block, a block's roundtrip includes the time spent on signal processing and stimulus display. The roundtrip finishes when the block enters the data acquisition module again. To fulfill the real-time constraint, roundtrip time may not exceed the physical duration of a sample block. For stable system operation, a weaker condition is sufficient: only the roundtrip's average value needs to stay below a sample block duration [26].

It has been also observed that when the blockSize is set as 64, the Roundtrip is better than for smaller blockSize such as 32. Also, when using a higher sampling rate, higher peaks show up, making Roundtrip and Block closer which could produce a warning or even stop the processing. If 1024 Hz sampling rate and 64 blockSize are used, the system shows a warning because the Roundtrip time exceeds block duration. This entails a problem, because the software used to obtain the EEG data uses this configuration. At this point three mainly options seems the solution; Check and improve the MATLAB code in order to make it faster, switch to a C++ signal processing module or change the EEG acquisition hardware. The most suitable options are those involving software rather than hardware.

5.3. BCI2000 and MATLAB

BCI2000 was chosen as a basis platform because of its multiple benefits such as its support for different data acquisition hardware, signal processing routines, and experimental paradigms. Also, the BCI2000 system does not rely on 3rd-party software components for its operation. Even for compilation, the system only requires affordable or free C++ compilers. Thus, both development and deployment of BCI2000 on multiple computers in potentially multiple sites is very economical. And, as mentioned before, the use of BCI2000 is free for academic and research institutions. Despites all the benefits and the information available on the internet, BCI2000 is not very intuitive and its operation is difficult to understand in the first uses. Extra care and patience must be taken when building its source code.

It was decided to use MATLAB for the signal processing because its multiple advantages; it is interactive, has a simple syntax and no explicit declaration of variables and functions is required. It is a standard for neuroscience data analysis and has many toolboxes available, algorithms implemented and data visualisation tools. However, its big disadvantage for this purpose is that it is slower than compiled code and is not open source. Given the time limitations for accomplish this project and the author’s lack of C++ signal processing basis, the slowness was assumed in order to work with MATLAB syntax rather than C++.

5.4. EEG analysis

Frequency mismatch could happen when using a monitor for SSVEP stimuli presentation, due to the inherent imprecision of the software and the refresh rate of a monitor. In example, it can be observed that the expected frequency 6 Hz became 5.313 Hz by simply taking a look at the power spectrum of the recorded EGG signals. Extra care must be applied when handling this, in order to avoid the isolation of incorrect frequencies by the band-pass filter. Also it can be observed the noise produced by the electrical grid, at 50 Hz and its harmonics.

6. Conclusion

6.1. Overall

The application and signal processing developed could be used as a research tool, which could allow the research to test diverse sets of frequencies and apply different channels configuration to achieve better performances.

The signal processing used does classify the choices as shown in the results. However, extra care is needed with the band-pass filter, because it could detach the desired frequencies. This frequency mismatch appears when using a flicker object in a monitor as SSVEP stimuli.

6.2. Further work

As mentioned in the Discussion section, the classification method used does not provide any information about the results reliability. This information could be calculated in the signal processing and later be added to the output signal served to the following module. As the output signal incoming to the application module has a considerable amount of useless padding, this filling could be replaced by relevant information about the classification consistency and it could be used later by the application.

BCI2000 offline analysis tools could be used for processing the recorded EEG data and replace the EEG online analysis needed to validate the flicker target as an accurate and precise SSVEP system. This online analysis could also be helpful to discover issues such as the noise produced by the electrical grid.

6.3. Evaluation of the work against the original goals

The objective previous to the beginning of the project development was to implement a SSVEP-based BCI, using a device to get the information from the brain and then use that information to communicate the brain with a software application. In that context it could be affirmed that the objective has been partially reached. The project was expected to be a good and fast communication pathway between a subject and a computer. There is still work to do in the acquisition part of the project and the signal processing has to be improved in order to achieve the speed expected. The present research and development could however be used as a basis for some other applications such as video games or computers controlling.

The academic objectives when the development of the project began relied on acquiring the right knowledge about research that could be useful in finishing the degree. The realization of this project has been helpful to the author and it surely will be beneficial in her future career and work related decisions. This research work will provide more perspective in order to get a proper job or a career in research.

Acknowledgements: The author would like to thank the supervisor Dr. S. Nasuto for the guidance. Special thanks also to Matthew Spencer for his help and support during the project development and testing.

7. References

[1] J.R. Wolpaw, N. Birbaumer, D. J. McFarland, G. Pfurtscheller, and T. M. Vaughan,“Brain-computer interfaces for communication and control” *Clinical Neurophysiology,* vol. 113, no.6, pp. 767–791, 2002.

[2]. E. C. Lalor , S. P. Kelly , C. Finucane , R. Burke , R. Smith , R. B. Reilly , G. McDarby, “Steady-state VEP-based brain-computer interface control in an immersive 3D gaming environment”, *EURASIP Journal on Applied Signal Processing,* v.2005 n.1, pp. 3156-3164, 1 January 2005.

[3] J. Mellinger , G. Schalk , C. Braun , H. Preissl , W. Rosenstiel , N. Birbaumer and A. Kuebler "An MEG-based brain-computer interface (BCI)", *NeuroImage,* vol. 36, pp.581 - 593 2007.

[4] S.-S. Yoo, T. Fairneny, N.-K. Chen, et al., "Brain-computer interface using fMRI: spatial navigation by thoughts," *NeuroReport* , vol. 15, no. 10, pp. 1591-1595, 2004.

[5] Reza Fazel-Rezai and Waqas Ahmad (2011). P300-based Brain-Computer Interface Paradigm Design, Recent Advances in Brain-Computer Interface Systems, Prof. Reza Fazel (Ed.), ISBN: 978-953-307-175-6, InTech, Available from: http://www.intechopen.com/books/recent-advances-in-brain-computer-interface-systems/p300-based-brain-computer-interface-paradigm-design

[6] Zani A., Mado Proverbio A. (2003). The Cognitive Electrophysiology of Mind and Brain. Academic Press.

[7] H. Segers, A. Combaz, N.V. Manyakov, N. Chumerin, K. Vanderperren, S. Van Huffel, and M.M. Van Hulle, (2011) "Steady State Visual Evoked Potential (SSVEP) -based Brain Spelling System with Synchronous and Asynchronous Typing Modes". In IFMBE Proceedings, vol. 34 15. *Nordic-Baltic Conference on Biomedical Engineering and Medical Physics (NBC15).* Aalborg, Denmark, June 14-17, 2011, pp. 164-167.

[8] Jia C, Gao X, Hong B, Gao S (2011) "Frequency and phase mixed coding in SSVEP-based brain–computer interface". *IEEE Trans Biomed* Eng. 58: pp. 200–206.

[9] Zhu D, Bieger J, Molina G G and Aarts R M 2010 "A survey of stimulation methods used in SSVEP-based BCIs". *Comput. Intell. Neurosci.* 1, 2010, p. 702357.

[10] Schalk et al., IEEE Trans Biomed Eng, 2004 Mellinger and Schalk, In: Brain-Computer Interfaces. MIT Press, 2007 [http://www.bci2000.org]

[11] Y. Renard, F. Lotte, G. Gibert, M. Congedo, E. Maby, V. Delannoy, O. Bertrand, A. Lécuyer, "OpenViBE: An Open-Source Software Platform to Design, Test and Use Brain-Computer Interfaces in Real and Virtual Environments", Presence : teleoperators and virtual environments, vol. 19, no 1, 2010.

[12] Wang, Y.; Wang, Y.T.; Jung, T.P. "Visual stimulus design for high-rate SSVEP BCI". Electron. Lett. 2010, 46, pp.1057-1058.

[13] S. Parini, L. Maggi, A. C. Turconi, and G. Andreoni, "A robust and self-paced BCI system based on a four class SSVEP paradigm: algorithms and protocols for a high-transfer-rate direct brain communication," *Computational Intelligence and Neuroscience*, vol. 2009, Article ID 864564, 11 pages, 2009.

[14] G. Bin, X. Gao, Z. Yan, B. Hong, and S. Gao, "An online multi-channel SSVEP-based brain-computer interface using a canonical correlation analysis method," *Journal of Neural Engineering*, vol. 6, no. 4, Article ID 046002, 6 pages, 2009.

[15] P. Martinez, H. Bakardjian, and A. Cichocki, "Fully Online, Multi-Command Brain Computer Interface with Visual Neurofeedback Using SSVEP Paradigm," J. *Computational Intelligence and Neuroscience.*

[16] J. J. Vidal, "Real-time detection of brain events in EEG," *Proc. IEEE*, vol. 65, no.5, pp. 633-641, 1977.

[17] BCI2000 Wiki: Programming Reference: Build System. Available at: http://www.bci2000.org/wiki/index.php/Programming_Howto:Building_BCI2000

[18] Schalk G., Mellinger J. (2010). Writing a Custom Matlab Filter. In: A Practical Guide to Brain–Computer Interfacing with BCI2000, pp. 114-119 Springer.

[19] BCI2000 Wiki: Programming Reference: MatlabFilter. Available at: http://www.bci2000.org/wiki/index.php/Programming_Reference:MatlabFilter

[20] BCI2000 Wiki: Programming Tutorial: Implementing a Matlab-based Filter. Available at: http://www.bci2000.org/wiki/index.php/Programming_Tutorial:Implementing_a_Matlab-based_Filter

[21] Schalk G., Mellinger J. (2010). MatlabFilter. In: A Practical Guide to Brain–Computer Interfacing with BCI2000, pp. 179-180 Springer.

[22] BCI2000 Wiki: Troubleshooting at Programming Reference: MatlabFilter. Available at: http://www.bci2000.org/wiki/index.php/Programming_Reference:MatlabFilter#Troubleshooting

[23] QT Creator IDE and tools. Available at: http://qt.nokia.com/products/developer-tools/

[24] Schalk G., Mellinger J. (2010). AppConnector. In: A Practical Guide to Brain–Computer Interfacing with BCI2000, pp. 92-96 Springer.

[25] Ding J, Sperling G, Srinivasan R (2006) "Attentional modulation of SSVEP power depends on the network tagged by the flicker frequency". Cereb Cortex 16: pp. 1016–1029.

[26] BCI2000 Wiki: User Reference: Timing. Available at: http://www.bci2000.org/wiki/index.php/User_Reference:Timing

All web addresses referred to in this report were verified on 22 April 2012.

APPENDIX 1: Code and extra information about the MATLAB scripts.

The list of the scripts being implemented is the following: bci_Construct, bci_Preflight, bci_Initialize and bci_Process. The execution order takes place as mentioned. It can be observed that besides the mandatory scripts presented in *Methods: Signal processing module* section, the bci_Construct script has also been implemented.

1. bci_Construct

The MATLAB filter does not have a set of default parameters that it uses. Instead, the user-supplied MATLAB functions that are executed by the MATLAB filter specify the parameters. After initialization, these parameters are displayed in the Operator module and can be modified there [21]. States information needs no extra care by the developer.

```
function [ parameters, states ] = bci_Construct
% bci_Construct Perform any initialization;
%   request BCI2000 parameters and states
%   by returning parameter and state definition lines as demonstrated
%   below.
%
%   María Madrid - March 2012

global targets
global blocksIn

parameters = {
    %Freqs and TimeSel parameters are set to be a user configurable param
    %See format in Parameter Format at BCI2000's Project Outline pp. 5-8
   'MatlabTry intlist Freqs= 4 5 12 17 23 0 0 0 // Bandpass frequencies',
   'MatlabTry int TimeSel= 1 1 0 5// Time per selection (seconds)'
};
states = { ...
    %No need to set any state
};

%Set initial values for targets and blocksIn
targets = 4; %Fixed value, the application works with 4 targets
blocksIn = 0;
```

2. bci_Preflight

This script was supposed to check whether parameters and states are accessible, and whether parameters have values that allow for safe processing by the bci_Process function. In order to avoid over complexity and improve the execution time, it was decided to use this mandatory script just to report the output signal dimension.

```
function [ out_signal_dim ] = bci_Preflight( in_signal_dim )
% bci_Preflight Report output signal dimensions
%   in the 'out_signal_dim' argument.
%   No verifications are performed.
%
%   María Madrid - March 2012

out_signal_dim = in_signal_dim;
```

3. bci_Initialize

```
function bci_Initialize( in_signal_dims, out_signal_dims )
% bci_Initialize    Perform configuration for the bci_Process script.
%   Get the appropriate variables from BCI2000 and calculate some others
%   needed in the bci_Process script.
%
%   María Madrid - March 2012

%global variables
global bci_Parameters bci_States;
global Fs bp numBlocks freqs numChannels blockSize buffer;

%Gets the values of the global variables from bci_Parameters
%Vector containing the target frequencies
freqs = str2num(char(bci_Parameters.Freqs));
%Sampling Frequency
Fs = str2num(strrep(lower(char(bci_Parameters.SamplingRate)),'hz',''));
%Time per selection
timeSel = str2num(char(bci_Parameters.TimeSel));
%Size of each input block
blockSize = str2num(char(bci_Parameters.SampleBlockSize));
%Number of used channels. Selected by the user in the config options.
numChannels = length(bci_Parameters.TransmitChList);

%Filter parameters
Att = 80;   %Attenuation
vpass = 1;  %half width of the pass band
vstp = 2;   %width between the pass band and the stop band
Apass  = 1;           % Pass band Ripple (dB)
match  = 'passband';  % Band to match exactly

%Number of blocks in the buffer
numBlocks = timeSel*Fs/blockSize;

%casting to freqs
freqs = double(freqs);

%Work out the filters for each frequency
h1  = fdesign.bandpass(freqs(1)-vstp, freqs(1)-vpass, freqs(1)+vpass,
freqs(1)+vstp, ...
    Att, Apass, Att, Fs);
bp1 = design(h1, 'butter', 'MatchExactly', match);
h2  = fdesign.bandpass(freqs(2)-vstp, freqs(2)-vpass, freqs(2)+vpass,
freqs(2)+vstp, ...
    Att, Apass, Att, Fs);
bp2 = design(h2, 'butter', 'MatchExactly', match);
```

```
h3  = fdesign.bandpass(freqs(3)-vstp, freqs(3)-vpass, freqs(3)+vpass,
freqs(3)+vstp, ...
    Att, Apass, Att, Fs);
bp3 = design(h3, 'butter', 'MatchExactly', match);
h4  = fdesign.bandpass(freqs(4)-vstp, freqs(4)-vpass, freqs(4)+vpass,
freqs(4)+vstp, ...
    Att, Apass, Att, Fs);
bp4 = design(h4, 'butter', 'MatchExactly', match);

%Put the filters together
bp = dfilt.parallel(bp1, bp2, bp3, bp4);

%Allocate buffer for bci_Process script
buffer = zeros(numChannels,numBlocks*blockSize);
```

4. bci_Process

As mentioned in 'Methods: Signal processing module' section, the use of a buffer became an obligation because of the effort required to transform an offline signal processing algorithm into a functional online implementation.

```
function out_signal = bci_Process( in_signal )
% bci_Process   Apply a filter to in_signal, and return the result in
%   out_signal. Signal dimensions are ( channels x samples ).
%
%   Filter is applied to a buffered signal
%   When the buffer is first filled, signal processing is applied,
%   Once is filled, the oldest block is replaced with a new one,
%   the signal processing is applied again to the buffer and so on.
%
%   MarMadrid - March 2012

out_signal = zeros(size(in_signal));

%global variables
global bci_Parameters bci_States;
global Fs bp targets buffer blocksIn numBlocks freqs blockSize numChannels;
% j;
global out_signal;

%addHead
start = blocksIn*blockSize + 1;
stop = start+blockSize-1;
buffer(:,start:stop) = in_signal;
blocksIn = blocksIn + 1;

if (blocksIn==numBlocks)

    samples = size(buffer, 2); %samples = blockSize*numBlocks
    str = zeros(numChannels, targets); %signal strengths for each target
frequency

    %Frequency axis
    axis = double(Fs*linspace(0, 1, samples));

    %Find the frequency indexes
    freqIndex = [find((axis-freqs(1))==min(abs(axis-freqs(1)))) ...
```

```
                find((axis-freqs(2))==min(abs(axis-freqs(2)))) ...
                find((axis-freqs(3))==min(abs(axis-freqs(3)))) ...
                find((axis-freqs(4))==min(abs(axis-freqs(4))))]';

    %Loop for channels
    for i=1:numChannels
            x = buffer(i, :);
            %Apply the filter
            y = filter(bp, x);
            %Calculate powerspectrum
            p = powerspectrum(y, Fs);
            pw = p(2,:); %power, (see powerspectrum.m)
            %Find the power for each freq
            str(i,:) = pw(freqIndex);
            %Fill the output signal
            out_signal(i, 1:targets) = str(i,:);
    end

    blocksIn = 0;

end
```

5. powerspectrum

The powerspectrum function used in the bci_Process is attached below.

```
function p=powerspectrum( signal, Fs )
% POWERSPECTRUM     Calculate the power spectrum of a signal.
%   p = powerspectrum(signal, Fs) takes the FFT of the data series and
%   returns meaningful power spectrum formated data.  SIGNAL is a 1-by-N
%   vector representing the signal samples and Fs is a scalar that holds
%   the sampling frequency.  Returns the 2-by-N matrix p that contains the
%   frequency values of the spectrum in the first row of p and the power
%   values at each of these frequencies in the second row of p.
%
%   Ian Daly - Pre-June 2011
%   Matthew Spencer - June 2011

    % Transpose inputs if the data is horizontal instead of vertical
    [m,n] = size(signal);
    if m<n
        signal=signal';
    end

    T = 1/Fs;
    L = size( signal , 1 );
    t = (0:L-1)*T;
    NFFT = 2^nextpow2(L);
    Y = fft(signal,NFFT)/L;
    f = Fs/2*linspace(0,1,NFFT/2);
    power = 2*abs(Y(1:NFFT/2));
    p(1,:) = f;
    p(2,:) = power;
end
```

6. bci2000harness

The script was developed by Mathew Spencer.

```
% Simulate a BCI2000 Matlab module

clearvars

sig_dim = [22 64];
Fs = 1024;

%% Construct
[params, states] = bci_Construct;

nparams = length(params);
nstates = length(states);

global bci_Parameters
global bci_States

% Loop through all parameters and parse them
for p=1:nparams
    % Parse [string] [string] [string without =] [string without\n]
    tokens = textscan(params{p},'%s %s %[^=]%[^\n]');

    % Second token is the type
    ptype = lower(tokens{2}{1});

    % Third token is the name
    pname = tokens{3}{1};

    % Fourth token contains the value and the comment
    valstr = tokens{4}{1};

    switch ptype
        case 'int'
            disp('Parsing Int32...')
            % Parse [string with =][int][int][int][int][string]
            vals = textscan(valstr,'%[=]%d%d%d%d%s');
            % The first token is '=', the second is the value
            pval = vals{2};
        case 'matrix'
            disp('Parsing Matrix...')
            % Parse [string with =][string without /][string]
            matvals = textscan(valstr,'%[=]%[^/]%s');
            % The first is '=', the second is a string of numbers
            % Parse the second into a vector of numbers
            vals = textscan(matvals{2}{1},'%d');
            pval = vals{1};
        otherwise
            disp('Unknown variable type.')
            pval = nan;
    end

    % Store the name and value into the bci_Parameters struct
    bci_Parameters.(pname) = pval;
end

% Parse all states
```

```
for s=1:nstates
    tokens = textscan(states{s},'%[^ ] %d%d%d%d');
    sname = tokens{1}{1};

    bci_States.(sname) = tokens{2};
end

%% Preflight
out_sig_dim = bci_Preflight(sig_dim);

%% Initialize
bci_Initialize(sig_dim, out_sig_dim);

%% Startrun

% TODO later when we actually want to test a startrun method

%% Process

% Initialize parameters for generating a dummy signal with some specific
% strong frequency components

% Frequency components (in Hz)
f = 0:0.2:Fs/2;

% Phase offsets for each frequency component (in radians)
p = normrnd(0,1,1,length(f)).*pi;
P = repmat(p,sig_dim(2),1);

% Sample times (in seconds)
t=0:1/Fs:(sig_dim(2)-1)/Fs;

% Frequency and time grids for faster calculation
[F,T]=meshgrid(f,t);

% Amplitudes for each frequency component
% All are random and positive (with more power around 0)
a = 40*exp(-f/0.3) + abs(normrnd(0.5,0.1,1,length(f)));

% Some are explicitly set, YOU CAN SET WHATEVER YOU LIKE HERE
a(f==6)=20;    % ie. Set an amplitude of 20 for frequency component 10
a(f==10)=20;
a(f==15)=20;
a(f==20)=30;

% Amplitude grid for faster calculation with frequency and time
A = repmat(a,sig_dim(2),1);

% Number of cycles to run for
ncycles = 50;

% Initialise an example buffer
exbuffer = zeros(1,sig_dim(2)*ncycles);

% Run the process loop
for i=0:ncycles-1

    % Generate Signal Block
    S = A.*cos(F.*(T+(i*sig_dim(2)/Fs)).*2*pi + P);
    signal = sum(S,2);
    % Store the current signal block into the buffer
    exbuffer(sig_dim(2)*i+1:sig_dim(2)*(i+1)) = signal;


    % Pass the signal block into the process method
    %out_signal = bci_Process(repmat(signal,1,sig_dim(1))');

end

%% Plot buffer
figure
subplot(3,1,1)
plot(f,a)
title('Synthetic Frequency Components')
subplot(3,1,2)
plot((0:length(exbuffer)-1)./Fs,exbuffer)
title('Synthetic Signal')
subplot(3,1,3)
p=powerspectrum(exbuffer,Fs);
plot(p(1,:),p(2,:))
title('Recovered Frequency Components')
```