Proceedings of the
7th Graz Brain-Computer Interface Conference 2017 DOI: 10.3217/978-3-85125-533-1-33

P300 SPELLER IMPLEMENTATION USING WEB DEVELOPMENT LANGUAGES

R.E.S. Harnarinesingh[1] and C.S. Syan[2]

[1] Department of Electrical and Computer Engineering, The University of the West Indies, St. Augustine, Trinidad

[2] Department of Mechanical and Manufacturing Engineering, The University of the West Indies, St. Augustine, Trinidad

E-mail: randy.harnarinesingh@sta.uwi.edu

ABSTRACT: Brain-computer Interface (BCI) applications present significant assistive potential for disabled individuals. BCI software is typically implemented on desktop and laptop computers. Mobile platforms such as smartphones and tablets however possess comparable processing power to desktops and laptops. Recent studies have investigated BCI implementation using mobile phones [1, 2]. However, the programs developed in these studies are intended for specific platforms and recoding is required to implemented the programs on other mobile devices.
This paper investigates the implementation of the P300 Speller Paradigm using web development languages. This provides an avenue for universal implementation of the paradigm without the need for recoding for specific platforms. The developed paradigm was tested to ensure that the temporal properties of the paradigm complied with the required timed delays. The testing showed a maximum mean error of 5.17ms and standard deviation of 11.23ms for 100ms temporal segments. This work demonstrates that web languages are a promising avenue for the implementation of BCI paradigms. However, modalities for increasing timing compliance will be explored. Further work will also investigate incorporating data collection and signal processing into the developed program to implementing training and testing sessions for full BCI implementation.

INTRODUCTION

Brain-Computer Interfaces (BCIs) allow user control of external devices using inherent brain activity. BCI paradigms require users to perform tasks which represent commands for external devices. For example, some paradigms can require users to gaze at a left arrow to issue motive commands to a driven wheelchair [3]. BCIs are responsible for recording the raw brain activity and processing the signals to interpret the subject command. This involves multiple hardware and software stages and the anatomy of the standard BCI is presented in Fig. 1.

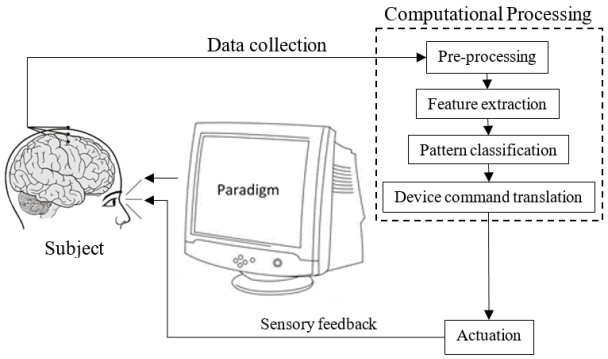

Figure 1: Anatomy of the standard BCI.

BCIs can be implemented using a host of paradigms. These paradigms include the P300 Speller [4], Steady State Visually Evoked Potentials (SSVEP) [5], Slow Cortical Potentials (SCP) [6] and Motor Imagery (MI) [7]. BCIs have been employed for a host of assistive applications such as wheelchair navigation [8] and keyboard control [9].

BCIs are typically implemented on desktop and laptop computers. Mobile platforms such as smartphones and tablets however possess comparable processing power to desktop and laptop machines whilst being smaller and more affordable. Recent studies have investigated BCI implementation using cell phones [1, 2].

These studies however implemented programs on select phone models using device specific software libraries and operating systems. This presents a hurdle to implementing the developed programs on other mobile platforms since code would have to be rewritten to be compatible on other devices.

This paper investigates the feasibility of implementing the P300 Speller Paradigm using web development languages. The expression of the P300 Speller in web development languages allows for implementation on all devices with web browsers regardless of operating systems. These devices include desktops, laptops, tablets and smartphones. This work however treats with the presentation aspects only.

 DOI: 10.3217/978-3-85125-533-1-33

MATERIALS AND METHODS

The development, testing and analysis was performed on a Dell Inspiron17R laptop with an Intel® Core™ i7-4500U CPU clocked at 1.8GHz with 8.00GB of Random Access Memory (RAM). The browser used for laptop testing was Google Chrome Version 55.0.2883.87m. The mobile phone used for testing was the Digicel DL1 manufactured by TCL.

P300 SPELLER DEVELOPMENT

This section provides background on the P300 Speller Paradigm as well as details of the implementation of the P300 Speller using web development languages.

P300 SPELLER PARADIGM

The seminal work on P300-based BCIs was done by Farwell and Donchin [10]. Various modifications to the P300 Speller have since been implemented such as alterations to the paradigm character set and stimulus delivery patterns.

The P300 Speller Paradigm itself is an oddball paradigm in which rare target stimuli are presented amongst frequently delivered non-target stimuli [11]. The Speller Paradigm requires subjects to view a single matrix element during the randomly ordered flashing of all matrix elements [12]. When a matrix element the subject is focusing on is flashed, the P300 response is evoked and the detection of the P300 is used to identify user focus [13]. The P300 Speller has been used for a host of practical BCI applications [8].

WEB DEVELOPMENT LANGUAGES

The sequential operation of the P300 Speller is embodied in the flow chart of Fig. 2. This flow chart represents the specific case of 100ms flash time and Inter-Stimulus Interval (ISI).

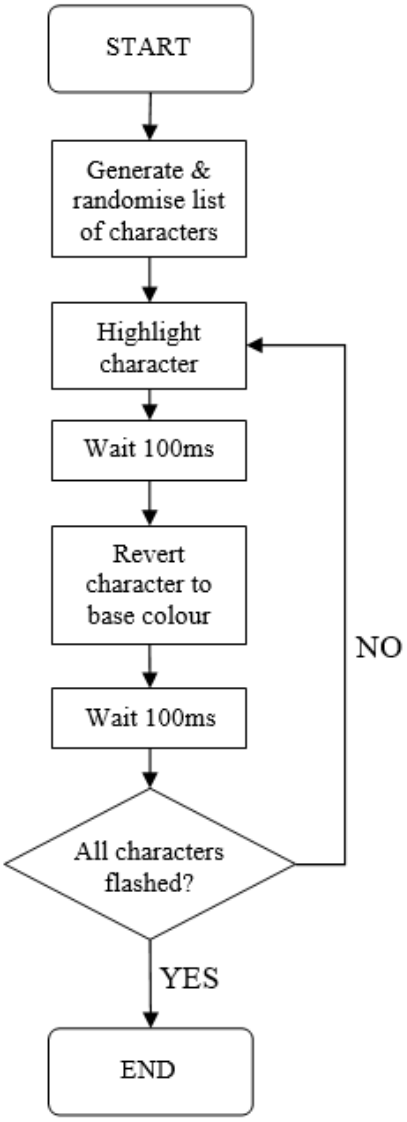

Figure 2: P300 Speller Paradigm Flow Chart

There are 3 main client-side programming languages that are used to implement webpages. They are Hyper Text Markup Language (HTML) [14], Cascading Style Sheets (CSS) [15] and JavaScript (JS) [16]. There are also server-side programming languages that are utilized in live websites. However, the P300 Speller developed in this work is intended to be used offline without the need for an active internet connection. This work therefore does not involve server-side programming languages.

The client-side languages each perform different functions as it relates to webpages. Some of their functions are pictured in Fig. 3.

 DOI: 10.3217/978-3-85125-533-1-33

CLIENT SIDE LANGUAGES

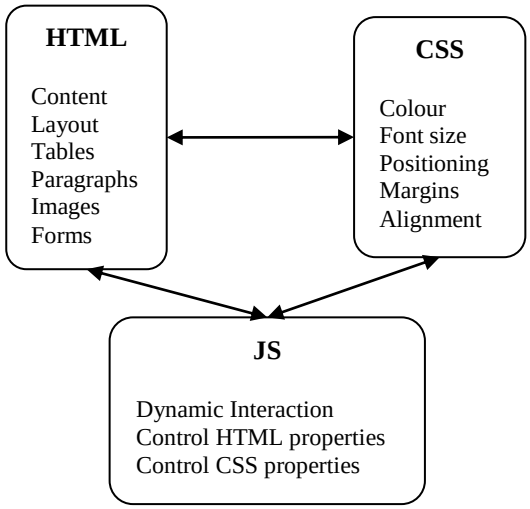

Figure 3: Functions of client side programming languages

There are some elements of the P300 Speller that do not vary with time. These time-static elements such as background colour and button positions can be realized using HTML and CSS. However, it can be seen from Fig. 2. that the P300 Speller also has time-dynamic elements such as element colour changes and timing delays. The only client-side language capable of expressing time-dynamic behaviour is JS. This work also uses JQuery [17] which is a JS Library capable of shorthand coding conventions.

PROGRAM IMPLEMENTATION

This section presents some code snippets and technical information regarding the developed program. The P300 Speller was implemented using HTML, CSS and JS.

The matrix of P300 Speller elements was realised as a table in HTML as shown in Fig. 4. The HTML code for the first row of Speller element is presented below. The remaining 5 rows denoted by the vertical ellipsis are identical to tow 1 besides the text and HTML ids.

```
<table>
    <tr>
        <td id="A">A</td>
        <td id="B">B</td>
        <td id="C">C</td>
        <td id="D">D</td>
        <td id="E">E</td>
        <td id="F">F</td>
    </tr>
                    ⋮
</table>
```

Figure 4: P300 Speller Paradigm HTML code snippet

Each element of the P300 Speller matrix was given a unique id label in HTML. This was done to enable easy targeting by the CSS for character flashing. The CSS code snippet that defines the margins and background-foreground colour for the P300 elements is presented in Fig. 5.

```
table {
    color: white;
    background-color: black !important;
    margin-left: auto;
    margin-right: auto;
}

td {
    width:120px;
    height:120px;
    font-size:6em;
    text-align: center;
    font-family: Arial;
}
```

Figure 5: P300 Speller Paradigm CSS code snippet

The JS code snippet that was responsible for the time dynamic aspects of the P300 Speller Paradigm is showed in Fig. 6.

```
function flash() {
    i=0;
    if(i<c) {
        var flash_index = new_chars[i];
        light_unlit(flash_index,1);

        setTimeout(
            function() {
                    light_unlit(flash_index,0);
                    setTimeout(flash,ISI);
            }
        ,flash_time);
    }
    i++;
}
```

Figure 6: P300 Speller Paradigm JS code snippet

The ‘new_chars’ variable with length ‘c’ is defined outside of the function and contains the total list of characters to be flashed based on the number of trials. The “light_unlit” function flashes a P300 matrix element if the second function input is ‘1’ and reverts the element base colour to white if the second input is ‘0’.

The JS language does not contain a standard delay function that halts code execution for a predefined period. The paradigm timing is therefore implemented using the “setTimeout” JS function. The “setTimeout” function waits for a certain time before running some specified code. Recursion is then employed to ensure

 DOI: 10.3217/978-3-85125-533-1-33

that the code progresses to highlight and then revert character colours until the total character set is flashed.

The developed P300 Speller as viewed on a laptop is shown in Fig. 7.

Figure 7: P300 Speller on Laptop Chrome Browser

A mobile version of P300 Speller was also developed and is viewed in Fig. 8. This is a modified P300 Speller with a smaller command matrix intended for better viewing on a mobile platform. The commands are tailored for the control of a vehicular platform which is a common P300-based BCI application.

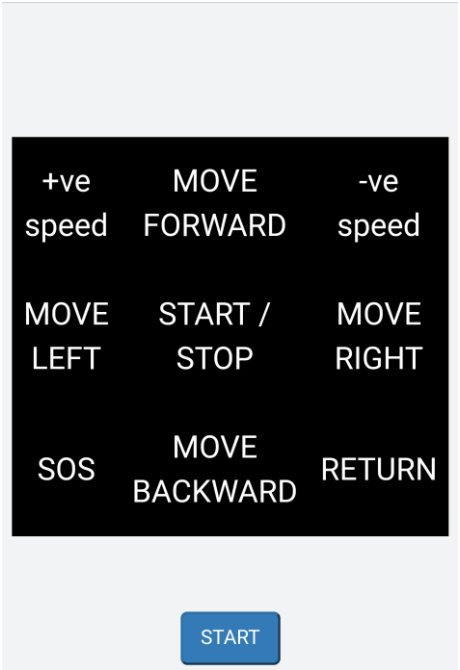

Figure 8: P300 Speller on Android Chrome Browser

RESULTS

The developed paradigm was executed on the Google Chrome Browser for Windows. The paradigm also successfully ran on Internet Explorer and Mozilla Firefox however Chrome was preferred due to previous author experience with the browser. A scaled down P300 Speller was also developed for Chrome on Android to highlight the feasibility of the program for mobile platforms.

The developed program accurately implemented the stimulus sequencing required of the P300 Speller. However, stimulus flash times and ISIs were obtained to determine compliance with the stimulus timing requirements. The PC monitor was therefore recorded using a software screen recorder which ran concurrently with the paradigm. The recording was done at 60 frames per second (fps) which coincides with the screen refresh rate. The recorded videos were then imported into MATLAB and decomposed into individual frames which were analysed to determine the stimulus flash times and ISI.

Fig. 9 and Fig. 10 presents examples of the image processing that was done to determine where a stimulus was flashing or base colour. The technique used utilised the non-flashing case as a subtractive reference image. The result image for subtraction from the non-flashing case is therefore a blank image output. The result for subtraction from the screen capture of the flashing case is a non-zero image. The latter result was therefore discriminated against the former using a basic RGB intensity count feature.

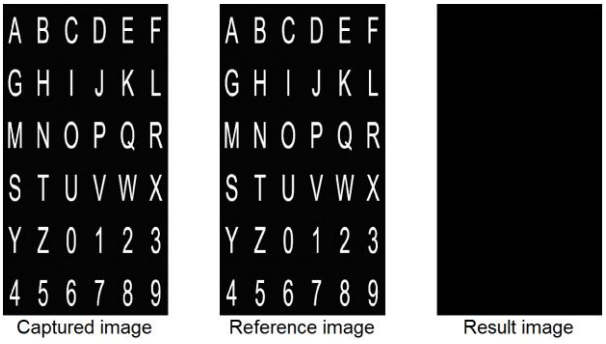

Figure 9: Image subtraction for ISI period

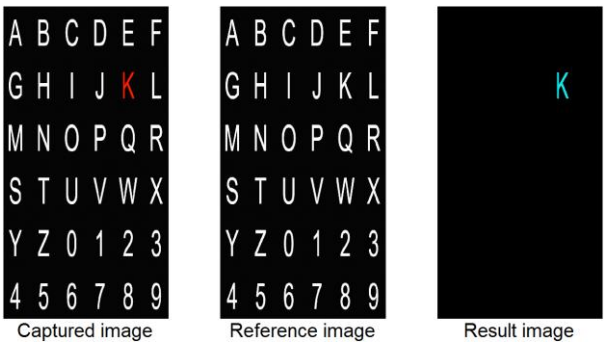

Figure 10: Image subtraction for stimulus highlight

 DOI: 10.3217/978-3-85125-533-1-33

The P300 Speller was run for 5 trials which entailed 180 individual character flashes. The frame analysis therefore testing two main timed elements: (1) 180 flash times and (2) 179 ISIs. The flash times and ISIs were collected for each session. This was repeated for 30 experiments in total and the average times were obtained.

Fig. 11 presents the histogram of stimulus highlight times in terms of number of frames for a single session. Fig. 12 shows a histogram of ISIs for a representative session. It is worth noting that 6 frames are equivalent to the programmed time of 100ms.

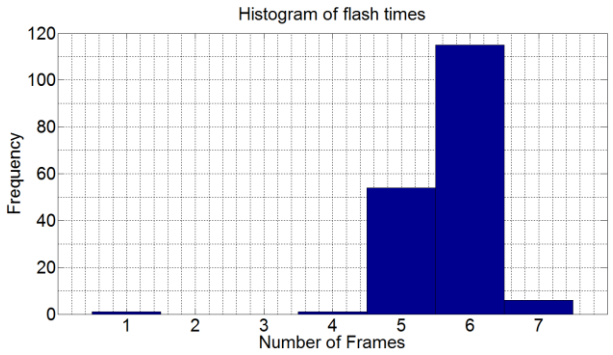

Figure 11: Histogram of stimulus flash times

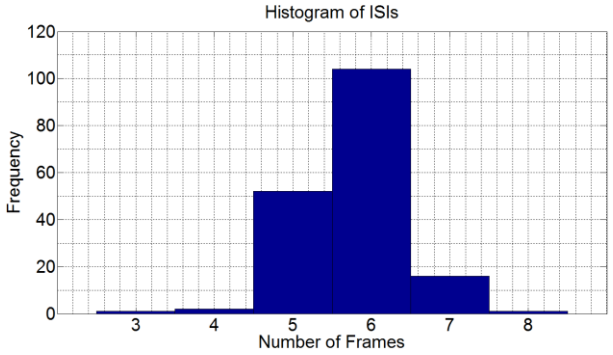

Figure 12: Histogram of ISIs

The global mean and standard deviation of stimulus flash times and ISIs across all testing sessions are presented in Table 2.

Table 2: Programmed and actual timings for P300 Speller

Feature	Programmed Timing	Mean observed time	Error	
			Mean	SD
Flash Time	100ms	94.83ms	5.17ms	10.65ms
ISI	100ms	96.17ms	3.83ms	11.23ms

DISCUSSION

This work investigated the implementation of the P300 Speller using web development languages. The developed paradigm was executed on both laptop and Android variants of the Google Chrome browser. The paradigm was captured and analysed to determine the compliance of the programmed timing delays. This was useful in evaluating time skew due to computational overheads and determining if time skew was significant. The results revealed that the actual flash times and ISIs deviated at most 5.17ms from the programmed time. These timing errors are expected in any time-based BCI paradigm. However, their magnitudes in this work were not detrimental to the success of the paradigm.

The successful implementation of the P300 Speller using web development languages provides a useful proof of concept for BCI implementation in a web browser. In addition, the advent of Android compatible EEG headsets [18] provides a pathway for full implementation of a BCI using mobile platforms such as smartphones and tablets.

There are also benefits to coding the P300 Speller in web development languages such as HTML/CSS/JS instead of device standard languages such as C++ and Java. Web pages can be executed on all modern devices with a web browser. This averts the requirement for any special library or virtual runtime and promises universal execution. This allows the benefit of a single program which can run on every operating system that implements a web browser without considerations for library and hardware constraints.

CONCLUSION & FUTURE WORK

This paper investigated the implementation of the P300 Speller Paradigm using web development languages. The results demonstrate that the developed paradigm complied with the timing delays required of the P300 Speller. The paradigm was executed successfully on both a laptop and Android smartphone. This paper therefore provided an important proof of concept for web browser based BCI paradigm presentation.

However, there are possible avenues of future work. The developed paradigm must be integrated to data collecting and signal processing elements to allow for a full BCI implementation that included training and testing sessions. Virtual PC COM ports provide an avenue for this integration. This would allow for the communication of data between JS and data collection programs in MATLAB for example. In addition, the user interface can be improved to allow for selectable parameters. This can be further expanded to allow for a testbed based approach.

 DOI: 10.3217/978-3-85125-533-1-33

REFERENCES

[1] Wang Y-T, Wang Y, Jung T-P. A cell-phone-based brain–computer interface for communication in daily life. Journal of neural engineering. 2011;8(2):025018

[2] Campbell A, Choudhury T, Hu S, Lu H, Mukerjee MK, Rabbi M, Raizada RD. NeuroPhone: brain-mobile phone interface using a wireless EEG headset, in Proc. Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds, 2010, 3-8

[3] Pires G, Castelo-Branco M, Nunes U. Visual P300-based BCI to steer a wheelchair: A Bayesian approach, in Proc. 30th Annual International Conference of the IEEE Engineering in Medicine and Biology Society, 658-661

[4] Krusienski DJ, Sellers EW, Mcfarland DJ, Vaughan TM, Wolpaw JR. Toward enhanced P300 speller performance. Journal of Neuroscience Methods. 2008;167(1):15-21

[5] Zhu D, Bieger J, Molina GG, Aarts RM. A survey of stimulation methods used in SSVEP-based BCIs. Computational intelligence and neuroscience. 2010;2010(1):1-13

[6] Hinterberger T, Weiskopf N, Veit R, Wilhelm B, Betta E, Birbaumer N. An EEG-driven brain-computer interface combined with functional magnetic resonance imaging (fMRI). IEEE transactions on biomedical engineering. 2004;51(6):971-974

[7] Scherer R, Lee F, Schlogl A, Leeb R, Bischof H, Pfurtscheller G. Toward Self-Paced Brain-Computer Communication: Navigation Through Virtual Worlds. IEEE transactions on biomedical engineering. 2008;55(2):675-682

[8] Iturrate IÃ, Antelis JM, Kubler A, Minguez J. A noninvasive brain-actuated wheelchair based on a P300 neurophysiological protocol and automated navigation. IEEE Transactions on Robotics. 2009;25(3):614-627

[9] Scherer R, Muller GR, Neuper C, Graimann B, Pfurtscheller G. An asynchronously controlled EEG-based virtual keyboard: improvement of the spelling rate. IEEE transactions on biomedical engineering. 2004;51(6):979-984

[10] Farwell LA, Donchin E. Talking off the top of your head: toward a mental prosthesis utilizing event-related brain potentials. Electroencephalography and clinical Neurophysiology. 1988;70(6):510-523

[11] Garcia-Larrea L, Lukaszewicz AC, Mauguiere F. Revisiting the oddball paradigm. Non-target vs neutral stimuli and the evaluation of ERP attentional effects. Neuropsychologia. 1992;30(8):723-741

[12] Segalowitz SJ, Barnes KL. The reliability of ERP components in the auditory oddball paradigm. Psychophysiology. 1993;30(5):451-459

[13] Guger C, Daban S, Sellers E, Holzner C, Krausz G, Carabalona R, Gramatica F, Edlinger G. How many people are able to control a P300-based brain-computer interface (BCI)? Neuroscience letters. 2009;462(1):94-98

[14] Frain B. Responsive web design with HTML5 and CSS3, Packt Publishing Ltd, (2012)

[15] Lie HW, Bos B. Cascading style sheets: Designing for the web, Portable Documents, Addison-Wesley Professional, (2005)

[16] Flanagan D. JavaScript: the definitive guide, O'Reilly Media, Inc., (2006)

[17] De Volder K. JQuery: A generic code browser with a declarative configuration language, in Proc. International Symposium on Practical Aspects of Declarative Languages, 2006, 88-102

[18] Duvinage M, Castermans T, Dutoit T, Petieau M, Hoellinger T, Saedeleer CD, Seetharaman K, Cheron G. A P300-based quantitative comparison between the Emotiv Epoc headset and a medical EEG device. Biomedical Engineering. 2012;765(1):2012-2764