# **FROM 10st July TO 16st July**

# **Project ID:**

# **2021J\_BV01\_BCI Browser**

# **Project Title:**

# **Design and development of Brain Computer Interface Browser on Web and Mobile**

# **Summary:**

P300 spellers are among the most popular types of brain–computer interfaces (BCIs) and are extremely useful assistive devices that enable severely disabled patients to communicate. However, P300 speller performances should be further improved to translate laboratory designs into practical applications. We aimed to design a new speller paradigm that could evoke higher event-related potentials (ERPs) than traditional P300 spellers, thus improving the performance of BCI systems.

# **Detail:**

In this study, we propose a novel three-dimensional (3-D) paradigm to increase the reliability of identifying target flashes. Various previous studies have attempted to combine BCIs with a 3-D virtual scene to develop new BCI applications. We proposed a new P300 speller paradigm based on three-dimensional (3-D) stereo visual stimuli. In this paradigm, flashing buttons are presented in 3-D stereo form. We designed two experiments, one that tested a traditional two-dimensional (2-D) speller and another that tested the proposed 3-D speller.

We also list the steps taken by us to deploy our P300 Speller Application made using Node.js to Heroku server.

**METHODS-:**

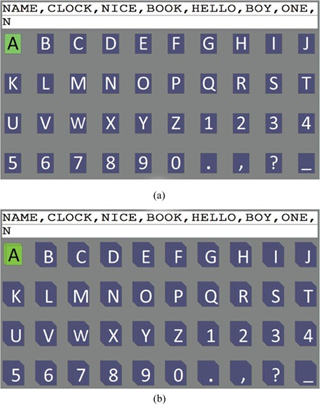
**Subjects-:** Twelve healthy volunteers between the ages of 19 and 32 years (mean, 23.5 years) participated in our experiment.

**EXPERIMENT-:**

We compared the ERPs elicited by the 2-D speller and the 3-D speller, and we also compared the classification accuracy, information transfer rate (ITR), and user workload between the two paradigms.

**Paradigms Design-:**

After preparation for EEG recording, the subject was seated approximately 100 cm in front of a 27-in LED monitor. The graphical user interface (GUI) was presented on the full screen with a refresh rate of 60 Hz and a resolution of 1920 × 1080 pixels. For comparison, two P300 speller paradigms were designed. One paradigm corresponded to a conventional 2-D paradigm, and the other paradigm was our novel 3-D paradigm. In our experiments, we used a 3-D LED monitor (LG D2792P, LG Electronics, Seoul, Korea), which displays 3-D views based on a polarization-multiplexed approach . The LED monitor can be switched between 2-D and 3-D mode. In the 2-D mode, the monitor can be used as a standard LED monitor, whereas in the 3-D mode, the monitor can merge adjacent pairs of side-by-side images to transform the view into polarized stereo images. In the first paradigm, 40 spelling characters were presented in a 4 × 10 2-D button matrix. The size of each button was 3 × 3.5 cm. We used an SC flash pattern in which each button flashed one by one. The LED monitor was set to 2-D mode to serve as an ordinary LED monitor. Inactive buttons were shown in blue on the background, and the color of each button changed to green when the button flashed. Each flash corresponded to when the stimulus changed from the background stimulus. During one round, each of the 40 buttons flashed once. The interstimulus interval (ISI) was set to 30 ms, and the stimulus duration was set to 100 ms. A group of rounds of the same target was called a trial. In the second paradigm, we designed 40 3-D buttons based on stereo display technology using the open source 3-D graphics engine OpenSceneGraph [Fig. 1(b)]. We chose this graphics engine because it could easily realize a stereo scene. The length and width of each 3-D button were the same as the 2-D button, but the 3-D button also has a depth of 3 cm. To allow each subject to see in stereo vision, the LED monitor was set to 3-D mode, and the subject wore 3-D glasses. Unlike the 2-D paradigm, when a 3-D button flashed, in addition to its color changing, a stereo depth of 1.5 cm toward the inside of the monitor was applied to the button such that the button appeared as though it had been “pressed” on a keyboard. The other parameters of the 3-D paradigm were the same as in the 2-D paradigm.



Two spelling paradigms were designed and employed in this study. (a) 2-D spelling paradigm (LED monitor set to 2-D mode). (b) 3-D spelling paradigm (based on 3-D flashing cubes). This figure provides only a sketch of the paradigm; to visualize the real 3-D stereo image, an LED monitor needs to be set to 3-D mode, and the viewer must wear 3-D glasses. Each stimulus in the 3-D paradigm, corresponding to a character, contained a fast flash (color change) and a 3-D motion of the cube (depth change).

**Results:**

The 3-D P300 speller elicited higher amplitudes of P300 waveforms than the traditional 2-D P300 speller. The online experimental results showed that the classification accuracy and the ITR were significantly improved with the 3-D P300 speller. We also found that the user workload of the 3-D P300 speller was significantly lower than that of the 2-D P300 speller

**HEROKU SERVER**

Heroku is a container-based cloud Platform as a Service (PaaS). Developers use Heroku to deploy, manage, and scale modern apps. Heroku is fully managed, giving developers the freedom to focus on their core product without the distraction of maintaining servers, hardware, or infrastructure.

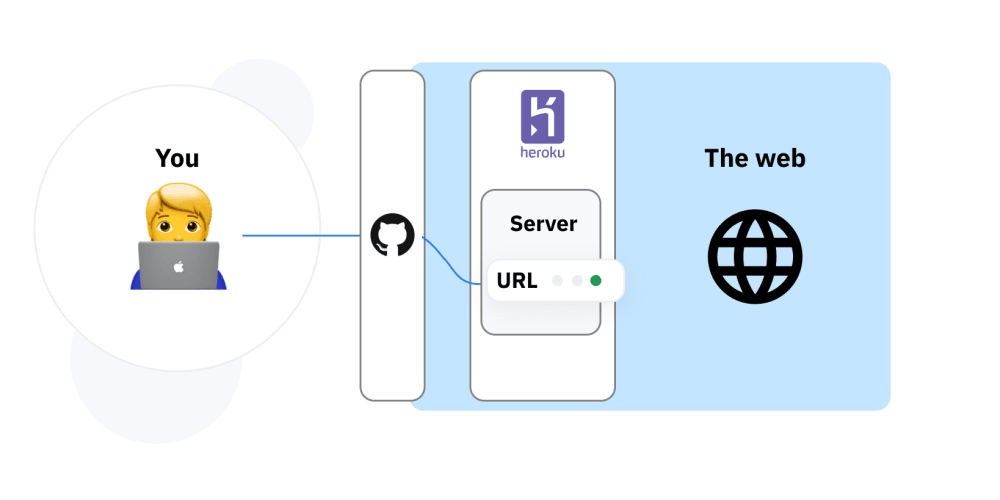


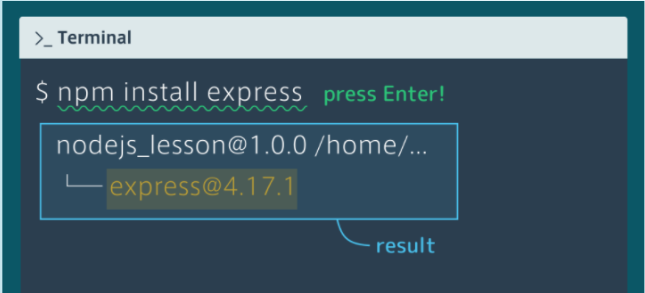
FIGURE- Deploying server to heroku

**NODE.JS**

Node.js is an open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.JavaScript is a client-side language, but Node.js allows it to be used for server-side as well.

**First step is to install npm.**

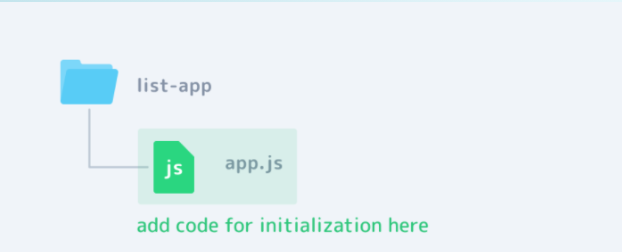
There is a system called npm (Node Package Manager) that lets you use packages. With npm, you can share and download packages.

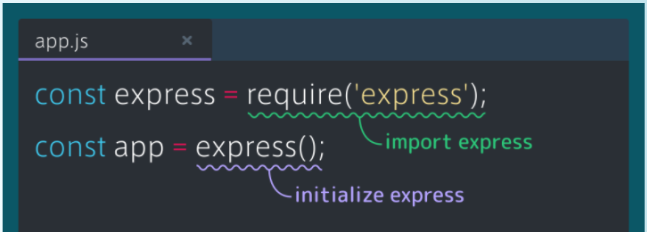


**Then we need to set up a tool called Express.**

Express is a framework for developing web apps with Node.js.Using frameworks helps make development a lot more efficient.

To use Express in your application, you need to import the package and add some code to initialize it, as shown below.

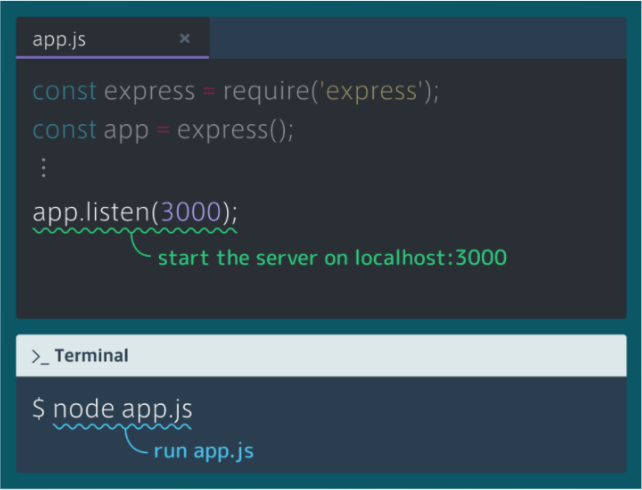




**Then, we start the server using Express.**

By starting the server, it will become able to receive requests and return responses, which will allow the web app to be displayed on the browser.

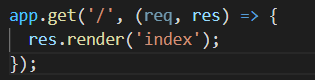
To start the server, we need to add the listen method to app.js, then run it in the terminal using the following command: node file\_name.



Determining how the app responds to a request to a particular URL is called routing. Each route can handle a request with a function, known as a route handler, that has req (request) and res (response) as parameters. These two parameters will hold information about the request and response.

****

You can specify which view file to show on the browser using the res.render function.

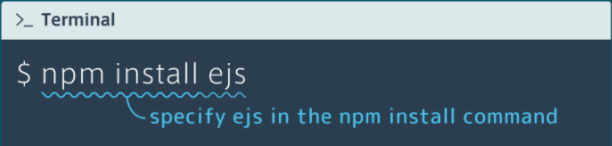


To serve static files such as images, CSS files, and JavaScript files,we use the express.static built-in middleware function in Express.



EJS is a Node.js package that lets you embed JavaScript code in your HTML file.

It stands for embedded JavaScript.EJS can be installed using npm



To embed JavaScript code, we can use <% %> or <%= %>.

<% %> is used in cases like defining a variable as it won't be displayed. <%= %>, on the other hand, is used for cases like printing a variable as it will be displayed.

Now to deploy the node.js application to our Heroku server,we took the following steps:

Make a free account on Heroku.

Download Heroku CLI on PC.

Download Git on PC.

Declare app dependencies:

The package.json file defines the dependencies that should be installed with your application. To create a package.json file for your app, run the command npm init in the root directory of your app. It will walk you through creating a package.json file. Use the Git Bash application to open a command shell on Windows.To install dependencies, use npm install <pkg>. This will install the package and also add it as a dependency in the package.json file. The version of Node.js that will be used to run your application on Heroku, should also be defined in your package.json file. You should always specify a Node.js version that matches the runtime you’re developing and testing with. To find your version type node --version.

Now that the dependencies are installed and the version of node to use has been specified, the package.json file will look something like this:



**Specifying a start script**

To determine how to start your app, Heroku first looks for a [Procfile](https://devcenter.heroku.com/articles/procfile). If no Procfile exists for a Node.js app, we will attempt to start a default web process via the [start script](https://docs.npmjs.com/misc/scripts) in your package.json.

The command in a web process type must bind to the port number [specified in the PORT environment variable](https://devcenter.heroku.com/articles/dynos#local-environment-variables). If it does not, the dyno will not start.

**Build the app and run it locally**

Run the npm install command in your local app directory to install the dependencies that you declared in your package.json file.

$ npm install

Start your app locally using the heroku local command, which is installed as part of the Heroku CLI.

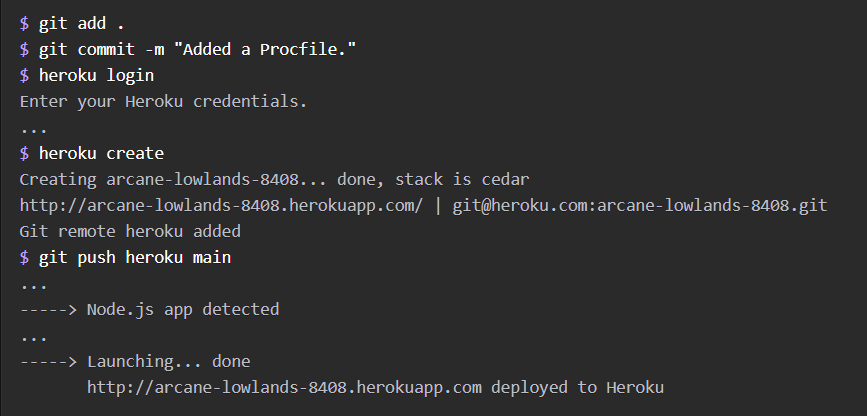
$ heroku local web

The app would now be running on [http://localhost:3000/](http://localhost:5000/).

Prevent build artifacts(node\_modules) from going into revision control by creating a [.gitignore](https://devcenter.heroku.com/articles/gitignore) file

**Deploying Application to Heroku**

After you commit your changes to git, you can deploy your app to Heroku.



To open the app in your browser, type heroku open.

Thus the app will be deployed on Heroku server.

Link to the P300 speller application:

https://blooming-hamlet-51215.herokuapp.com/

**CONCLUSION-:**

The proposed 3-D P300 speller based on stereo visual stimuli outperformed a traditional 2-D P300 speller. This finding indicates that our 3-D paradigm offers a new method that will improve the performance of P300 BCI systems. The Node.js application was deployed on Heroku Server.