# **FROM 21st August TO 27th August**

# **Project ID:**

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

# **Project Title:**

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

# **Summary:**

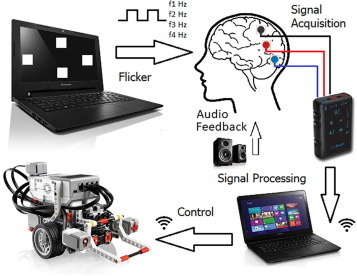
* SSVEP based BCI

# **Detail:**

**SSVEP BASED BCI**

People which are afflicted with neurological conditions or neurodegenerative diseases can’t control own muscles by neural pathways. [Brain computer interface](https://www.sciencedirect.com/topics/engineering/brain-computer-interface) (BCI) systems offer these people another alternative path from their own neural pathways. This [alternative pathway](https://www.sciencedirect.com/topics/engineering/alternative-pathway) is the direct use of brain signals by a computer without using any vocal muscle. The steady state visual evoked potential (SSVEP) approach currently provides the high performance and reliable communication for the implementation of a non-invasive BCI. In SSVEP based BCI systems, Electroencephalography (EEG) signal detection time (signal window length) and accuracy are the most important performance parameters. Performance is usually measured by Information Transfer Rate (ITR).

In the presented paper a SSVEP based BCI robot control application is introduced and system performance is analyzed for different signal window lengths. At first, the number of eye blinks of the subjects is determined by fast eye [artifact](https://www.sciencedirect.com/topics/engineering/artefact) detection method (FEAD) which based on visual eye blink detection. These eye blink counts are used for system activation. [System usability](https://www.sciencedirect.com/topics/computer-science/usability-systems) is improved by this control. Two consecutive eye blinks which detecting by FEAD method are used for system activation. System deactivation is also provided by the same command. Synchronous and asynchronous experiments are performed on four healthy subjects for performance analyses. EEG data is analyzed in details by asynchronous experiments. During the synchronous experiments, subjects are tried to complete a predefined route which has twelve steps by navigating the robot (Lego Mindstorms EV3). The minimum energy combination (MEC) and canonical correlation analysis (CCA) methods are applied to EEG segments that are different in length in order to detect SSVEPs in both type experiments. ITR values are calculated for different signal window lengths. The results show that the detection accuracy of the MEC method is similar to that of the CCA method, although it is higher than that of the CCA method in situations where the SSVEP has low strength. In synchronous experiments, using MEC method a system peak ITR of 133.33 bit/min is reached for one subject with a 0.9 s signal window length.



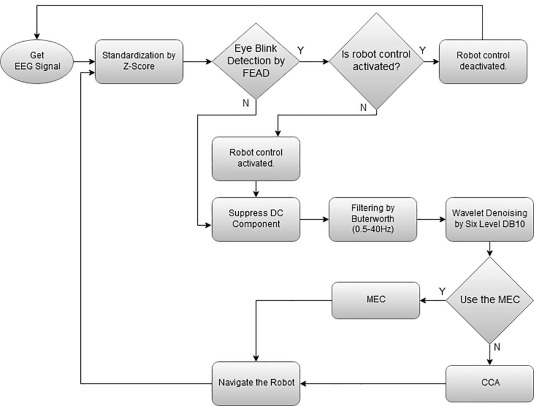


Fig:- Flowchart of Signal Processing

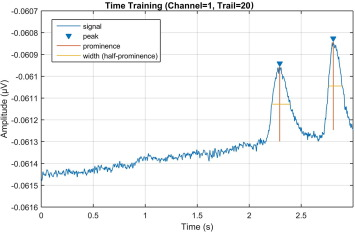


Fig:- Samples trial by eye vertical blink [artifact](https://www.sciencedirect.com/topics/engineering/artefact) and FEAD properties

# **How to create a file upload form**

**Introduction:** File uploading means a user from client machine requests to upload file to the server. For example, users can upload images, videos, etc on Facebook, Instagram, etc.

**Features of Multer module:** File can be uploaded to the server using Multer module. There are other modules in market but multer is very popular when it comes to file uploading. Multer is a node.js middleware which is used for handling multipart/form-data, which is mostly used library for uploading files.

**Note:** Multer will process only those form which are multipart (multipart/form-data). So whenever you use multer, make sure you put multipart in form.

**Introduction:**

1. It’s easy to get started and easy to use.
2. It is widely used and popular module for file uploading.
3. Users can upload either single or multiple files at a time.

**Installation of Multer module:**

1. You can visit the link [Install multer module](https://www.npmjs.com/package/multer). You can install this package by using this command.  
   npm install multer
2. After installing multer you can check your multer version in command prompt using the command.  
   npm version multer
3. After that, you can just create a folder and add a file for example index.js, To run this file you need to run the following command.  
   node index.js
4. **Requiring module:** You need to include multer module in your file by using these lines.  
   var multer = require('multer');
5. So Multer basically adds a file object or files object and a body object to the request object. The file/files object contains all the files which are uploaded through the form and all the values of the text fields of the form are contained in the body object. This is how multer binds the data whenever a form is submitted.  
   **Filename: Signup.ejs**

| <!DOCTYPE html>  <html>    <head>  <title>FILE UPLOAD DEMO</title>  </head>    <body>  <h1>Single File Upload Demo</h1>    <form action="/uploadProfilePicture"  enctype="multipart/form-data" method="POST">    <span>Upload Profile Picture:</span>  <input type="file" name="mypic" required/> <br>  <input type="submit" value="submit">  </form>  </body>    </html> |
| --- |

1. **Filename: index.js**

**CREATION OF REGISTRATION PAGE:CODE SNIPPET**

**APP.JS**

**// bcrypt**

**var bcrypt = require("bcrypt");**

**// body-parser**

**var bodyParser = require("body-parser");**

**var urlEncodedParser = bodyParser.urlencoded({extended: false});**

**// express**

**var express = require("express");**

**var reactViews = require('express-react-views');**

**var app = express();**

**app.set('views', \_\_dirname + '/views');**

**app.set('view engine', 'jsx');**

**app.engine('jsx', reactViews.createEngine());**

**//mysql**

**var mysql = require("mysql");**

**// connect strings for mysql**

**var connection = mysql.createConnection({**

**host: "localhost",**

**user: "root",**

**password: "Morley@99",**

**database: "test"**

**});**

**// connecting ......**

**connection.connect();**

**// requesting express to get data as text**

**app.use(bodyParser.text());**

**// using express for post method**

**app.post("/register", urlEncodedParser, function(request, response) {**

**if(request.url!="/favicon.ico") {**

**if(request.body.regOrLogin=="Register") {**

**bcrypt.genSalt(10, function(err, salt) {**

**bcrypt.hash(request.body.pwd, salt, function(err, hash) {**

**var body = request.body;**

**var date = new Date();**

**var currentDate = date.getFullYear()+"-"+date.getMonth()+"-"+date.getDay();**

**var postVars = {username: body.username, password: hash, dob: body.dob, reg\_date: currentDate};**

**// insertion into MySQL**

**connection.query("SELECT \* FROM user WHERE username='"+body.username+"'", function(err, res, fields){**

**if(err) {**

**response.render( "Error.jsx", {error: 'Error while querying', name: 'Register'});**

**} else {**

**if(res.length) {**

**response.render( "Error.jsx", {error: 'User already exist!!', name: 'Register'});**

**}**

**else {**

**connection.query("INSERT INTO user set ?", postVars, function(err, result) {**

**if(err) {**

**console.log("error", err);**

**response.render( "Error.jsx", {error: 'User registration problem', name: 'Register'});**

**} else {**

**response.render( "Login.jsx");**

**}**

**});**

**}**

**}**

**});**

**});**

**});**

**}**

**}**

**});**

**app.post("/login", urlEncodedParser, function(request, response) {**

**if(request.url != "/favicon.ico") {**

**if (request.body.regOrLogin=="Login") {**

**var body = request.body;**

**connection.query("SELECT \* FROM user WHERE username='"+body.username+"'", function(err, res, fields){**

**if(err) {**

**response.render( "Error.jsx", {error: 'Username and Password is not correct'});**

**} else {**

**if(res.length) {**

**bcrypt.compare(body.pwd, res[0].password, function(err, res) {**

**if(res) {**

**response.render( "Welcome.jsx", {name: body.username});**

**} else {**

**response.render( "Error.jsx", {error: 'Password is not correct'});**

**}**

**});**

**} else {**

**response.render( "Error.jsx", {error: 'Username is not correct'});**

**}**

**}**

**});**

**}**

**}**

**})**

**app.get('/', function(req, res) {**

**// render the page and pass in any flash data if it exists**

**res.render('Login.jsx');**

**});**

**app.get('/register', function(req, res) {**

**res.render('Register.jsx');**

**});**

**app.listen(3000);**

**REGISTER.JSX**

**import React from 'react'**

**const Register = React.createClass({**

**render() {**

**return (**

**<section className="column is-offset-6 is-4">**

**<center>**

**<h1>Register</h1>**

**<p>Enter your username and password</p>**

**<table>**

**<form action="/register" method="post">**

**<tr>**

**<td>Username</td>**

**<td><input type="text" name="username"/></td>**

**</tr>**

**<tr>**

**<td>Password</td>**

**<td><input type="password" name="pwd"/></td>**

**</tr>**

**<tr>**

**<td>Date of birth:</td>**

**<td> <input type="date" name="dob"/></td>**

**</tr>**

**<tr>**

**<td align="center" colspan="2"><input type="submit"value="Register" name="regOrLogin" class="class1"/></td>**

**<td></td>**

**</tr>**

**</form>**

**</table>**

**<td align="center" colspan="2"><a href='/' ><button>Login</button></a></td>**

**</center>**

**</section>**

**)**

**},**

**});**

**export default Register**

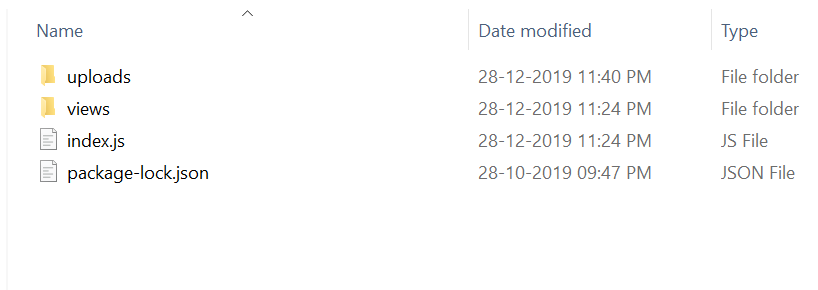
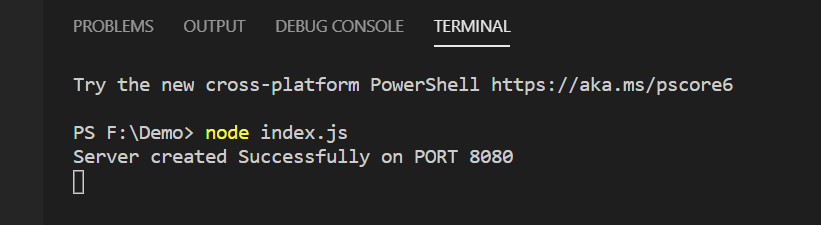
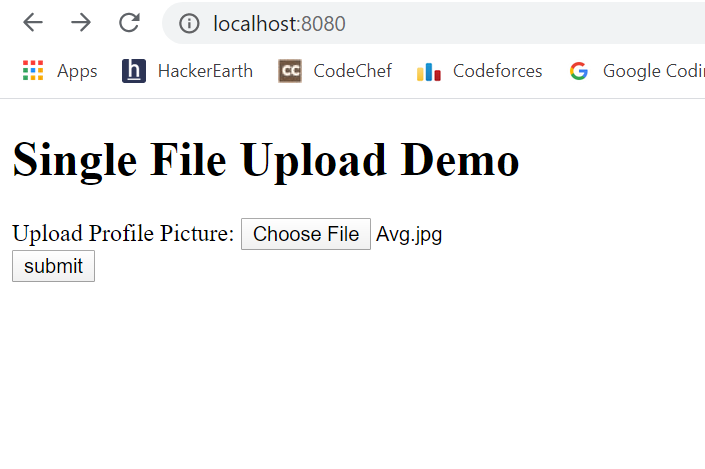
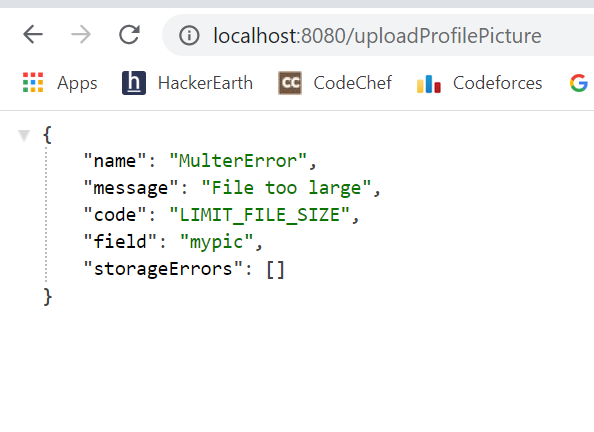
**Db.sql**

**create database test;**

**use test;**

**create table user (username varchar(255), password varchar(255), dob date, reg\_date date);**

| const express = require("express")  const path = require("path")  const multer = require("multer")  const app = express()    // View Engine Setup  app.set("views",path.join(\_\_dirname,"views"))  app.set("view engine","ejs")    // var upload = multer({ dest: "Upload\_folder\_name" })  // If you do not want to use diskStorage then uncomment it    var storage = multer.diskStorage({  destination: function (req, file, cb) {    // Uploads is the Upload\_folder\_name  cb(null, "uploads")  },  filename: function (req, file, cb) {  cb(null, file.fieldname + "-" + Date.now()+".jpg")  }  })    // Define the maximum size for uploading  // picture i.e. 1 MB. it is optional  const maxSize = 1 \* 1000 \* 1000;    var upload = multer({  storage: storage,  limits: { fileSize: maxSize },  fileFilter: function (req, file, cb){    // Set the filetypes, it is optional  var filetypes = /jpeg|jpg|png/;  var mimetype = filetypes.test(file.mimetype);    var extname = filetypes.test(path.extname(  file.originalname).toLowerCase());    if (mimetype && extname) {  return cb(null, true);  }    cb("Error: File upload only supports the "  + "following filetypes - " + filetypes);  }    // mypic is the name of file attribute  }).single("mypic");    app.get("/",function(req,res){  res.render("Signup");  })    app.post("/uploadProfilePicture",function (req, res, next) {    // Error MiddleWare for multer file upload, so if any  // error occurs, the image would not be uploaded!  upload(req,res,function(err) {    if(err) {    // ERROR occured (here it can be occured due  // to uploading image of size greater than  // 1MB or uploading different file type)  res.send(err)  }  else {    // SUCCESS, image successfully uploaded  res.send("Success, Image uploaded!")  }  })  })    // Take any port number of your choice which  // is not taken by any other process  app.listen(8080,function(error) {  if(error) throw error  console.log("Server created Successfully on PORT 8080")  }) |
| --- |

1. **Steps to run the program:**
   1. The project structure will look like this:  
        
      Here “uploads” is the folder where our files will be uploaded, currently it is empty. The “Singup.ejs” is kept in the views folder.
   2. Make sure you have ‘view engine’ like I have used “ejs” and also install express and multer using following commands:  
      npm install ejs  
      npm install express  
      npm install multer
   3. Run index.js file using below command:  
      node index.js  
      
   4. Open browser and type this URL:  
      http://localhost:8080/
   5. Then you will see the Singup form as shown below:  
      
   6. Then choose a file to be uploaded and click on submit button.  
      If error occurs, then following message will be displayed:  
        
      And if no errors occurs, then following message will be displayed:  
      