Ujwal Dinesha

UNI: ud2130

ECBM 4090: BCI Mini Project 2

1.Introduction

Brain computer interface (BCI) paradigms in decoding imagined movement are largely based on the Mu waves, which are synchronized patterns that appear when the motor cortex is idle. Mu waves appear between 7.5Hz and 12.5Hz. Unlike the alpha wave, which occurs at a similar frequency over the resting visual cortex at the back of the scalp, the mu wave is found over the motor cortex, in a band approximately from ear to ear.

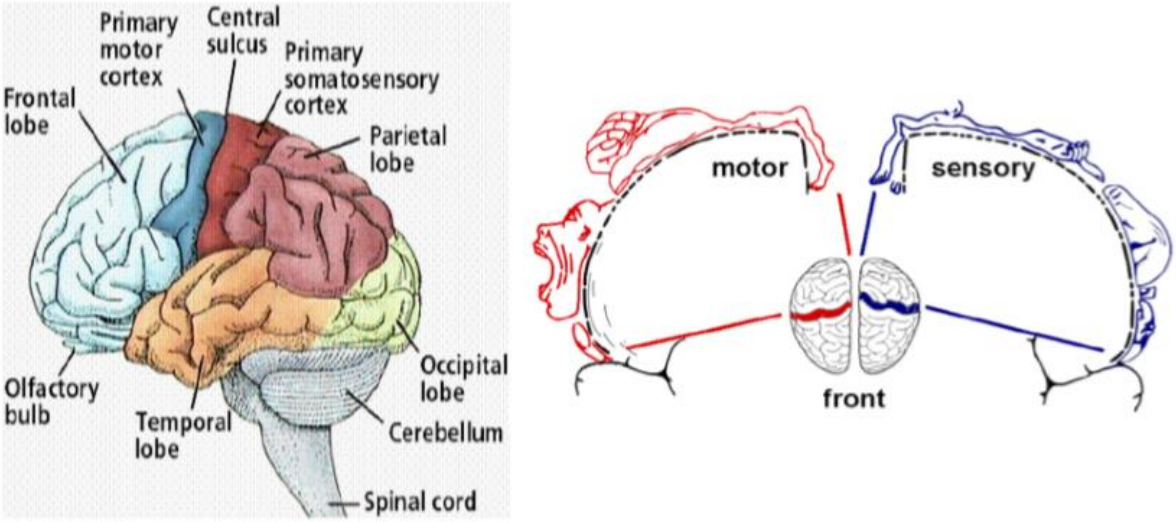


Figure 1: Left - Primary motor cortex and other parts of the brain Right- Mapping of motor cortex to control of different body parts

A person suppresses mu wave patterns when he or she performs a motor action or, with practice, when he or she visualizes performing a motor action. This suppression is called desynchronization of the wave because EEG wave forms are caused by large numbers of neurons firing in synchrony. The mu wave is even suppressed when one observes another person performing a motor action or an abstract motion with biological characteristics. Since the right motor cortex controls the left half of the body and vice versa, the spatial pattern of Mu wave

activity can be used to determine left and right imagined movements. A technique called common spatial projection (CSP) aids us in improving the separability of the EEG waves corresponding to left and right imagined movements.

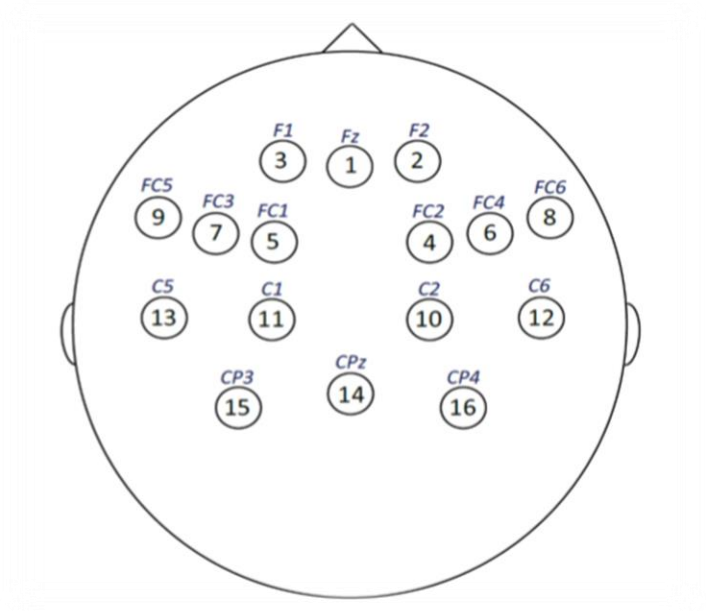


Figure 2: Electrode placement for the experiment

This experiment uses an existing Simulink module to train a CSP filter and return feedback to train subject to accurately perform imagined movements of the left and right hand. The g.Tech hardware and software components are used for brain data collection. Figure 2 depicts the positions of the electrodes around the scalp. The subject wore a cap containing all of these 16 electrodes and gel was placed to create a low-impedance connection between each electrode and the surface of the scalp.

An overview of the experimental paradigm is depicted in figure 3. The subject starts the experiment by training a classifier. In this stage, the subject starts by fixating on a cross. Then, instructions are given for the subject to imagine movement either in the left hand or right hand. Then, the imagined movement data is collected for 40 trials. After this initial run, the data is loaded into gbsanalyze and used to calculate the CSP and weight vector. Upon completion, a two class CSP classifier is created.

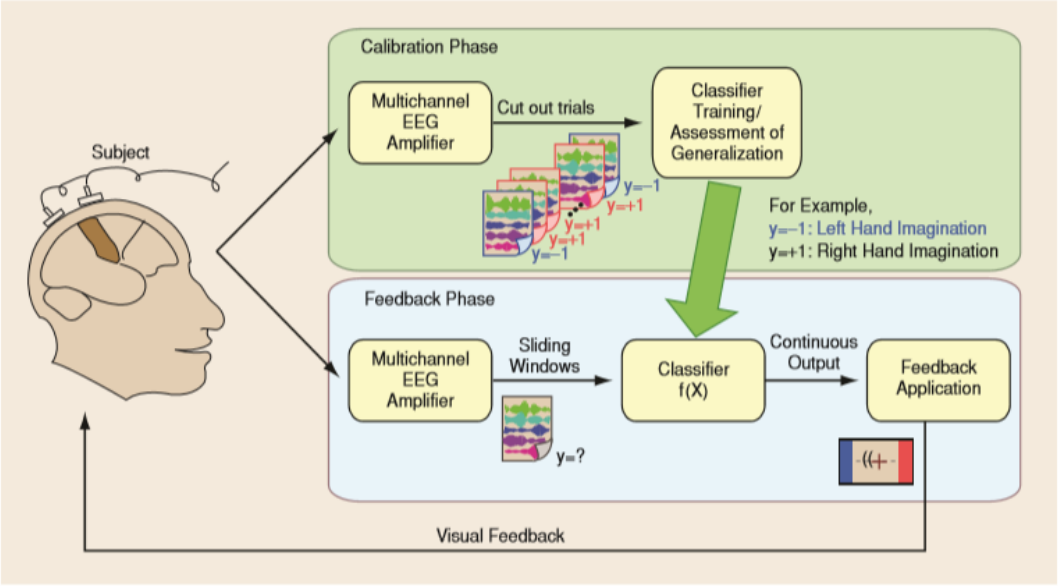


Figure 3: Overview of the machine-learning-based BCI system. The system runs in two phases. In the calibration phase, we instruct the subjects to perform certain tasks and collect short segments of labeled EEG (trials). We train the classifier based on these examples. In the feedback phase, we take sliding windows from continuous stream of EEG; the classifier outputs a real value that quantifies the likeliness of class membership; we run a feedback application that takes the output of the classifier as an input. Finally the subject receives the feedback on the screen as, e.g., cursor control.

Once the classifier is trained, the one with the lowest error is applied in the Simulink model and the feedback is turned on for the subsequent runs. In these runs, the subject is instructed to imagine movement in left and right hands as before, but, this time, live feedback is given to the subject. Furthermore, this phase of the experiment allows for the subject to train his brain to effectively imagine movement with accurate classification by trying different types of behaviors. Detailed timeline of events in each trial is depicted in figure 5.

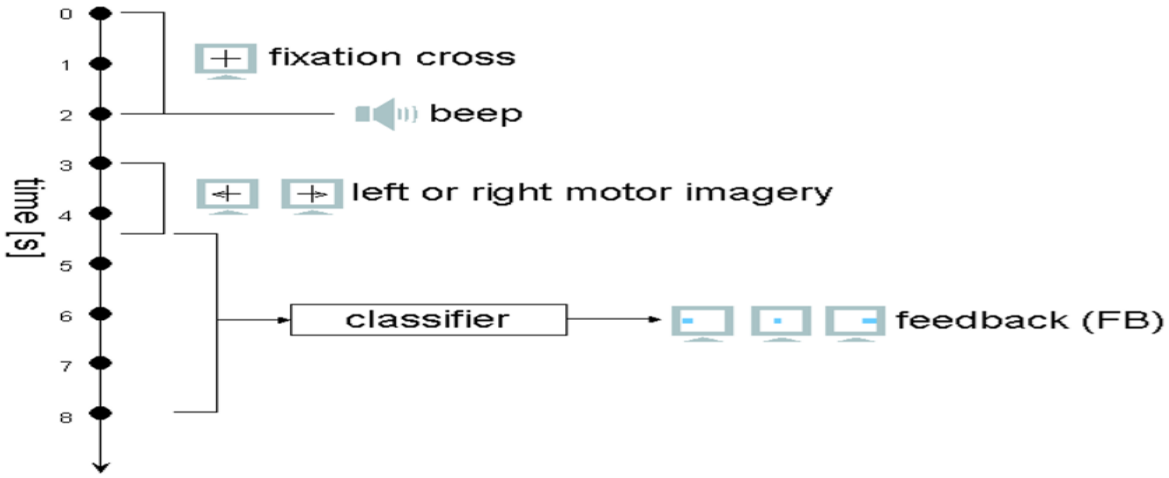


Figure 4: Timeline of events in each trial

2.Stimuli and procedure

Data from 16 electrodes was collected from locations shown in figure2. All of the data was filtered with a notch filter at 60Hz to reduce powerline noise. A bandpass filter was used from 0.5Hz to 30Hz during the data collection. In post-processing, another band pass filter from 8Hz to 20Hz was used.

The sampling frequency (Fs) of the equipment was 256Hz. The results represent analysis from eight runs of the paradigm. Each run involves 40 trials. Therefore, a total of 320 trials with 160 left movement imagination trials and 160 right movement imaginations were recorded. The first run was used to train the classifier and the remaining seven runs included feedback on the monitor to train the subject. In the left imagination case, the subject imagined punching her hand in the air, and in the right imagination case, the subject found that rotating her right wrist produced the best results.

A trigger representing the 2 second mark for all trials was stored in row 18 for all of the runs, and a separate file contained the series of cue data. Based on this information, the data was separated into left and right trials. The signal quality was checked using the impedance checker block provided as a Simulink extension with the g.Tech software package. The impedance was checked before each run and the subject was periodically re-gelled to satisfy all of the impedance requirements.

3.Results

3.1) Simulink Classifier –

This was the classifier that was trained during the first run and then subsequently used to provide feedback in order to train the subject. The error rate of this classifier was about 12.5%. More specifics about the classifier is represented in the images below.

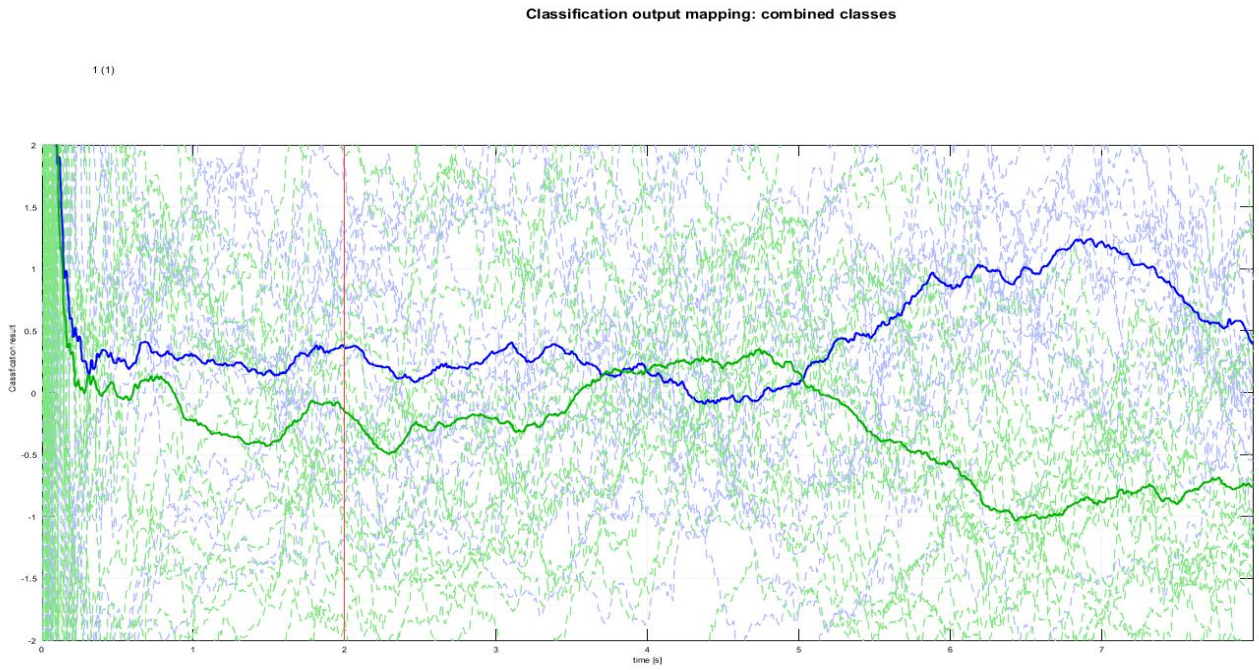


Figure 5:Separation in left and right imagined movement classification.

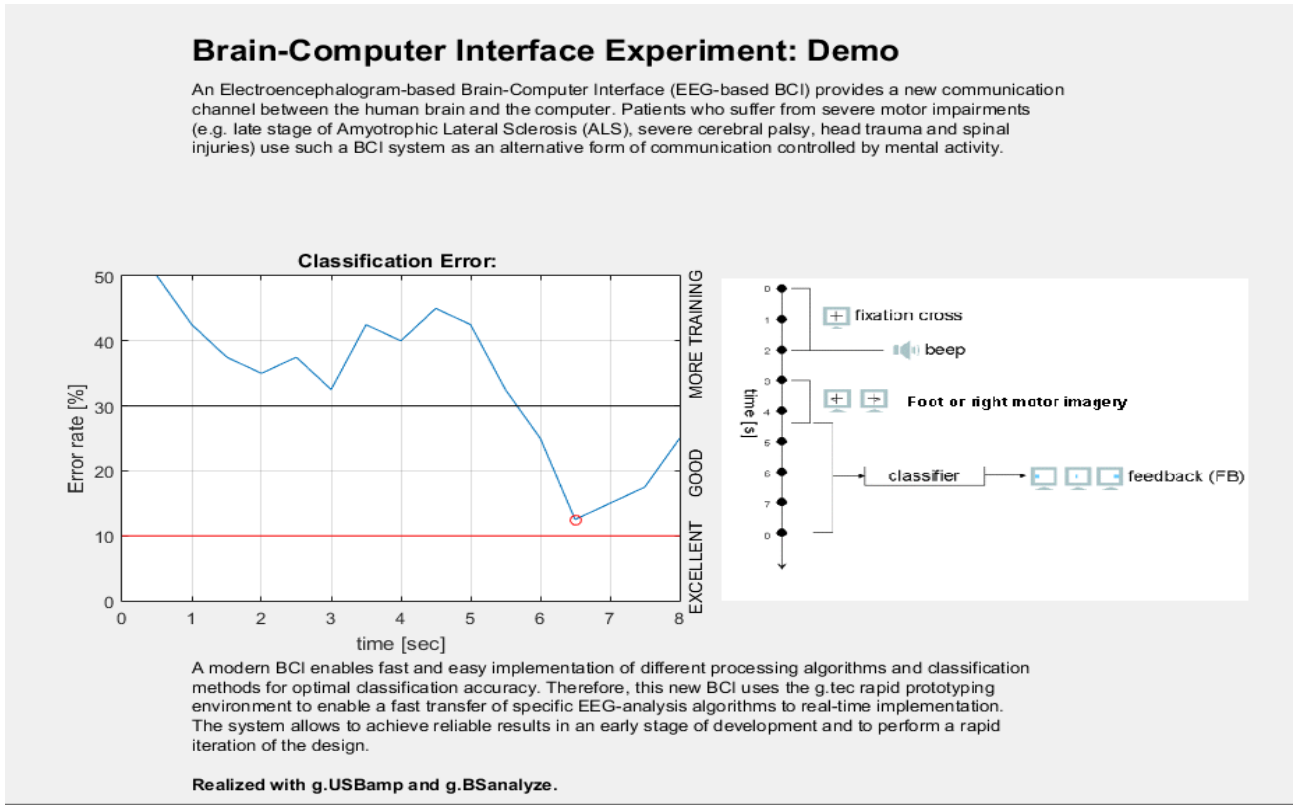


Figure 6: Classification error overview

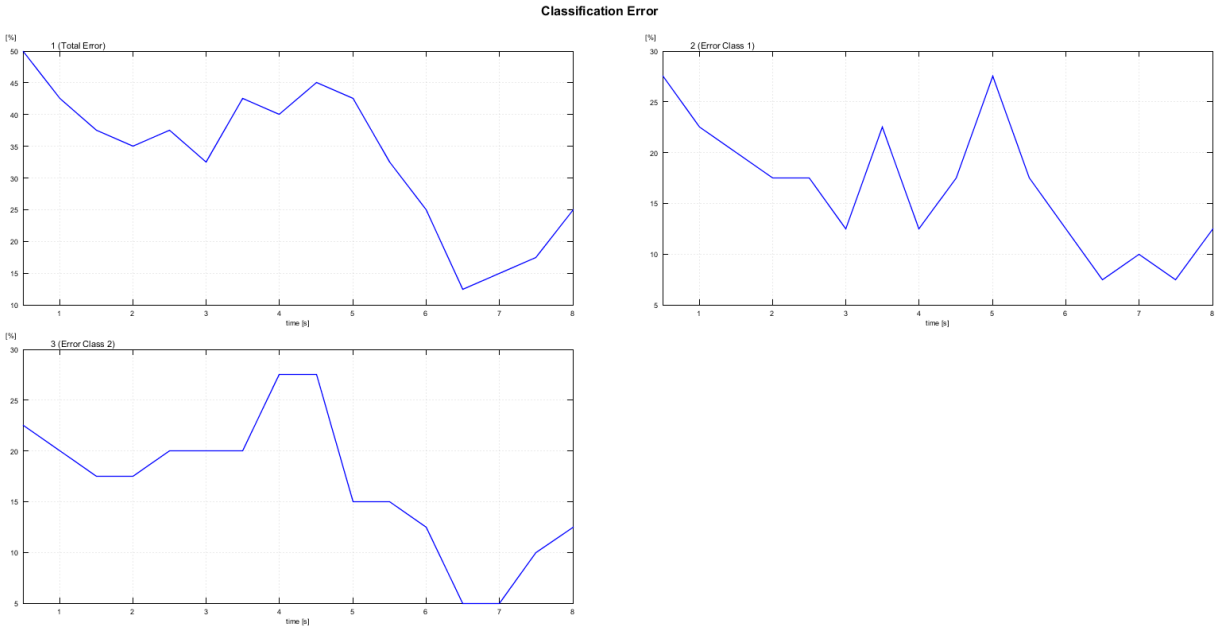


Figure 7: Classification error specifics

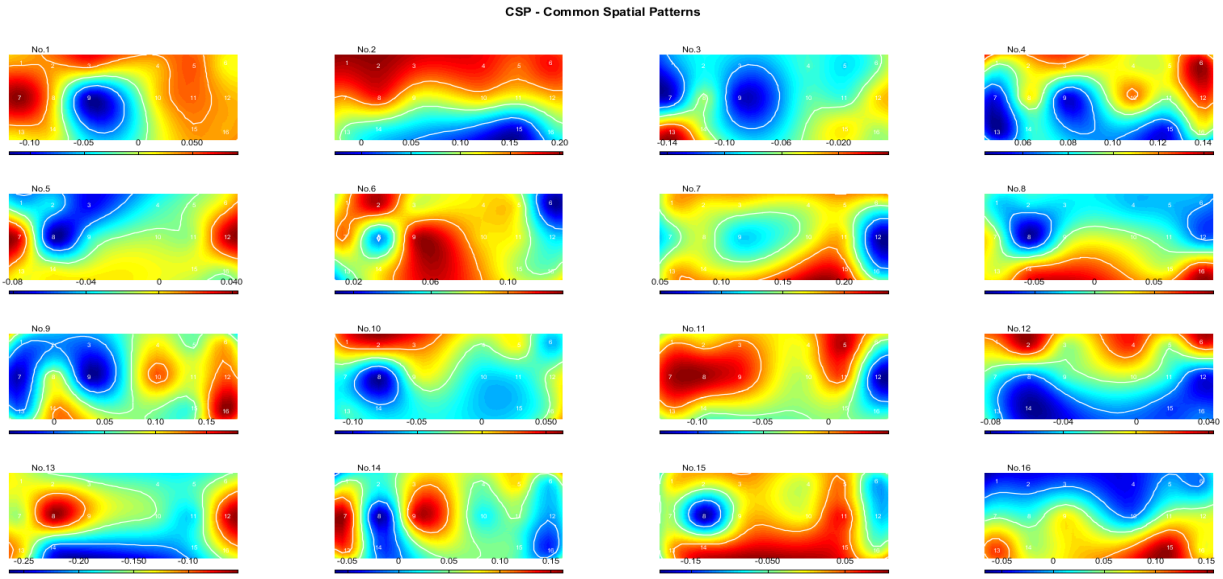


Figure 8: Associated CSPs

3.1) MATLAB Classifier –

This is the classifier we built using data from all the trial runs. Following is the code.

Contents

Mini Project 2 Ujwal Dinesha ud2130

```
clear all; close all; clc;
```

Load and process data

```
run1=load('run1_1.mat').y;
run2=load('run2_1.mat').y;
run3=load('run3_1.mat').y;
run4=load('run4_1.mat').y;
run5=load('run1_2.mat').y;
run6=load('run2_2.mat').y;
run7=load('run3_2.mat').y;
run8=load('run4_2.mat').y;

load('classrun1.mat');
load('classrun2.mat');
load('classrun3.mat');
load('classrun4.mat');

fs=256;

t=squeeze(bandpass(run1(2:17,:)',[8 20], fs));
run1(2:17,:) = t';
t=squeeze(bandpass(run2(2:17,:)',[8 20], fs));
run2(2:17,:) = t';
t=squeeze(bandpass(run3(2:17,:)',[8 20], fs));
run3(2:17,:) = t';
t=squeeze(bandpass(run4(2:17,:)',[8 20], fs));
run4(2:17,:) = t';
t=squeeze(bandpass(run5(2:17,:)',[8 20], fs));
run5(2:17,:) = t';
t=squeeze(bandpass(run6(2:17,:)',[8 20], fs));
run6(2:17,:) = t';
t=squeeze(bandpass(run7(2:17,:)',[8 20], fs));
run7(2:17,:) = t';
t=squeeze(bandpass(run8(2:17,:)',[8 20], fs));
run8(2:17,:) = t';

run1 = squeeze(run1);
run2 = squeeze(run2);
run3 = squeeze(run3);
run4 = squeeze(run4);
run5 = squeeze(run5);
run6 = squeeze(run6);
run7 = squeeze(run7);
run8 = squeeze(run8);


start_8 = find(diff(run8(18,:)) > 0);
start_7 = find(diff(run7(18,:)) > 0);
start_6 = find(diff(run6(18,:)) > 0);
start_5 = find(diff(run5(18,:)) > 0);


start_4 = find(diff(run4(18,:)) > 0);
start_3 = find(diff(run3(18,:)) > 0);
```

```
start_2 = find(diff(run2(18,:)) > 0);
start_1 = find(diff(run1(18,:)) > 0);


left=0;
right=0;
for i=1:40

    if z1(1,i)==1
        left = left+1;
        left_wave(left,:,:) = run1(2:17,start_1(i)+floor(2.5*fs):floor(6*fs)+start_1(i));
        left = left+1;
        left_wave(left,:,:) = run5(2:17,start_5(i)+floor(2.5*fs):floor(6*fs)+start_5(i));
    else
        right = right+1;
        right_wave(right,:,:) = run1(2:17,start_1(i)+floor(2.5*fs):floor(6*fs)+start_1(i));
        right = right+1;
        right_wave(right,:,:) = run5(2:17,start_5(i)+floor(2.5*fs):floor(6*fs)+start_5(i));
    end
    if z2(1,i)==1
        left = left+1;
        left_wave(left,:,:) = run2(2:17,start_2(i)+floor(2.5*fs):floor(6*fs)+start_2(i));
        left = left+1;
        left_wave(left,:,:) = run6(2:17,start_6(i)+floor(2.5*fs):floor(6*fs)+start_6(i));
    else
        right = right+1;
        right_wave(right,:,:) = run2(2:17,start_2(i)+floor(2.5*fs):floor(6*fs)+start_2(i));
        right = right+1;
        right_wave(right,:,:) = run6(2:17,start_6(i)+floor(2.5*fs):floor(6*fs)+start_6(i));
    end
    if z3(1,i)==1
        left = left+1;
        left_wave(left,:,:) = run3(2:17,start_3(i)+floor(2.5*fs):floor(6*fs)+start_3(i));
        left = left+1;
        left_wave(left,:,:) = run7(2:17,start_7(i)+floor(2.5*fs):floor(6*fs)+start_7(i));
    else
        right = right+1;
        right_wave(right,:,:) = run3(2:17,start_3(i)+floor(2.5*fs):floor(6*fs)+start_3(i));
        right = right+1;
        right_wave(right,:,:) = run7(2:17,start_7(i)+floor(2.5*fs):floor(6*fs)+start_7(i));
    end
    if z4(1,i)==1
        left = left+1;
        left_wave(left,:,:) = run4(2:17,start_4(i)+floor(2.5*fs):floor(6*fs)+start_4(i));
        left = left+1;
        left_wave(left,:,:) = run8(2:17,start_8(i)+floor(2.5*fs):floor(6*fs)+start_8(i));
    else
        right = right+1;
        right_wave(right,:,:) = run4(2:17,start_4(i)+floor(2.5*fs):floor(6*fs)+start_4(i));
        right = right+1;
        right_wave(right,:,:) = run8(2:17,start_8(i)+floor(2.5*fs):floor(6*fs)+start_8(i));
    end
end
```

Apply CSP

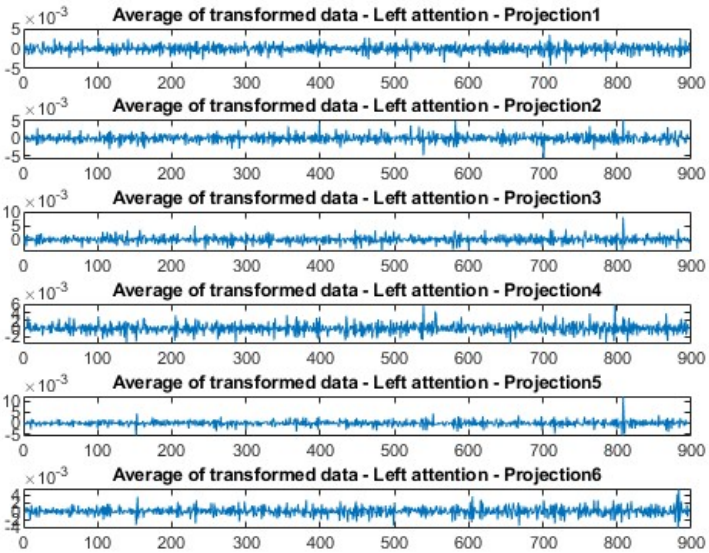
```
fprintf("Steps involved in applying CSP:-\n");
fprintf("1.)Calculate covariance matrices of each class, S1 and S2.\n");
fprintf("2.)Solve the generalized Eigen value problem to ?nd the best projection W.\n");
fprintf("3.) Choose the ?rst 6 Eigen vectors (columns of W) that correspond to the largest Eigen values:CSP projections.\n");
fprintf("4.) Project the data (channel x time) using the CSP projections (6 x time)\n");

lw = [];
rw = [];

for i=1:160
    lw = [lw squeeze(left_wave(i,:,:))];
    rw = [rw squeeze(right_wave(i,:,:))];
end


lw = squeeze(lw); rw = squeeze(rw);
```

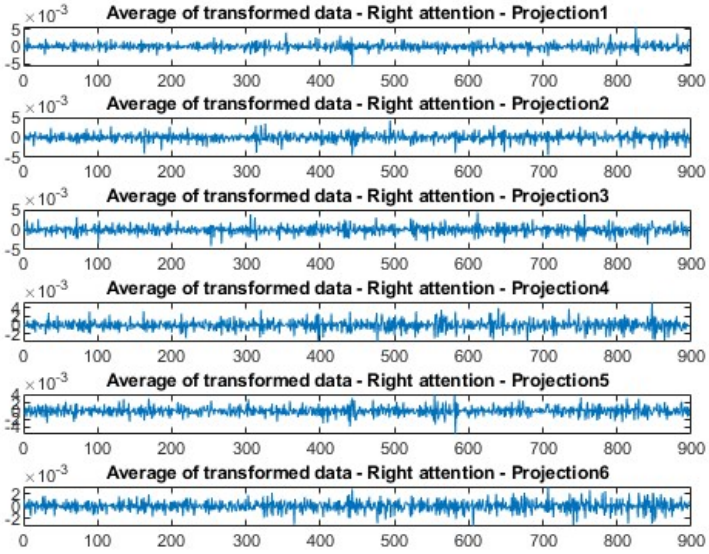
```
S1 = lw * lw'/160;
S2 = rw * rw'/160;

[W,X] = eig(S1,S2);

W = W(:,end-5:end); %Choose top 6 eigen vectors


left_CSP = W'*lw;
right_CSP = W'*rw;

left_std_CSP = std(reshape(left_CSP,6,160,[]),0,3)';
right_std_CSP = std(reshape(right_CSP,6,160,[]),0,3)';
```

```
Steps involved in applying CSP:-
1.)Calculate covariance matrices of each class, S1 and S2.
2.)Solve the generalized Eigen value problem to ?nd the best projection W.
3.) Choose the ?rst 6 Eigen vectors (columns of W) that correspond to the largest Eigen values:CSP projections.
4.) Project the data (channel x time) using the CSP projections (6 x time)
```

Plot after CSP filtering

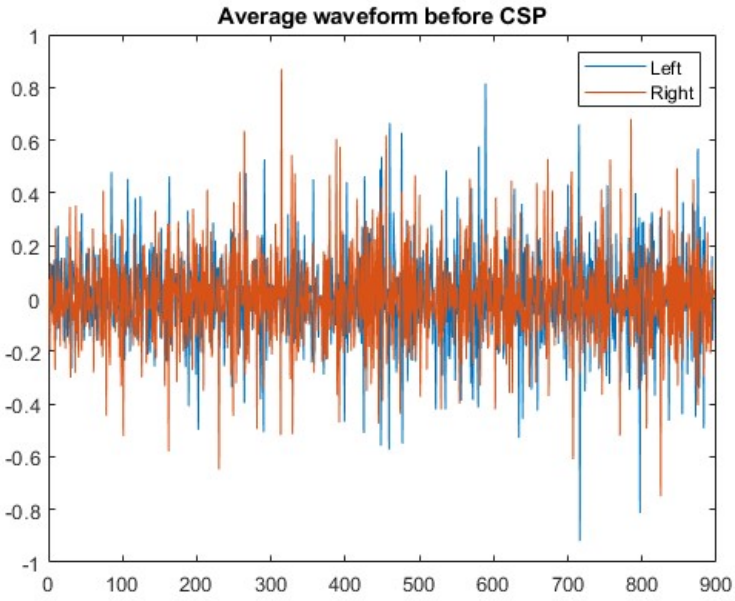
```
left_avg = squeeze(mean(reshape(left_CSP,6,160,[]),2));
right_avg = squeeze(mean(reshape(right_CSP,6,160,[]),2));
figure();
for i =1:6
    subplot(6,1,i);
    plot(left_avg(i,:));
    title(strcat('Average of transformed data - Left attention - Projection ',int2str(i)))
end
figure();
for i =1:6
    subplot(6,1,i);
    plot(right_avg(i,:));
    title(strcat('Average of transformed data - Right attention - Projection ',int2str(i)))
end
figure();
```

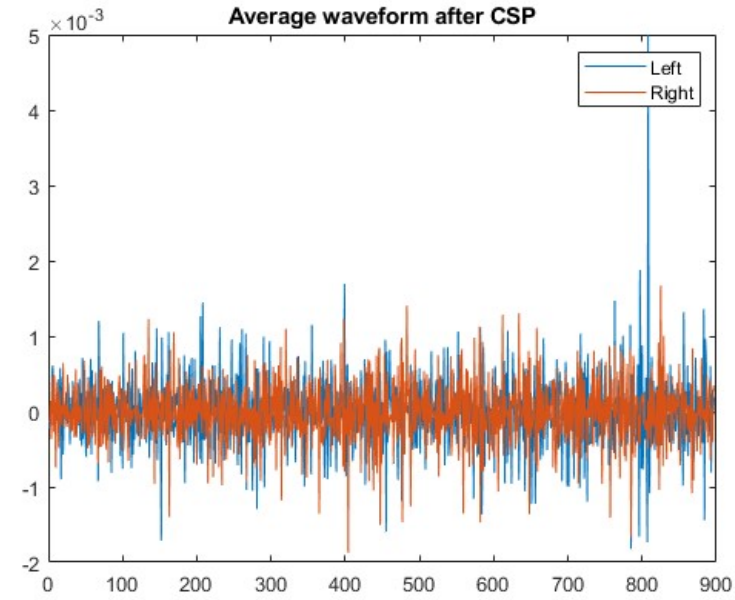




Demonstrate Separability via average waveform plots

```
left_avg_before = squeeze(mean(reshape(lw,16,160,[]),2));
right_avg_before = squeeze(mean(reshape(rw,16,160,[]),2));

figure();
plot(mean(left_avg_before));
hold on;
plot(mean(right_avg_before));
legend('Left','Right');
title('Average waveform before CSP');

figure();
plot(mean(left_avg));
hold on;
plot(mean(right_avg));
legend('Left','Right');
title('Average waveform after CSP');
figure();
```

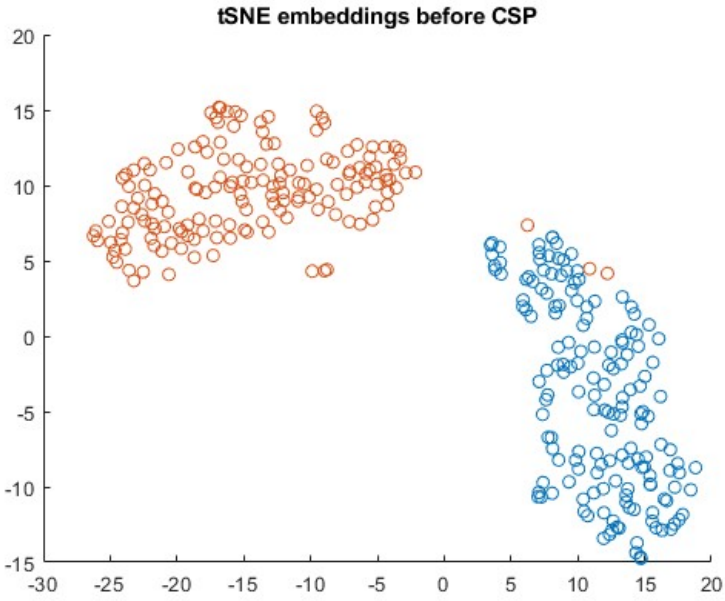

Average waveform before CSP
Left
Right
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0
100
200
300
400
500
600
700
800
900

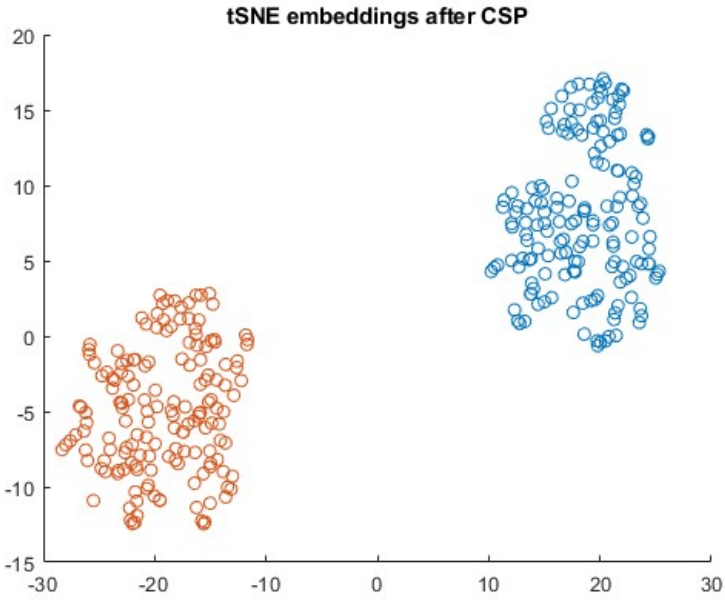

Average waveform after CSP
×10-3
Left
Right
5
4
3
2
1
0
-1
-2
0
100
200
300
400
500
600
700
800
900

Demonstrate Separability via tsne

```
left_std=std(reshape(lw,16,160,[]),0,3)';
right_std=std(reshape(rw,16,160,[]),0,3)';


Data_before_transform = [left_std ; right_std];


tsne_before_transform = tsne(Data_before_transform);
tsne_before_transform = tsne_before_transform';
figure();
scatter(tsne_before_transform(1,1:160),tsne_before_transform(2,1:160));
hold on;
scatter(tsne_before_transform(1,161:320),tsne_before_transform(2,161:320));
title('tSNE embeddings before CSP')
Data_before_transform = [left_std_CSP ;right_std_CSP];


tsne_before_transform = tsne(Data_before_transform);
tsne_before_transform = tsne_before_transform';
figure()
scatter(tsne_before_transform(1,1:160),tsne_before_transform(2,1:160));
hold on;
scatter(tsne_before_transform(1,161:320),tsne_before_transform(2,161:320));
title('tSNE embeddings after CSP');

fprintf("Spacial filtering of the data makes it more linearly separable.\n");

figure();
```

```
Spacial filtering of the data makes it more linearly separable.
```

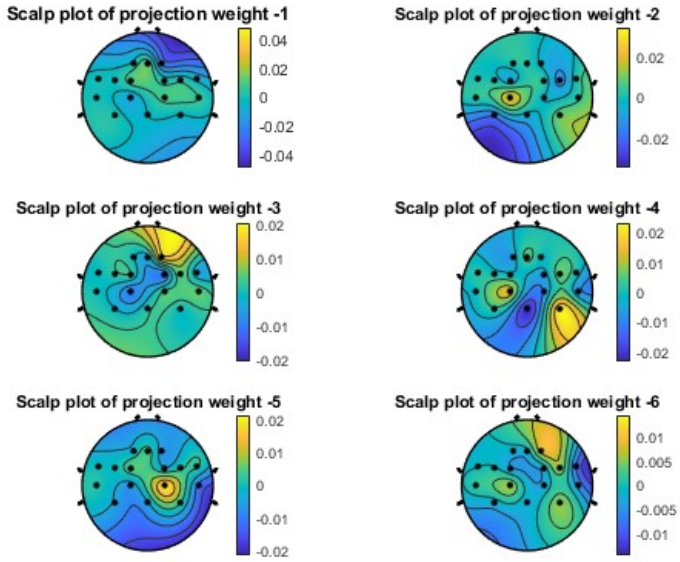

tSNE embeddings before CSP
20
15
10
5
0
-5
-10
-15
-30
-25
-20
-15
-10
-5
0
5
10
15
20


tSNE embeddings after CSP
20
15
10
5
0
-5
-10
-15
-30
-20
-10
0
10
20
30

Plot CSP filters on scalp

```
figure();
for i=1:6
    subplot(3,2,i)
    topoplot(W(:,i),'CSP.locs');
    colorbar();
    title(strcat('Scalp plot of projection weight - ',int2str(i)))
end

fprintf("Benefit of using standard deviation to remove the time dimension as opposed to, say, the mean\n");
fprintf("is that standard deviation prevents outliers from skewing the data distribution.\n");
fprintf("This allows for better separability in the data after CSP filtering.\n");
figure();
```

```
Benefit of using standard deviation to remove the time dimension as opposed to, say, the mean
is that standard deviation prevents outliers from skewing the data distribution.
This allows for better separability in the data after CSP filtering.
```

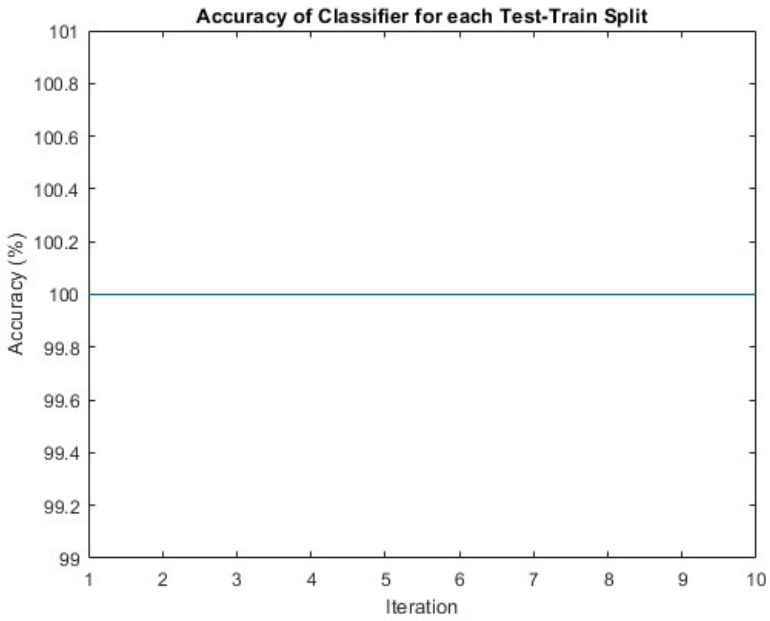


Classify

```
%Shuffle data
left_std_CSP = left_std_CSP(randperm(length(left_std_CSP)),:);
right_std_CSP = right_std_CSP(randperm(length(right_std_CSP)),:);

for i = 1:10
    test_index_vector = 16*i-15:16*i;
    train_index_vector = setdiff(1:160,test_index_vector);
    leftTestData = left_std_CSP(test_index_vector,:,:);
    rightTestData = right_std_CSP(test_index_vector,:,:);
    leftTrainData = left_std_CSP(train_index_vector,:,:);
    rightTrainData = right_std_CSP(train_index_vector,:,:);

    %join left and right (stack)
    testData = [leftTestData; rightTestData];
    trainData = [leftTrainData; rightTrainData];
    labels = [ ones(1,144) zeros(1,144)];

    %fit model
```

```
    LDA = fitcdiscr(trainData,labels');

    %calculate test error

    pred_l = predict(LDA,leftTestData);
    pred_r = predict(LDA,rightTestData);

    error(i) = (sum(pred_l==0)+sum(pred_r==1))/size(testData,1);

end

figure();
plot(1:10, 100*(1-error));
title('Accuracy of Classifier for each Test-Train Split');
xlabel('Iteration');
ylabel('Accuracy (%)');

disp(['The average accuracy over trials is ', num2str(mean(100*(1-error))), '%']);
stdError = std(100*error)/sqrt(10);
disp(['The standard error for the Linear Discriminant Analysis classifier is found to be ',...
    num2str(stdError), '%']);

fprintf("We see that in our case 100%% of the trials are correctly classified\n");
```

```
The average accuracy over trials is 100%
The standard error for the Linear Discriminant Analysis classifier is found to be 0%
We see that in our case 100% of the trials are correctly classified
```



Published with MATLAB® R2019b