BMEG 3330 Final Report

Group 1

CHAN, Cheuk Ka **(1155174356)**

HO, Yu On **(1155175831)**

LAM, Cheuk Sam **(1155174681)**

LEE, Yu Ching **(1155159743)**

WONG, Chee Ching **(1155155836)**

Table of Contents

1 Project Overview 4

1.1 Introduction 4

1.2 Project Design 4

1.3 Design Rationale 4

2 Classifier Training 5

2.1 Raw EEG Data Acquisition 5

2.1.1 Preparation 5

2.1.2 Data Acquisition 6

2.2 EEG Data Pre-processing 6

2.2.1 Data Segmentation 6

2.2.2 Feature Extraction 7

2.3 Classification Algorithm 8

2.3.1 Model Choosing 8

2.3.2 Model Tuning 9

3 Game Playing 10

3.1 Raw EEG Data Acquisition 10

3.1.1 Preparation 10

3.1.2 Data Acquisition 10

3.2 EEG Data Pre-processing 10

3.2.1 Data Concatenation 10

3.2.2 Feature Extraction 10

3.3 Classifier Prediction 10

4 Results 11

5 Encountered Problems 12

5.1 Problem 1: High Input Impedance 12

5.1.1 Problem Description 12

5.1.2 Solution 12
5.2 Problem 2: Difficulty in Concentration for Subject 12
5.2.1 Problem Description 12
5.2.2 Solution 13
6 References 14

1 Project Overview

1.1 Introduction

Brain-computer interface (BCI) systems allow direct communication between the brain and a computer, enabling one to interface with a computer system through brain activities alone without physical interaction. Electroencephalography (EEG) is a popular non-invasive method of recording brain activity by electrodes placed on the subject's scalp.

Our project aims to play a robot racing game consisting of four actions (running, jumping, rotating, and sliding) corresponding to four different EEG patterns. This report introduces the design and implementation of our BCI system to control the racing game's virtual character.

1.2 Project Design

The subject is told to correlate each game action with a motion (calming down, holding a cup with the left hand, holding a cup with the right hand, and jumping) to generate sensorimotor stimulation. EEG signals are collected using an EEG cap. Data collected are pre-processed before classification. The classifier determines the motor imagery of sensorimotor action in each segment and translates it back into an in-game action. The command is then given to the game, and the robot character does the corresponding action.

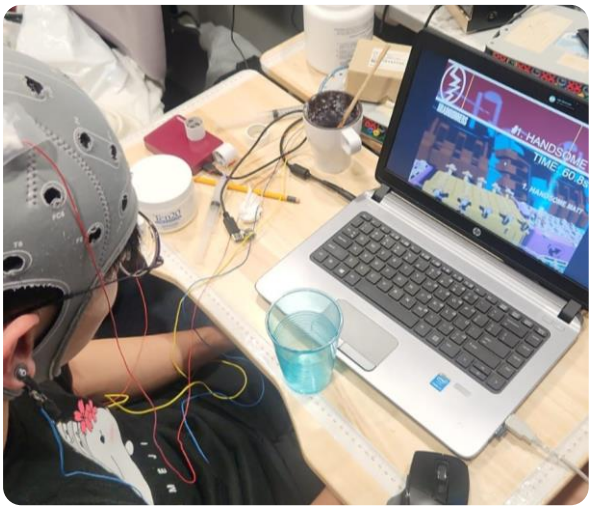

Figure 1
Picture of subject playing the robot game using EEG

1.3 Design Rationale

In our BCI system design, EEG signal is collected at C3 and C4 by the international 10-20 system (See *Figure 2*). These two regions were chosen since they have been proven to be the optimal locations to collect signals related to motor imagery states [1]. C3 and C4 signals correspond to movements on the right side of the body and those on the left, respectively.

Reference electrodes are located on the earlobes of the subject, i.e. A1 and A2, by the 10-20 system (See *Figure 2*).

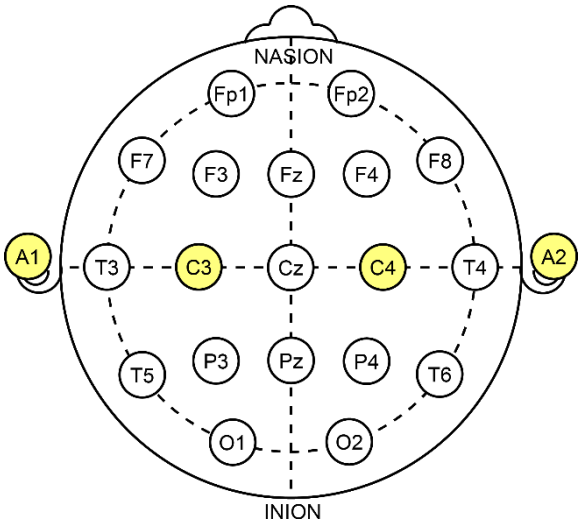

Figure 2
The international 10-20 system with 19 active electrodes; used electrodes highlighted

2 Classifier Training

2.1 Raw EEG Data Acquisition

2.1.1 Preparation

1. The EEG cap is worn on the subject. The position is adjusted by measuring the subject's head's centre position (Cz). Cz is located by intersecting the mid-frontal plane (defined by two vertical lines perpendicular to the ground plane extending from the tragus of both ears), the mid-sagittal plane, and the vertex of the subject's head.
2. The subject's earlobes are cleansed with an exfoliating gel and tissue paper to reduce electrical impedance between the scalp and the electrodes by removing dead skill cells.
3. The cavities of the cup-shaped reference electrodes are filled with conductive paste and attached to the subject's earlobes.
4. The subject's hair is moved using a small-gauge needle to expose the scalp at C3 and C4.
5. Floating electrodes are placed at C3 and C4.
6. Liquid conductive gel is injected in and around the electrodes using the small-gauge needle to improve conduction and reduce input impedance.
7. Tape is used to secure the electrodes.
8. The input impedance is checked using a MATLAB script (`check_impedance.m`). The setup is adjusted until the impedance of both channels is 5kΩ or lower.

2.1.2 Data Acquisition

We used a training data collection script to acquire training data for the classifier. The training was split into 40 segments, with ten segments for each sensorimotor action in a random order. The subject was instructed to imagine each action (calming down, holding a cup with the left hand, holding a cup with the right hand, and jumping) for six seconds when prompted. The script automatically collects the EEG signals from both channels at 256 Hz for the entire acquisition duration and labels it with the segment action.

2.2 EEG Data Pre-processing

2.2.1 Data Segmentation

The raw EEG data was split into segments using the given segment timestamps. Each segment is six seconds long. To remove potential time-related biases and to augment the data, the segments were further divided into smaller sub-segments of two seconds with an offset of 0.0625 seconds per.

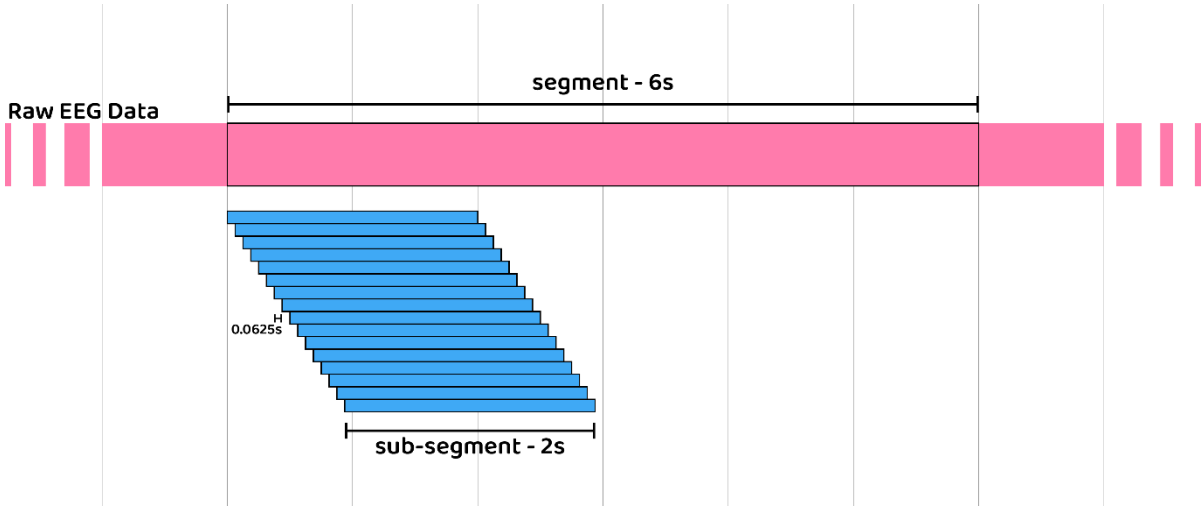

Figure 3
Data segmentation process illustration

Below is the code for sub-segmenting the EEG data — excerpt from `processEEG.m`.

```
1  for s = 1 : subsegment_count
2      subStartSample = startSample + (s - 1) / SUBSEGMENTS_PER_SECOND * SampleRate;
3      subEndSample = subStartSample + SUBSEGMENT_DURATION * SampleRate - 1;
4      subsegments{(n - 1) * subsegment_count + s}.type = Stimulus_Type(n);
```

In lines 2 and 3, the start and end sample point indices are calculated.

In line 4, the extracted sub-segment is added to a cell array for further processing.

Below are the constant values used in the code above — excerpt from `processEEG.m`.

```
SEGMENT_DURATION = 6; % seconds
SUBSEGMENTS_PER_SECOND = 16; % segments
SUBSEGMENT_DURATION = 2; % seconds

subsegment_count = (SEGMENT_DURATION - 1) * SUBSEGMENTS_PER_SECOND + 1;
subsegments = cell(subsegment_count, 1);
```

2.2.2 Feature Extraction

For each sub-segment, the band powers of the sub-α bands (0-8 Hz), α-band (8-12 Hz), high-α band (10-13 Hz), low-β band (12-20 Hz), and β-band (13-30 Hz) for both channels are calculated. Ten frequency-domain features are extracted from each sub-segment.

The α- and low-β bands are selected since they coincide with the sensorimotor-related μ-rhythm and other sensorimotor oscillations; these two bands make up the sensorimotor rhythms. Frequencies higher than 30 Hz are not used since they are highly susceptible to noise from the power source. Other in-between frequency bands are also included to give the classifier more context.

Below is the code for extracting the features — excerpt from `processEEG.m`.

```
for f = 1 : length(BAND_FREQUENCY)
    frequencyBand = BAND_FREQUENCY{f};
    subsegments{(n - 1) * subsegment_count + s}.powers{f}.channels{1} = ...
        bandpower(EEGData(subStartSample : subEndSample, 1), SampleRate, frequencyBand);
    subsegments{(n - 1) * subsegment_count + s}.powers{f}.channels{2} = ...
        bandpower(EEGData(subStartSample : subEndSample, 2), SampleRate, frequencyBand);
end
```

In lines 4 and 6, the band power of each frequency band is computed.

Below are the constant values used in the code above — excerpt from `main.m`.

```
% ~ constants
BAND_FREQUENCY = {[0 8], [8 12], [10 13], [12 20], [13 30]}; % Hz
```

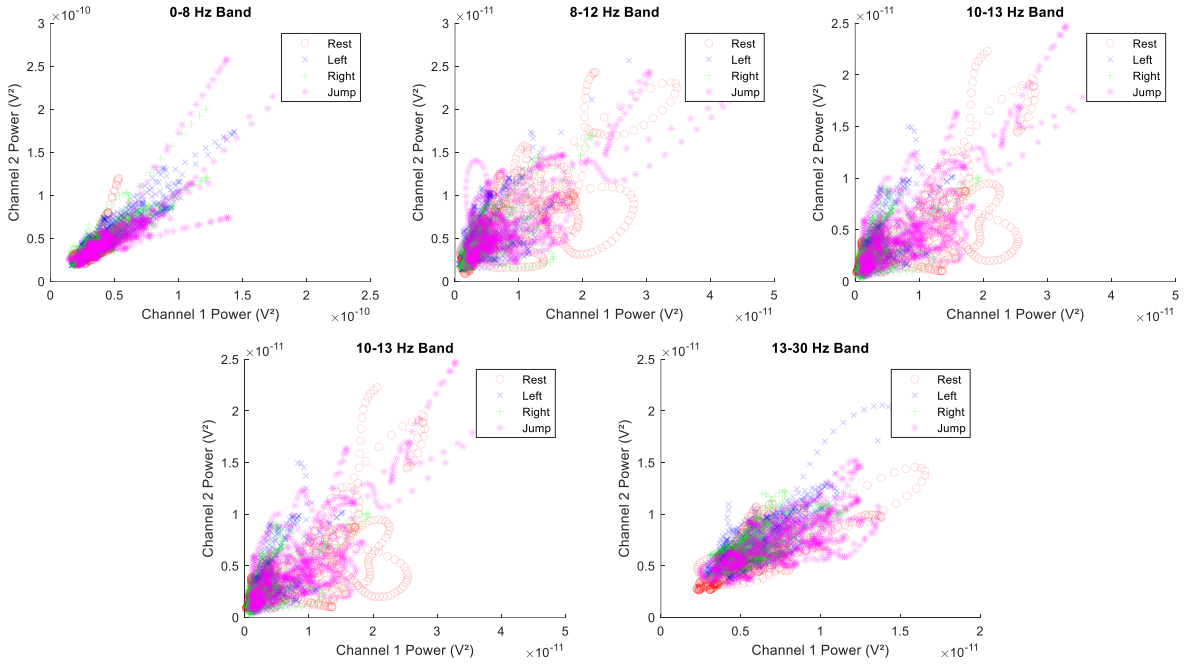

Figure 4
Channel band powers for each featured band

2.3 Classification Algorithm

2.3.1 Model Choosing

We attempted to employ many different families of models to differentiate the classes and chose the one with the best performance. Empirically, the *k*-nearest neighbour (KNN) method performed well most consistently with 30-40% accuracy on unseen data.

KNN is a non-parametric method which classifies an object by a majority vote of its neighbours based on comparing testing data with training data; hence, the object is assigned to the class most common among its *k* nearest neighbours [2]. *k* is the number of nearest neighbours to be considered in selecting the class, where *k* is a positive integer [3]. Since the band power of the same motor imagery task is expected to be similar, the data points of the training data of the same task form a cluster, forming a class.

Below is the code for training and evaluating a model — excerpt from `main.m`.

(Note: Similar code is used for evaluating other model families.)

```
1  %! % ──────────────────────────────── KNN ──────────────────────────────── %
2  fprintf('═══════════════════════════════════════════════════════\n');
3  KNNModel = fitcknn(trainingParameters.X, trainingParameters.y, 'NumNeighbors', 1);
4  trainingParameters.yPredicted_knn = predict(KNNModel, trainingParameters.X);
5  trainingParameters.accuracy_knn = sum(trainingParameters.yPredicted_knn == trainingParameters.y) / length(trainingParameters.y);
6  fprintf('Training accuracy (KNN): %f\n', trainingParameters.accuracy_knn);
7  trainingParameters.confusionMatrix_knn = confusionmat(trainingParameters.y, trainingParameters.yPredicted_knn);
8  fprintf('Training confusion matrix (KNN):\n');
9  disp(trainingParameters.confusionMatrix_knn);
10 testingParameters.yPredicted_knn = predict(KNNModel, testingParameters.X);
11 testingParameters.accuracy_knn = sum(testingParameters.yPredicted_knn == testingParameters.y) / length(testingParameters.y);
12 fprintf('Test accuracy (KNN): %f\n', testingParameters.accuracy_knn);
13 testingParameters.confusionMatrix_knn = confusionmat(testingParameters.y, testingParameters.yPredicted_knn);
14 fprintf('Test confusion matrix (KNN):\n');
15 disp(testingParameters.confusionMatrix_knn);
16 % export model
17 save('model/knnModel.mat', 'KNNModel');
```

In line 3, the KNN model is trained.

In line 10, the KNN is asked to predict the classes of untrained data.

In line 17, the trained model was exported and saved for future use.

During the data acquisition phase in 2.1.2 above, we collected a total of four sets of data. However, we found that since the input impedances and other environmental factors vary significantly between each trial, it is challenging for the classifier to identify the classes correctly. Empirically, we found that using the data collected in trial three as training data showed the best performance in unseen data.

(Note: We also attempted to use a neural network to classify the data; however, it was not fast enough to train and compute a neural network in MATLAB.)

2.3.2 Model Tuning

In KNN, using a moderate neighbour number to smooth out the boundaries is generally advisable [4]. However, we found that increasing the neighbour number empirically worsened the accuracy since the *k* value reduces noise and increases error rate. The final model we used has its neighbour number kept as one.

3 Game Playing

3.1 Raw EEG Data Acquisition

3.1.1 Preparation

Preparation steps are identical to those shown in 2.1.1 above.

3.1.2 Data Acquisition

An accompanying MATLAB script (`Game_start.m`) measures the EEG signals in real-time when playing the game. The script acquires the data from both channels at 256 Hz and calls the classifier function in one-second intervals.

3.2 EEG Data Pre-processing

3.2.1 Data Concatenation

Since the classifier requires data acquired over at least two seconds as input (*See 2.2.1 above*) while the script is called every second, the script stores the data obtained in the last function call. It then concatenates the latest data to construct data spanning two seconds.

Below is the code for extracting the features — excerpt from `your_classifier.m`.

```
1  if ~exist('classifierInitialised', 'var')
2      global classifierInitialised;
3      global classifierModel;
4      global lastSecondData;
5      classifierInitialised = true;
6      classifierModel = load('./training_data/Group1_model/bagTreeModel.mat');
7      classifierModel = struct2cell(classifierModel);
8      classifierModel = classifierModel{1};
9
10 end
11
12 totalData = [lastSecondData; data];
13 lastSecondData = data;
```

In line 4, a global variable is created to store the data acquired in this function call.

In line 12, the latest data is concatenated with data from the previous function call.

3.2.2 Feature Extraction

Features are extracted in the same manner as detailed in 2.2.2 above.

3.3 Classifier Prediction

The extracted features are formatted and given to the classifier for prediction. After predicting a sensorimotor action class, the MATLAB script (`your_classifier.m`) translates it into a game state to instruct the robot to perform different actions. Since the classifier interprets the

classes differently from the robot game, we must first pass the prediction result through an interpreter to translate it into a suitable format.

Below is the code for interpreting the results — excerpt from `your_classifier.m`.

```
1  % Class returns prediction of trained model
2  % 0: Rest
3  % 1: Left Hand
4  % 2: Right Hand
5  % 3: Jump
6
7  Class = predict(classifierModel, X);
8  fprintf('Predicted Class: %d\n', Class);
9
10 switch Class % REST = 0, ROTATE = 11, JUMP = 12, SLIDE = 13
11     case 0
12         state = 0;
13     case 1
14         state = 11;
15     case 2
16         state = 12;
17     case 3
18         state = 13;
19 end
```

In line 7, the classifier issues a prediction result based on the input features.

In lines 10-19, the result is mapped from the classifier format to the robot game one.

4 Results

Tabulated below are the game results.

Run	Time (seconds)
1	109
2	111
3	117
4	118
5	121
Average of the best three	**112**

5 Encountered Problems

5.1 Problem 1: High Input Impedance

5.1.1 Problem Description

There was high input impedance from the electrodes during preparation. High impedance introduces several challenges in EEG data acquisition, such as increased noise and signal attenuation, decreased signal-to-noise ratio, and reduced spatial resolution. These challenges led to poor signal quality, negatively affecting the system's performance.

5.1.2 Solution

To mitigate the negative impacts, proper electrode preparation was practised. We improved the skin-electrode contact and reduced the input impedance by skin cleansing and exfoliation using conductive gel and scrub. Skin cleansing and exfoliation remove the outer layer of dead skin cells, oils, or other contaminants on the skin surface. It promotes better contact between the electrodes and the skin as barriers and other hindering factors have been eliminated. Conductive gel further reduces the input impedance by filling the gap between the electrodes and the skin and enhancing electrical contact. The subject's hair is washed before each data acquisition and competition to remove the oils and dust on the head.

Besides using only tape, we filled the gaps in the wells of the EEG cap with crumbled tissue paper pieces to secure the electrodes further. The anchorage provided limits the movement of the electrodes, therefore minimising the impedance change by sudden motions of the subject.

After practising these measures, the input impedance from the electrodes is significantly reduced from $\sim 30\text{k}\Omega$, to $\sim 15\text{k}\Omega$, even $\leq 5\text{k}\Omega$ in a vast majority of situation.

5.2 Problem 2: Difficulty in Concentration for Subject

5.2.1 Problem Description

In practice, the subject reflected that it is challenging to vividly, quickly, and immediately imagine the prompted action. The resultant sensorimotor signals were weak, indistinct, and inconsistent, which hindered the classifier's performance. Furthermore, it was non-trivial to imagine the same motion every time. This introduced a lot of noise and variations in the training data.

5.2.2 Solution

To combat the trigger latency problem, we split the raw EEG data into sub-segments (*See 2.2.1 above*) such that the sub-segments within the latency period account for only a tiny proportion of the training dataset.

To combat the motion inconsistency problem, we instructed the subject to imagine a more repeatable motion (shaking his left hand, rotating his right arm in a circular motion) instead of imagining the suggested serialised motions (grabbing the cup with the left or right hand). This approach produced more consistent and robust signals for analysis.

6 References

[1] S. Hu, H. Wang, J. Zhang, W. Kong and Y. Cao, "Causality from Cz to C3/C4 or between C3 and C4 revealed by granger causality and new causality during motor imagery," in *2020 IEEE 9th International Power Electronics and Motion Control Conference (IPEMC2020-ECCE Asia)*, Beijing, China, 2014.

[2] T. Rahman, A. K. Ghosh, M. H. Shuvo and M. Rahman, "Mental Stress Recognition using K-Nearest Neighbor (KNN) Classifier on EEG Signals," in *Conference: International Conference on Materials, Electronics & Information Engineering, ICMEIE-2015*, University of Rajshahi, Bangladesh, 2015.

[3] C. W. Yean, W. Khairunizam, M. I. Omar, M. Murugappan, B. S. Zheng, S. A. Bakar, Z. M. Razlan and Z. Ibrahim, "Analysis of The Distance Metrics of KNN Classifier for EEG Signal in Stroke Patients," in *2019 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, Kuching, Malaysia, 2018.

[4] B. S. Everitt, S. Landau, M. Leese and D. Stahl, "Miscellaneous clustering methods," in *Cluster Analysis*, 5th ed., D. J. Balding, N. A. C. Cressie, G. M. Fitzmaurice, H. Goldstein, G. Molenberghs, D. W. Scott, A. F. M. Smith, R. S. Tsay and S. Weisberg, Eds., King's College London, UK, 2011, pp. 215-256.