Investigating Immersion of Environments affecting the performance of BCI

Mahrad Pisheh Var

A thesis submitted for the degree of Master of Science in University of Essex

Supervisor: Prof. Francisco Sepulveda

School of Computer Science and Electronic Engineering

University of Essex

August 2019

Abstract

BCI is a gateway to eliminate any physical peripherals to be controlled by hand and uses the brain signals (EEG) to establish a communication between the computer and users. The project is built around the question “does immersion have a positive impact on BCI?” or “Will VR create immersion and therefore improve the performance of the BCI?”. These two questions are investigated through experiments set to answer these questions logically. Two environments are created in these experiments, one with simple textures and colours and the other with complex textures that look realistic. On the BCI side, SSVEP is used with two classifiers such as SVM and LDA to train the signals gathered from the test subjects. Each classifier is used to classify the brain signals where the subjects are set to interact with both environments. The stimuli are set to 20, 15, 12 and 10 hertz where the subjects are instructed to observe them interact with the environments. This project is created with four different languages; Java, C#, Python and Lua. The software used is Unity and OpenVibe. The code for this project can be found at https://cseegit.essex.ac.uk/ce901/ce901_pisheh_var_m.

Contents

List of Figures

List of tables

Chapter 1 – Introduction

1.1 Motivation

A human brain is capable of using the body's perceptual parts to become aware of its surroundings. Thus, with this capability, it will allow the bearer of the brain to establish connections and interact with its environment. The action taken to interact with the environment will allow the brain to share its thoughts; muscles cause this interaction. It would be possible to bypass the communication between the brain and muscles with brain-computer interfaces. The system will allow the brain to directly communicate with the system specifically created to listen to brain signals. For example, one of the important uses of BCI is allowing a disabled person to move or interact with objects they normally would be unable to interact with. A common suggested way is to allow the user to activate an emergency button; this allows the user to bypass the effort of getting to the emergency button and physically pressing it.

To establish the communication between the human brain and computer, hardware is required. The most common method to measure brain activity is electroencephalography (EEG) [1]. The human brain consists of different cortexes. Each cortex will establish an inner and outer connection with neurons. The EEG contains series of electrodes that measure the mean membrane potential of neuron population [1]. The EEG as discussed in [2] can achieve signals in many different ways, and the most common way is electrodes attached to scalp. In the past this hardware have been used for medical purposes such as diagnosis of epilepsy and sleep disorders. [2] mentions that in medical studies, it has been discovered that certain EEG activities happen before the beginning of seizure. This data was only discovered by automatic classification of data gathered through EEG. This information allows us to understand that the brain establishes these EEG activities prior to any seizures that have occurred; this can happen with any other brain activity or physical activity. One of the other uses of automatic classification is to take control of humanoid robot mentioned by [3]. One of the most common products has been seen on television, and in documentaries controlling exoskeletons, mainly used for disabled individuals [4].

1.2 Project Description

The project as it is mentioned in [5], creates two different environments and use Enobio 20 [6], to establish a connection with user's EEG activities and the environment, these activities were used mostly to control lights and water taps.

The project has two sides that were designed and programmed with four different languages (Java, C#, Lua, and Python). These sections are:

1- Environment side

2- BCI side

In the environment side, one environment is created with simple textures and uses few numbers of polygons in the creation of objects. The simplicity of the environment is set to be the least comprehensive, the user will have a complete understanding of their environment. The other environment is created with more complex textures and used a higher number of polygons. This environment has thorough architecture and lights located and positioned to bring realism to the environment. The complex environment has seven different sets of mood which will be explained further in the implementation.

In each environment, a set of stimuli was used to trigger EEG activities in the recording of the signals.

In the BCI side of the project, OpenVibe [7] was used to create seven different scenarios to prosses the data and allow the signals to be classified and trained and to be processed for real-time classification and be used to establish a connection with our environment.

1.3 Goal

The goal for this research is to understand the BCI further and observe brain activities in interaction with the environments created. There are many observations made and recorded in this project.

The performance of BCI was tested with the list below:

1- 3D simple environment.

2- 3D complex environment.

3- 2D stimulus with only Black Background.

4- VR integration to our environments.

5- Two ways to flicker our stimulus (during the observations it was noticed in a 3D environment, machines can have different frames per second to visualise the stimulus and flicker them. Therefore, a Jar file was written for the C# written program to control the Jar file to have a completely different thread in GPU dedicated to the Jar file).

The project has different features that were implemented and will go over in the design and implementation chapter:

1- Two environments, one simple and one complex that is similar and built with the same architecture.

2- A complete BCI system to acquire the signals process, measure the performance and output of the stimulations.

3- An automatic test that records the data which calculates their performance based on the recorded data.

1.4 Dissertation structure

This dissertation consists of 6 chapters starting with the Introduction to chapters:

- Chapter 2: Literature review; reviewing the existing literature related to the project, and further explanation of terms and concepts that are used in this project.
- Chapter 3: Requirements and Design; contains all the information about the program structure, and BCI designs made in the OpenVibe scenario.
- Chapter 4: Implementation; further testing in how the environments were made and programmed and how each scenario in OpenVibe was created and scripted.
- Chapter 5: Results and Discussion; illustrate and explains all the results received from the five

tests.

- Chapter 6: Conclusion; concludes the project and further recommendations for the project.

Chapter 2 - Literature review

2.1 What is BCI?

Brain-Computer Interface is a system that independently help severely disabled people. The aid is applied by converting their intentions from the signal form into machine-readable commands [8]. The electroencephalogram EEG is commonly used because of its safety and high time resolution [8]. BCI can be described as a peripheral that allows users to eliminate the usage of the manual controller and directly use the brain signals or "intent" to interact with their environment. The philosophy behind these studies originates from helping disabled people who are not able to perform desired actions.

Figure 1 illustrates a BCI system architecture; the system starts with signals to be acquired from the subject, once the data is acquired, each signal will go through feature extraction, the extracted features will be processed by the classifier and output will be decided.

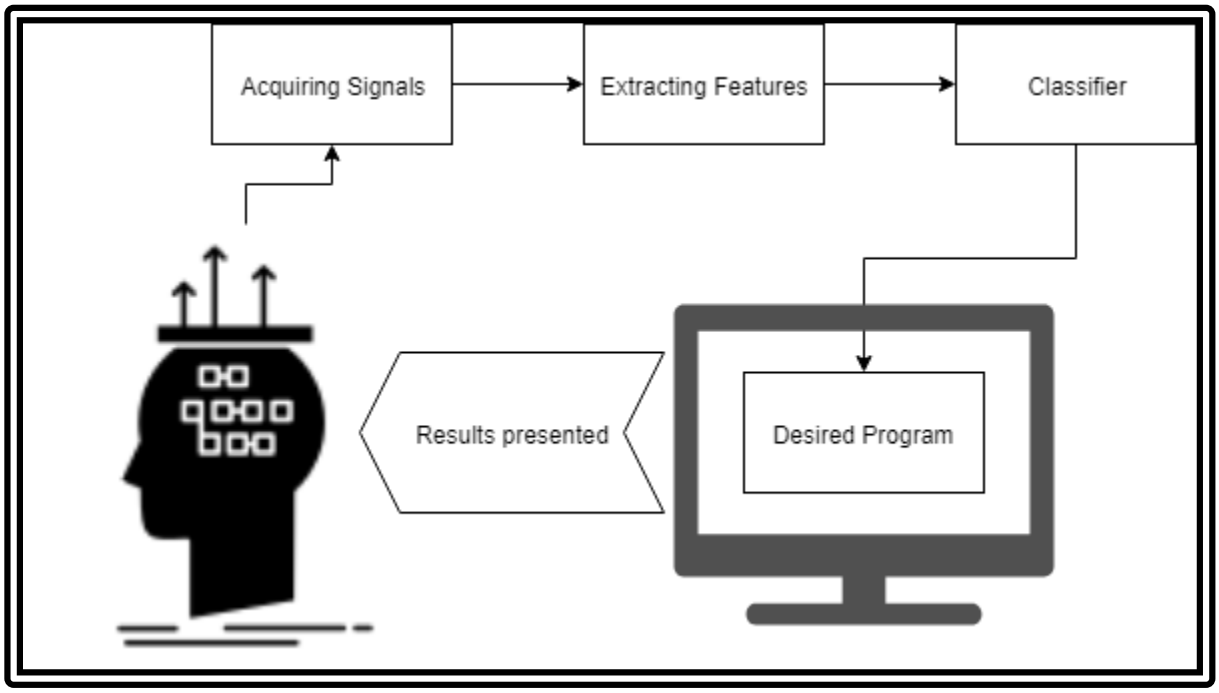

FIGURE 1 AN OVERVIEW OF THE BCI SYSTEM

2.2 Classification in BCI

One of the core parts of the BCI system is its classification unit. The classification unit will receive all the features extracted by the EEG and translates into machine-readable commands. In the classification, according to [9] some properties need some level of consideration.

[9] mentions six different properties of the BCI systems that must be considered in classification:

1- EEG Noise that exists naturally.

2- Outliers caused by poor signal to noise ratio.

3- Training data size, in the case of acquiring signals to train, these signals are gathered in a real-time pace, which can be very time-consuming. It will not be possible to have a test subject sit and train our data specifically for SSVEP where they must stare at a flickering light.

4- The volatility of signals, some EEG signals often change over time; these changes cause the trained classifier to perform poorly in the future.

5- Time information, this information is gathered over time; therefore, in some cases, the data is appended to the last data.

6- Features ought to suffer from high dimensionality.

2.3 EEG

For signal acquisition, EEG or ElectroEncephaloGraphy is commonly used, other methods such as MRI and MEG is used in a wider and more technical range. Drytrodes, as mentioned in [5], are commonly used in Enobio 20 [6], but other electrodes can be placed on the subject's head, and the signals are monitored. A different method to dry electrodes, is a conductive gel to be applied to the electrodes which allow the signals to gather higher strength. As mentioned, EEG is used to measure signals and which signals represent the electrical activity occurring in the brain. The brain consists of many neurons and each neuron establishes a connection by electrical discharges. Each discharge represents a communication occurring in the brain. These electrical discharges are not picked up individually by the electrodes. Therefore, a group of neurons must discharge together and be synchronized. Therefore, the number of active neurons and how much they are synchronized will affects the signal's strength [10].

[11] and [12] mention two disadvantages of using non-invasive EEG, these include using the electrodes on the scalp. [11] claims that non-invasive EEGs consist of poor spatial resolution, and most of the brain activities occur at a deeper level of the brain and the non-invasive EEGs cannot measure their activities. Further, [12] claims that signals get distorted by the skull, and sometimes EEG is biased in certain neuron types.

2.4 What is SSVEP?

"SSVEP or Steady-State Visual Evoked Potential is a resonance phenomenon" [13], this phenomenon causes a larger amplitude to be oscillated by the system, and this compared to the force applied at other frequencies is a lot larger. "The phenomenon only occurs when the frequency at which a force is periodically applied is equal or nearly equal to one of the natural frequencies of the system on which it acts" [13].

In BCI, the resonance phenomenon is checked and observed when the subject is set to look at a light source with a flickering frequency that stays the same throughout the observation, the occipital and parietal lobe of the brain are the main areas to check these signals that highlight resonance phenomenon [8]. One of the reasons SSVEPs are commonly used is because of its robustness and high signal to noise ratio (SNR) [8].

2.5 What is Stimuli?

When an organ or tissue is stimulated by an event or an object the stimulation evokes the organ or tissue to respond with a reaction [14]. In SSVEP, there are different ways to create stimuli and evoke the brain to respond.

2.6 How to SSVEP?

[15] discusses four steps in implementing the steady-state visual-evoked potentials in the BCI system:

1- Configuration step.

2- Acquisition of training data step.

3- Training the classifier step.

4- Online testing step.

In the configuration step, the experiment must first set up the peripheral settings; this setting will allow the program to understand the display's refresh rate which will limit the simulation based on the frame per second of the monitor.

"In the SSVEP-based BCI experiments need to reproduce a flickering stimulus at a constant frequency" [15]. The training and testing of the data must follow this rule, and the flickering object must be presented in a shape that holds a constant frequency to change colour [15].

The interval time between each flicker must be measured with high precision.

To be more in-depth about electrical screens and refresh rates, a 60 HZ display is a common refresh rate that is used in most of the displays, in some gaming displays the refresh rate goes up to 144 HZ. In 60 HZ displays, frequencies in the range of 30, 20, 15, 12, 10 or maybe lower are possible to be displayed. Comparatively, a 50 HZ will only display 25, 16.66, 12.5 and 10 or lower frequencies [15].

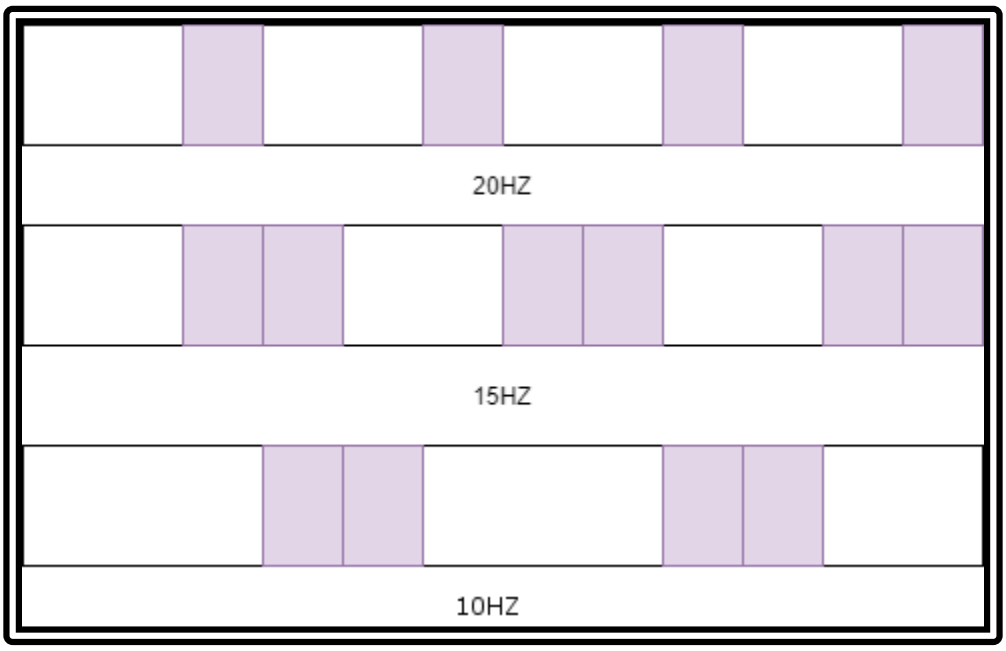

FIGURE 2 STIMULATION FREQUENCIES ON A 60HZ MONITOR, EACH COLOUR REPRESENTS A CHANGE IN COLOUR IN THE TIMELINE OF THE EXPERIMENT

In the training step, the subject must go through a simulation where each occipital area of the scalp will be recorded, and signals will be acquired at a specified time.

In the simulation, three squares must be presented with the additional non-flickering square, which indicates the no stimulation state. Each square must flicker in a predefined frequency, and each square must be focused separately by the subject [15].

In the training of the classifiers, two different trainings must be done.

1- CSP training that highlights the best characteristics of the data.

2- Classifier training that will learn from the best characteristics chosen by CSP training.

The online testing step is mentioned by [15], where real-time signals must be pre-processed and sent to the classifier, the streamed matrix must be processed before it is sent through a port to another program.

2.4 What is virtual reality?

An act of simulation that mimics the physical presence only possible by simulated environments created by computers. The feeling of physical presence is simulated by creating a world where its visualisation holds features taken from the real environment [16].

One of the goals of this project is to create immersion, [16] experiments to create immersive environments, the point of the experiment is to compare the performance of humans on a list of tasks in a real physical environment and immersive virtual environment. The experiment in [16] was done by creating an identical 3D model of an environment; the environment was set to have different sets of lighting. [16] introduces four different lighting setting with a permanent natural light emitted through a window:

1- No light emitted.

2- Two light bulbs turned on.

3- Four light bulbs turned on.

4- Six light bulbs turned on.

[17] strengthens [16] argument, [17] claims humans' performance is influenced by colour and illuminance level on cognitive tasks or interpersonal behaviour. In the results, [16] claimed that most of the participants felt natural with immersive environments, and the p-value indicated that tasks were completed by keeping the performance similar to the real physical environment.

2.5 State of Art

When it comes to the EEG data acquisition, it is always a question whether to use dry-EEG or wet-EEG. A dry-EEG will use drytrodes that does not need any gel to be added to the electrodes and wet-EEG requires the gel to be added. In a study made by [18], [18] showed that wet-EEG had high performance in comparison to dry-EEG. The experiment was set in locomotion, and the accuracy was measured for standing and different speeds at walking. [18] claims wet-EEG obtained higher accuracy by 4 percent in standing state and 10 percent in the walking state. During the walking state, a 32-channel dry-EEG was used, the SSVEP stimuli are fixated on 11 and 12 hertz. In this project, we are unable to use any wet-EEG. Therefore, a 4 to 10 percent in accuracy must be taken into account as dry-EEGs may affect the results. Comparatively, triggering a lower limb exoskeleton, a study carried out in [19] paper, [19]'s aim in their paper was to use a 20-channel dry-EEG headset to assemble a system for motor imagery BCI to have the asynchronous attribute. [19] claims that the dry-wireless EEG may underperform by a small amount, but the setup-time for these headsets are incomparable to wet-EEGs. Therefore, it will save time in process of experimenting and implementing at the same time.

Aside from the hardware side of BCI, it is important how the data is trained and what type of classifiers are used. One of the common approaches in classification is deep learning. Moreover, in BCI applications this method is applied commonly, one of the papers related to motor imagery which also involves classification of SSVEP signals uses deep learning with convolutional neural networks (CNN) in EEG mapping and end-to-end EEG analysis, [20] claims that their results have shown that machine learning has reached a good performance in comparison to other states of art.

In contrast, Paradigms list below was introduced by [21] to be used in a Convolutional Neural Network (CNN) model:

1- Sensorimotor Rhythm.

2- P300 Event-Related Potential.

3- Error-Related Negativity.

4- Movement-Related Cortical Potential.

[21] also used wet-EEG for data acquisition and introduced a method called EEGNet.

[21] states, each data is pre-processed before training. In the pre-processing, both deep and shallow CNN was used across and within-subject classification. In the results, [21] claims for each approach paradigms explained performed differently, and there is an in-conclusion for this matter.

Similar to [20], [22] used five stimuli which visualised as LEDs, each LED was controlled to have different frequencies in flickering. [22] paper states that the project aimed to use a visual stimulus generator to command an exoskeleton. [22] states that in the SSVEP signal measurement, a set of eight wet-EEG was used. In the implementation, [22] states a comparison was made with three different neural networks (NN), these proposed NN are:

1- Three-layer network (CNN-1).

2- Four-layer network (CNN-2).

3- Fully-connected NN.

These comparisons were possible with measuring by Canonical Correlation Analysis (CCA), Canonical Correlation Analysis with k-Nearest Neighbours (CCA-KNN), Multivariate Synchronization Index (MSI). In [22]'s paper, it is mentioned that real-time data was not used. Instead, a set of pre-processed data was proposed. [22] states the CNN-1 method achieved best accuracy in comparison to other NNs.

A five-class SSVEP signal has a classification issue where the author [23] stated that the classification was made possible by combining deep learning and traditional machine learning. [23] used datasets from [24]. Five flickering stimuli frequencies made the dataset. Each signal was gathered by traditional wet-EEG. [23] proposed to compare deep learning methods against classifiers such as Multi-layer Perceptron (MLP), decision trees, k-Nearest Neighbour (KNN) and Support Vector Machine (SVM). [23] states the deep learning methods include, CNN and Recurrent Neural Network (RNN) with Long-Short Term Memory (LSTM). [23] claims CNN performance had the highest performance in comparison to other approaches, the mean accuracy achieved was 69.03%. Comparatively, SVM achieved highest overall accuracy.

On the higher level of implementation of BCI, virtual reality is combined with BCI for any real-time experiments and simulations. [25] states that they have demonstrated BCI with VR by implementing a smart home application in a virtual reality environment. [25] states the environment was made in extreme VR (XVR) and a BCI with P300 system was connected to the environment. [25] claims that it made possible for the subjects to interact with the environment like switching lights through BCI. In [25] experiment, 8, 4, and two flashes of stimuli were used. The experiment for each flash was timed, and accuracy was measured. [25] claims that most subjects were successful in commanding the program and achieved 79 percent accuracy for eight flashes. In contrast, the experiment showed four flashes achieved 70 percent accuracy, and two flashes had the lowest accuracy of 54 percent. [25] suggested that the P300 based BCI system is the most optimal method to control a smart home application.

Similar to [25], [26] used the game engine Unity [27] with BCI2000 to control the environment. [25] states that they aimed to demonstrate the validity of controls. The controls are directed to manipulate the unity-based VR environment. [25] showed the difference in using VR and normal screen and its effect on

the performance of BCI classification. [25] states that the performance did not change during Va R tasks on the naïve level, but it had small increase in the adept level of experiment.

Chapter 3 – Design

3.1 Overall system

The system includes main components such as:

1- Environment.

2- BCI.

3- Subjects.

3.2 BCI side design

The BCI side of the system uses the OpenVibe [7] to create scenarios and manipulate and acquis signals. Each subject will be put through training by BCI first before they are set to interact with the environment. Further, in the process, the communication is made between BCI system and the environment system which the subject will use the BCI system to communicate with the environment. The Environment is made by Unity [27] which two environments are created, the two environments share the same architecture and resemble each other. The complex environment uses Unity's high-resolution assets taken from the asset store. The simple environment uses Unity's low polygonal shapes which creates a comprehensive object for the subject to distinguish shapes and the appliance they are intended to be similar.

Figure 3 illustrates the BCI side of the system, each subject will first go through acquisition testing, where the connection is tested, and channels will be observed. In the next step, the signals will be required twice, once for training the classifiers and the other will be used for testing the classifiers. After the data is gathered for training, a comma spatial pattern training is taken place to create configuration files especially for target separators and dividing frequencies for training the classifiers separately for each frequency chosen for buttons to flicker. The configuration files will be used in classifier training which will use LDA and SVM to classify the data gathered. An optional scenario was made to measure the performance of each classifier and illustrate a confusion matrix for further analysing. In the final stage and scenario, the main processor tends to use TCP to get connected to Unity [27] and sends the stimulations predicted.

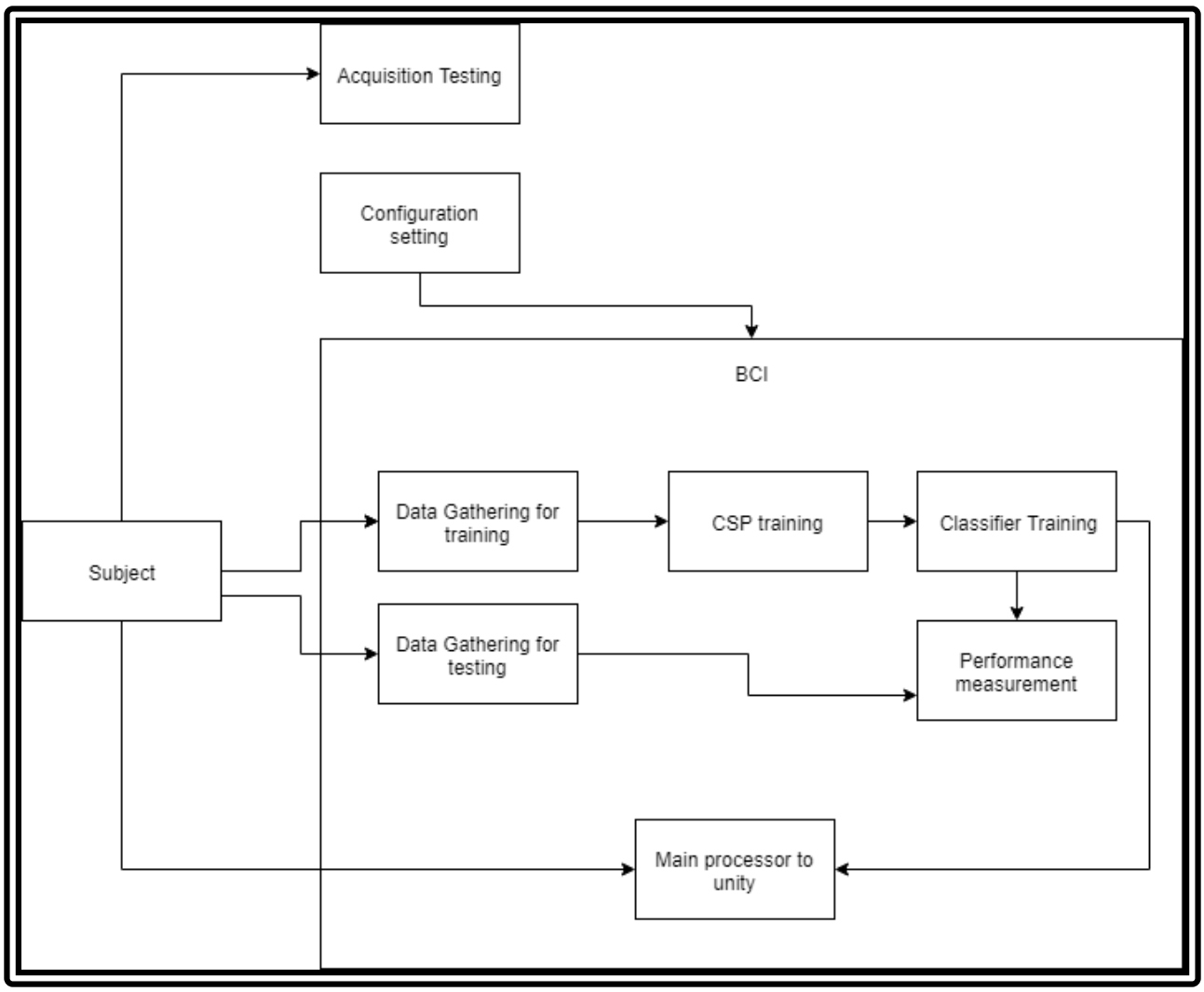

FIGURE 3 BCI SIDE OF THE SYSTEM

3.3 environment side design

There are two environments created to test BCI. The two environments follow the same blueprint in the placement of furniture and kitchen appliances. Figure 4 illustrates the blueprint made for this project.

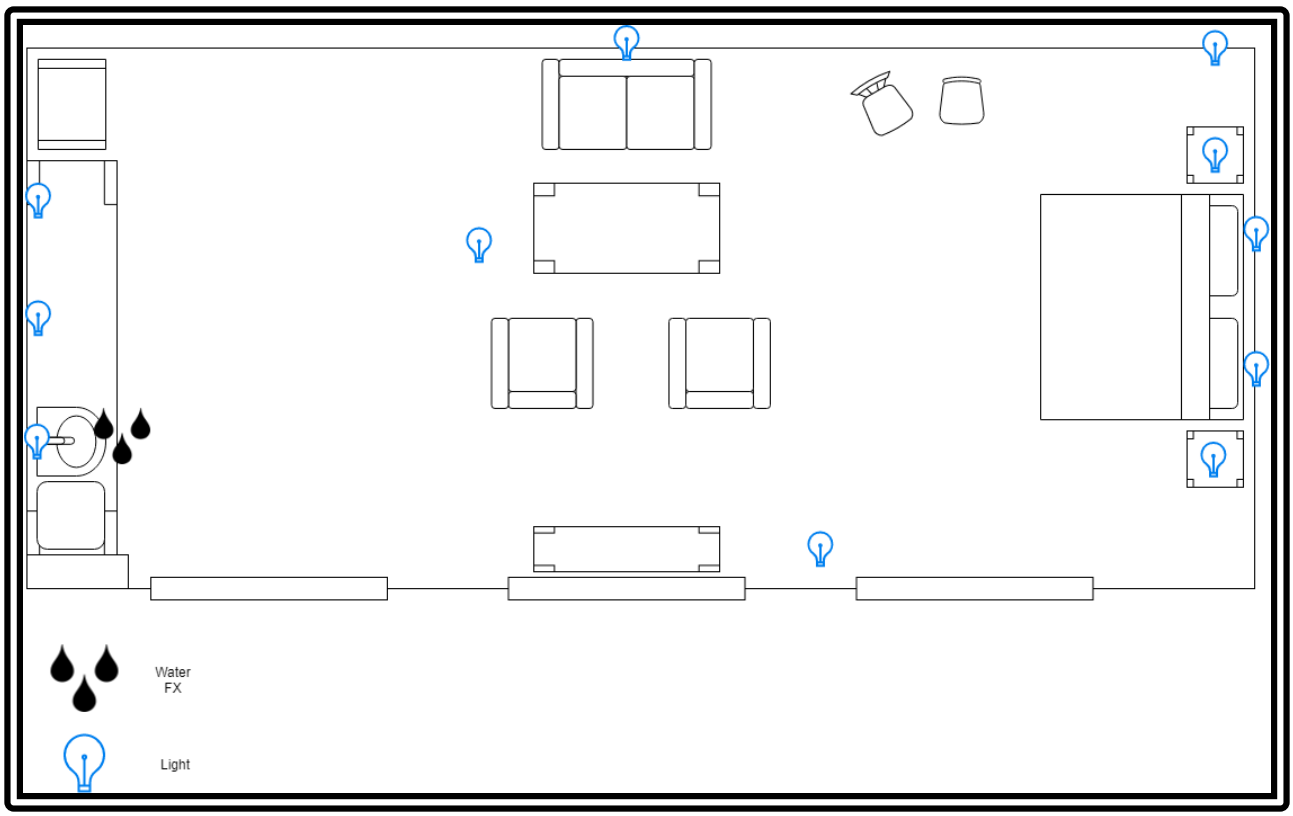

FIGURE 4 BLUEPRINT OF THE ENVIRONMENT

The simple environment uses the simplistic low resolution and a low number of edges in polygon shapes used in the environment.

The appliances used in this environment was taken from Unity's asset store; the name of the asset is "Simple Home Stuff" [28].

Figures 5 and 6 use the assets from [28] and designed in a simple environment. As shown, it is intended to apply simplistic designs to the environment and eliminate any clutter.

FIGURE 5 SIMPLE HOME STUFF [28] LIGHT MODEL AS IT WAS USED IN THE SIMPLE ENVIRONMENT.

FIGURE 6 SIMPLE HOME STUFF [28], KITCHEN APPLIANCES MODELS AS IT WAS USED IN THE SIMPLE ENVIRONMENT.

The complex environment will have more consideration in creating and choosing the textures. The textures chosen will have high resolution, and the object shapes are created with a high number of edges.

The number of assets used is highly numbered; therefore, they are listed in Table 1.

TABLE 1 COMPLEX ENVIRONMENT'S ASSETS

ID	Name of the Asset	Author
1	Water Fx Particles	All VR Education [29]
2	Croissants Pack	Optimesh [30]
3	ArchVizPRO Interior Vol.2	ARCHVISPRO [31]
4	Kitchen Creation Kit	Studio Krokidana [32]
5	Table with chairs x3 Free	3DIZ-ART [33]
6	Realistic Furniture and Interior Props Pack	Sevastian Merevoy [34]

Comparatively to Figures 5 and 6, Figures 7 and 8 represent the same corner but from the complex environment.

FIGURE 7 COMPLEX ENVIRONMENT'S LAMP COMPARED TO FIGURE 5

FIGURE 8 COMPLEX ENVIRONMENT'S KITCHEN COMPARISON MADE TO FIGURE 6

As seen in Figures 7 and 8, a softer touch of light is added to create a warm and immersive area. Several different textures are used for different walls and floor, and each appliance holds a different texture. It is intended the shadows are baked on the surfaces which will create more understanding of shapes.

Figure 9 illustrates the system that was used in the presentation and background processes of the environments. The subjects are intended to interact with the created environments separately, and each is powered by the Hover_UI system where most main background processes of SSVEP is occurring in this component. The process controls each stimulus to move and flicker. The SSVEP unit controls each button, and the system has made it possible for buttons to be easily modified in the unity editor. Several mood lightings are added to the environment to represent and mimic the real-life natural lights such as morning,

afternoon and evening lighting.

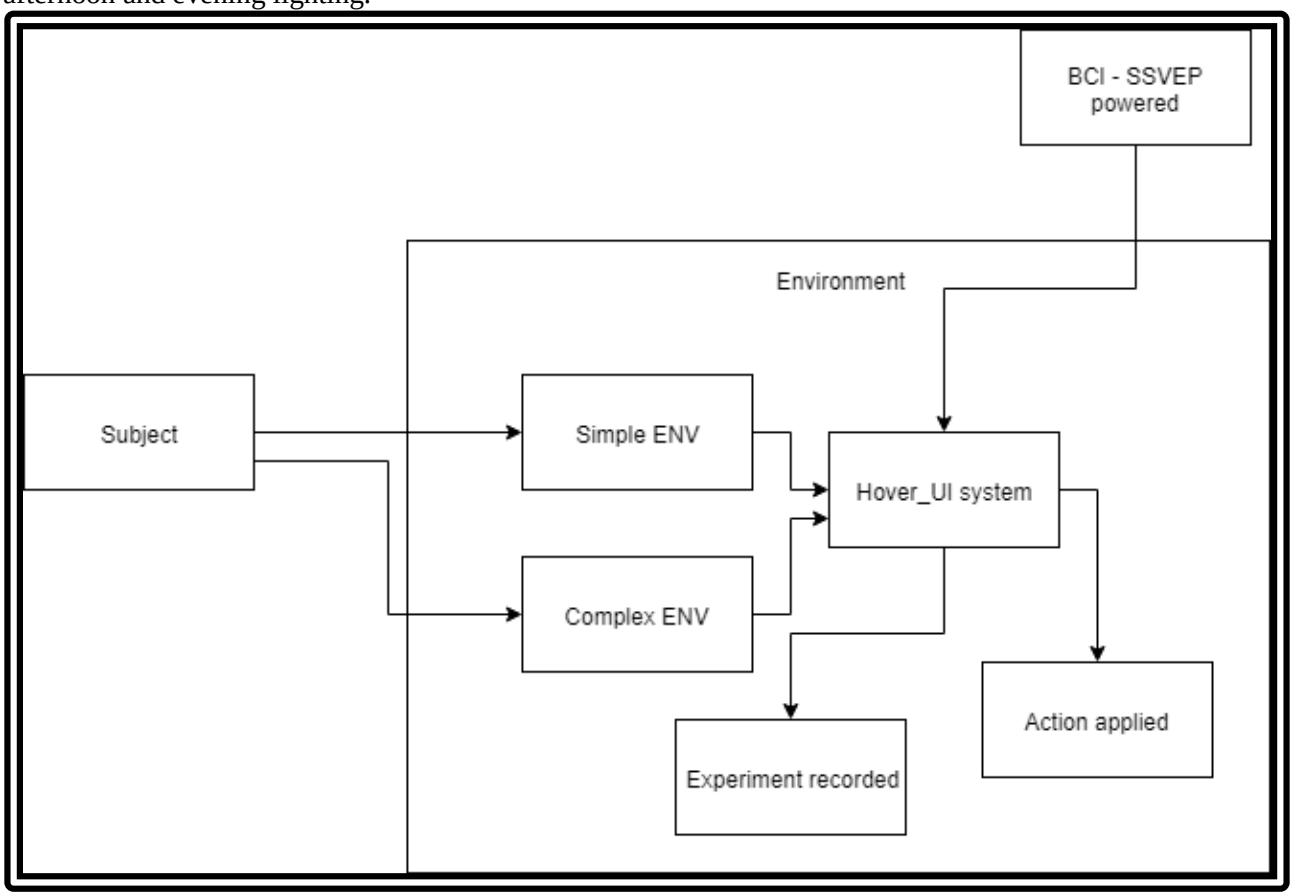

FIGURE 9 ENVIRONMENTS SSVEP COOPERATED SYSTEM

Figure 8 shows that under the *Buttons* option, there is a size variable that is serialized and maximum of four different buttons can be added. Each button is customizable in placement, action it will do if triggered, the frequency it will flicker, and the text presented on the button.

The Hover_UI also uses another system, where is the experiment system that will record and instruct users to go through tests to measure the BCI system performance with the environment chosen. It is intended for the actions for each button to be triggered if the stimulus code of that button matches the BCI system's predicted stimulus.

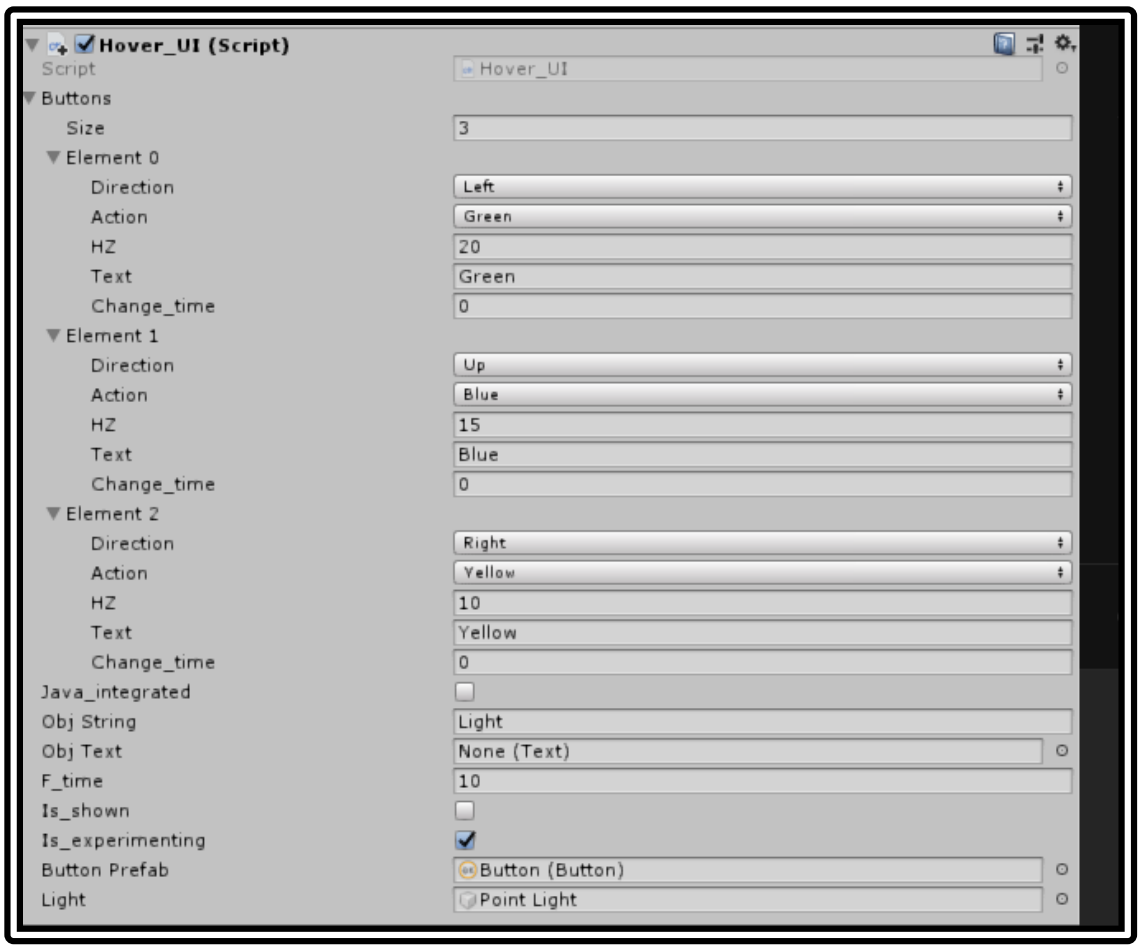

FIGURE 10 HOVER_UI'S EDITOR

3.4 Testing design

To answer the research questions mentioned in the proposal [5], a set of tests must be designed to introduce an analysis to these questions to answer them. A brief from the research questions can be explained in the list below:

1- Does an immersive environment improve BCI SSVEP pattern detection?

2- Does VR create that immersion to improve classification?

To address these questions, a test must be designed to cover these questions and provide an analysis to be concluded into an answer. Figure 11 illustrates the testing; there are two results. The first results represent the experiment done on the offline data where it was previously gathered from the subjects, and the classifier's confusion matrix and accuracy are measured. The second results are gathered online and from the environments directly, which will measure the performance of the classifier by measuring the time and number of maximum tries until the classifier got the desired stimuli frequency correctly. The second results are intended to experiment with a normal screen display and VR in both a simple and complex environment. The classifiers used are Support Vector Machine (SVM) and Linear Discriminant Analysis (LDA).

The table 2 below shows how the data will be represented for the second experiment results.

TABLE 2 COMPLEX ENVIRONMENT'S ASSETS.

Subject No. X	Button frequency (HZ)	Successful tries	Total No. tries	Time Taken	Classifier
VR/Display	(20:10) HZ	Number of predictions that were correct during the stimulation	Total number of predictions made for that particular frequency	(duration it took to succeed or fail)	SVM/LDA

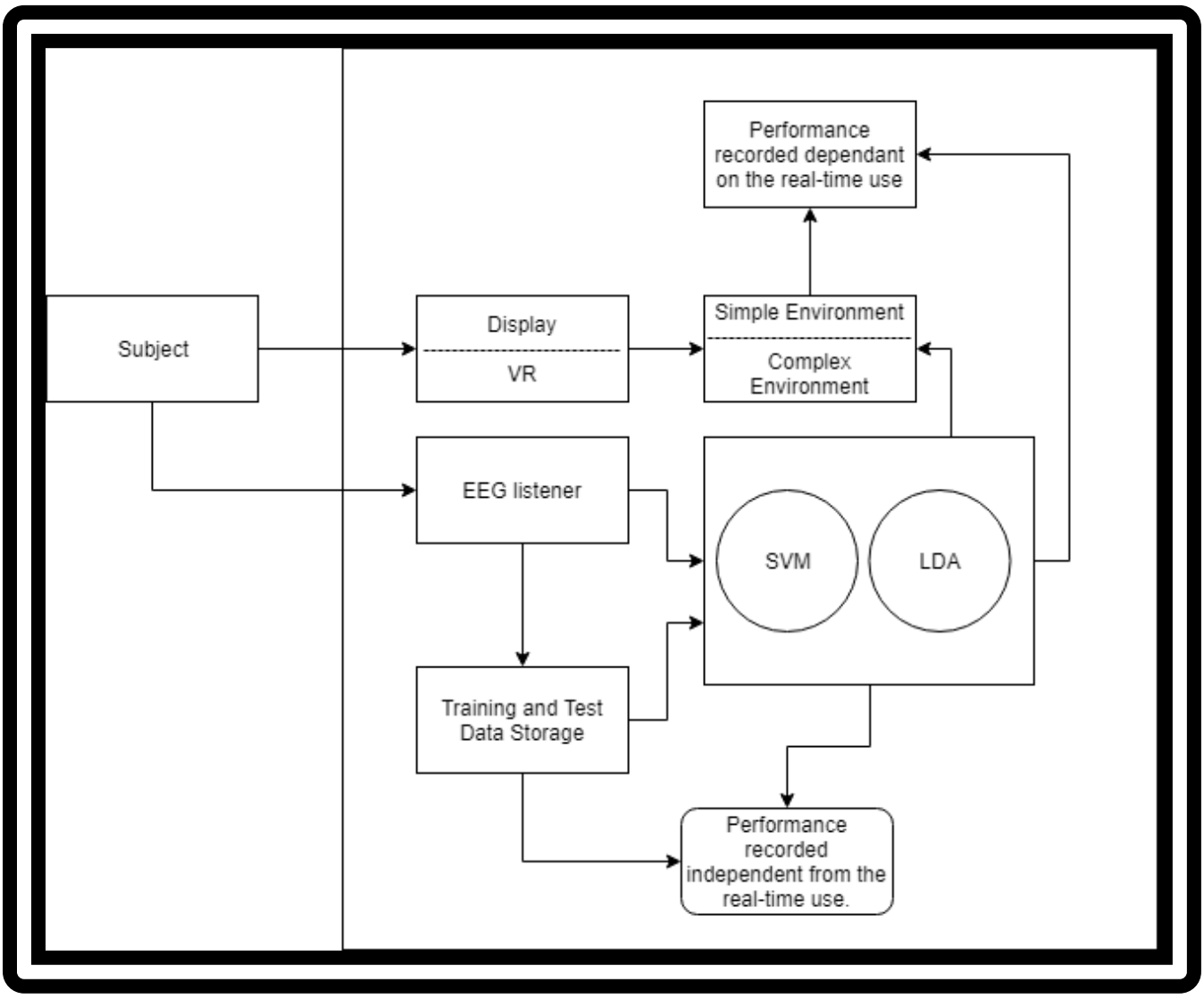

FIGURE 11 DESIGN OF TESTING COVERING THE RESEARCH QUESTIONS PROVIDED IN [5].

Chapter 4 – Implementation and testing

The project was made possible by using the Unity game engine and OpenVibe, and the project is written in 4 different programming languages such as:

1- Java

2- Python

3- Lua

4- C#

The usage of each language is mentioned in each section of the implementation.

4.1 BCI-side implementation

As it was explained in the Design chapter, the BCI-side of the program is inspired by [15]'s methodology to perform SSVEP-based BCI. In OpenVibe [7], seven different scenarios are made where each will perform a key task for this unit.

These scenarios created in OpenVibe:

1- Acquisition testing.

2- Configuration setting.

3- Data acquisition.

4- CSP training.

5- Classifier training.

6- Performance measurement.

7- Main process (online testing).

4.1.1 Acquisition testing

Acquisition testing is presented in Figure 12. This scenario will be used before running any other scenario. This scenario will first establish a connection with the client or user and will illustrate the signals in a raw form, and it also will perform spectral analysis where it will analyse the spectrum of frequencies.

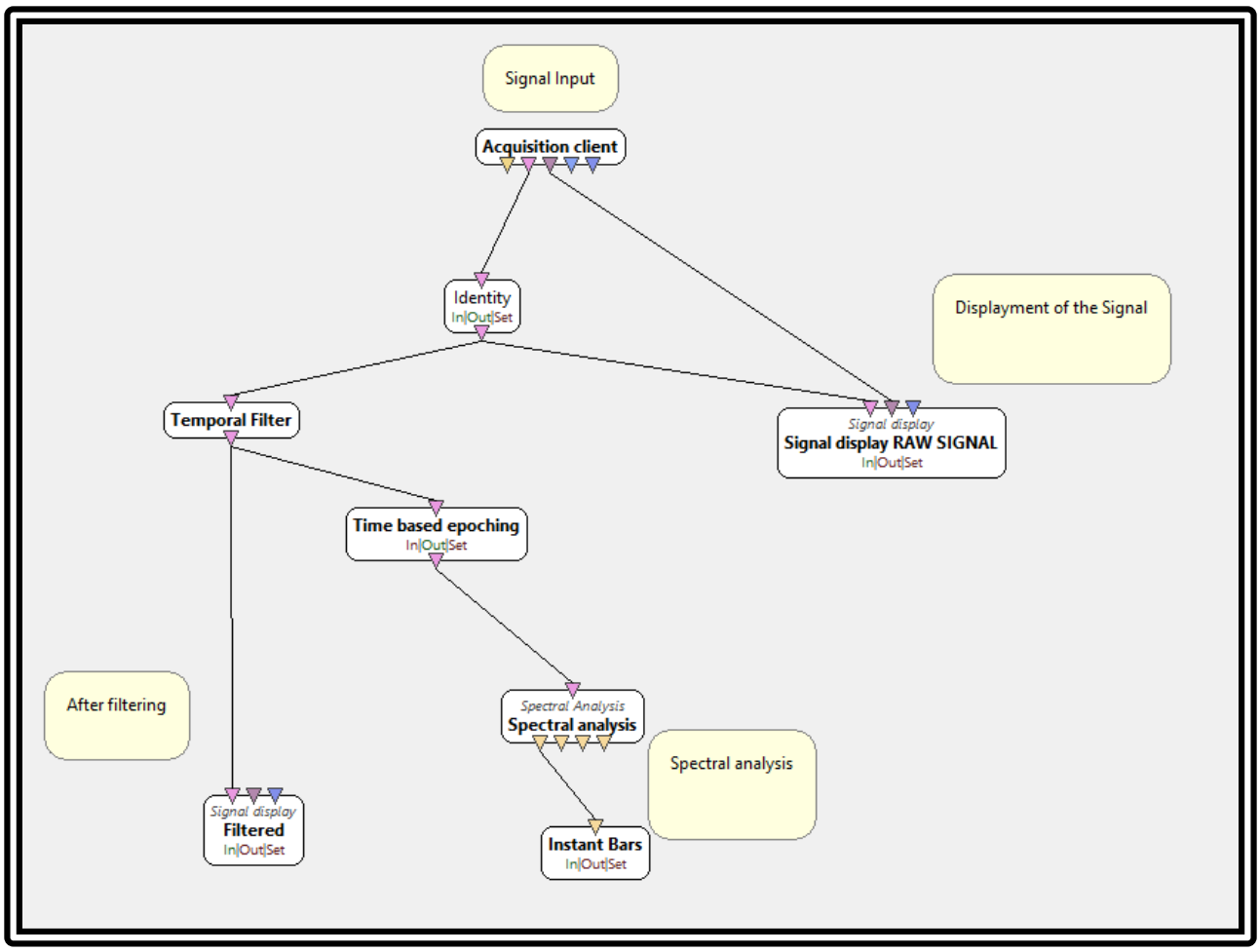

FIGURE 12 ACQUISITION TESTING SCENARIO

4.1.2 Configuration setting

Before any training and testing, settings must be configured. These configurations are related to screen refresh rate and SSVEP stimulus behavioural configuration where are scripted in "Monitor_settings.lua" and "Experiment_settings.lua" respectively.

Monitor settings script will pass the inputted refresh rate of the display and stores the inputted value in a "config" type of file named *SSVEP_ScreenRefreshRate*. The value is written will be stored as:

$$< SSVEP_{ScreenRefreshRate} = Screen_{RF} >$$

Experiment setting script as it is illustrated in figure 13 will have attributes such as:

1- Target Light and Dark Colour: indicates the two colours used to produce the flickering of the targets. In some cases, it may be used dark, and light is arbitrary, but in some cases, more frames will be used to represent the light colour. This occurs when there is an odd number of frames in the target's frequency.

2- Stimulation Frequencies: holds a comma-separated row of frequencies, each will be assigned to the targets.

3- Processing Epoch Duration: as for any AI classifiers Epoch must be defined, and processing epoch duration holds the size of the slice of the signal.

4- Processing Epoch interval: the value will indicate the interval's speed between each epoch or slices of the signal.

5- Processing Frequency Tolerance: classification is heavily dependent on this value, and the value defines the width of the frequency girdle around the simulation frequency.

6- Selected Channels: is a range-based value separated by a colon, it will define the lowest to the highest channel selected for the data acquisition.

The selected channels and the rest of the settings are stored separately in "channel-selector.cfg" and "time-based-epoching.cfg" respectively.

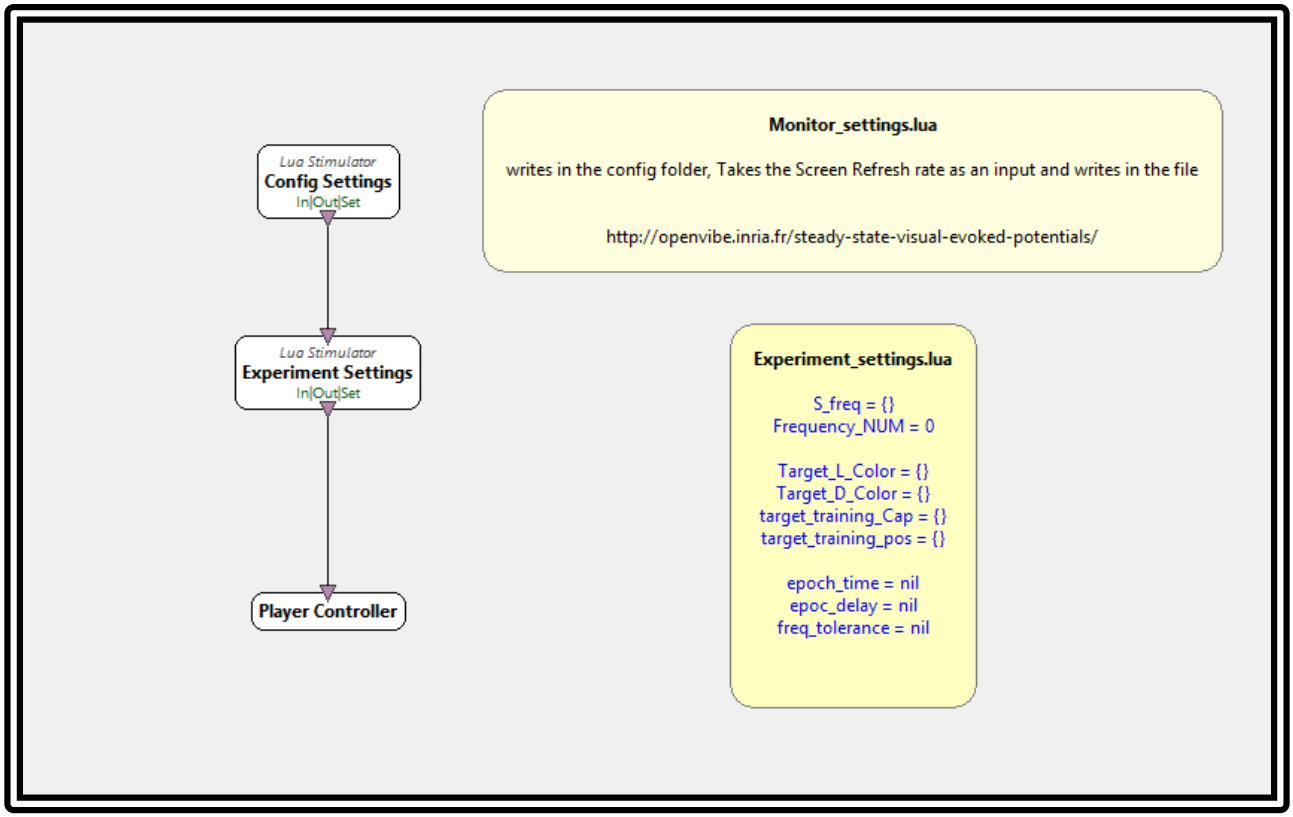

FIGURE 13 CONFIGURATION SETTING SCENARIO

4.1.3 Data acquisition

Before any training, a set of data is needed for classifiers to learn about classifying the data, it was first planned to use a series of pre-recorded data, but it was found in [15] that each subject must be recorded separately for SSVEP system to work properly and classifiers perform a personalised classification for each individual.

As shown in Figure 14, this section works with OpenVibe's premade demo, which is perfect to use for the simulation and gather data from. A list of values in the form of a config file must be first created before running the personalised simulation. Therefore, *training_aquisiton_controller* script is written to ease the process of supplying the values to the simulation.

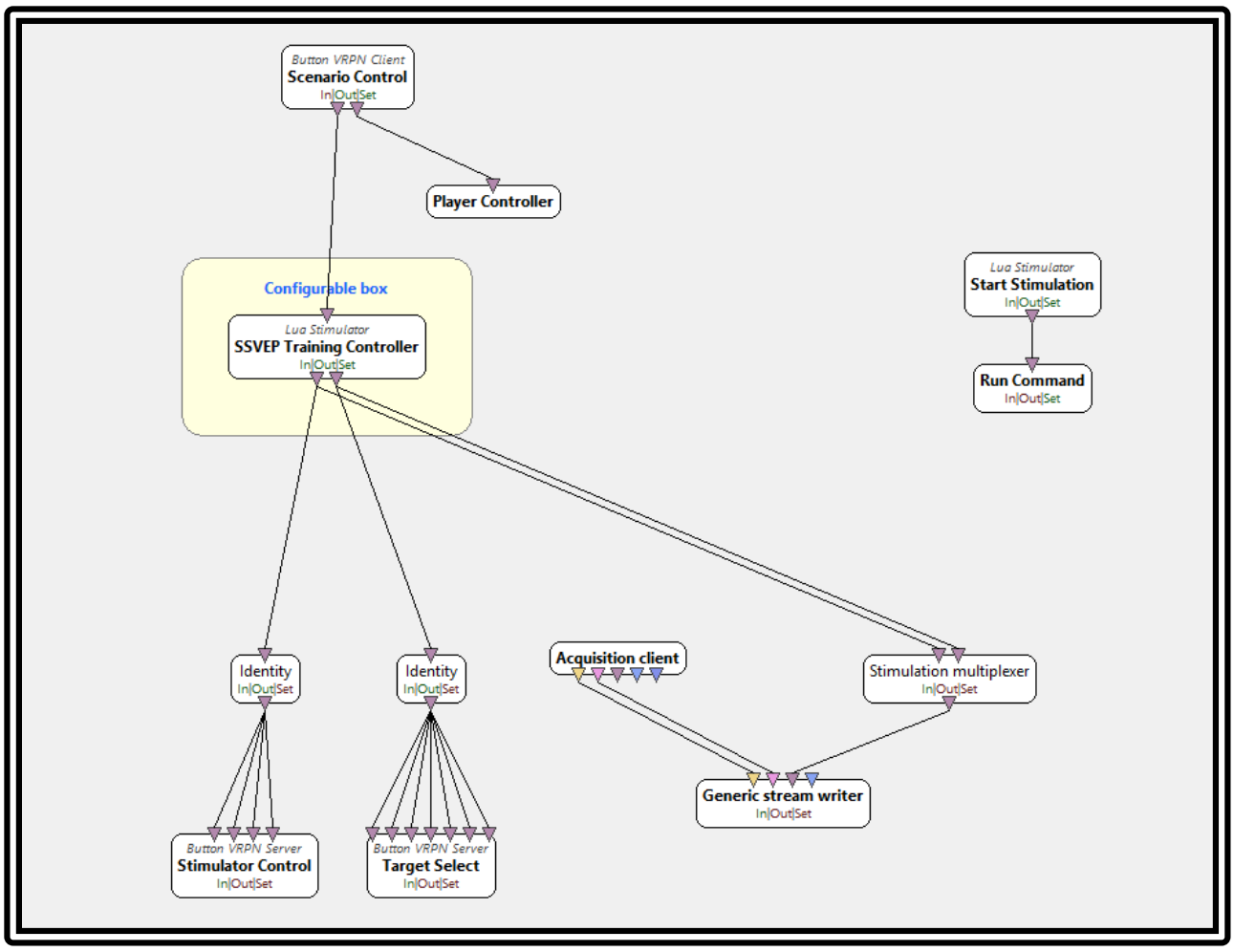

FIGURE 14 DATA ACQUISITION COOPERATED BY OPENVIBE'S SSVEP DEMO

In Figure 14, the SSVEP training Controller box is separately visualised in Figure 15. Values listed below are stored in a configuration file to be used in the OpenVibe's SSVEP demo:

1- Goal sequence, the value supplied must be space-separated, and each number in this section will indicate the order of flickering objects. It is advised that number 0 entered in this section represents the No target as intended for subjects to focus on a blank screen. This helps the signals to give a clear separation to when stimulation must be received and when it shouldn't.

2- Stimulation duration indicates the duration when the goal sequence targets are flickering defined in seconds.

3- Break duration indicates the size of the period when no objects are flickering on the screen.

4- Flickering delay defines the delay in seconds, which occurs between the moment where the indicator is pointing at a target for the subject to look at and the time the flickering starts.

5- The training target size holds a value that will define the size of the flickering objects on the screen.

6- Training Target's position holds position coordination of the flickering objects. The centre of the screen will hold a position value of (0, 0).

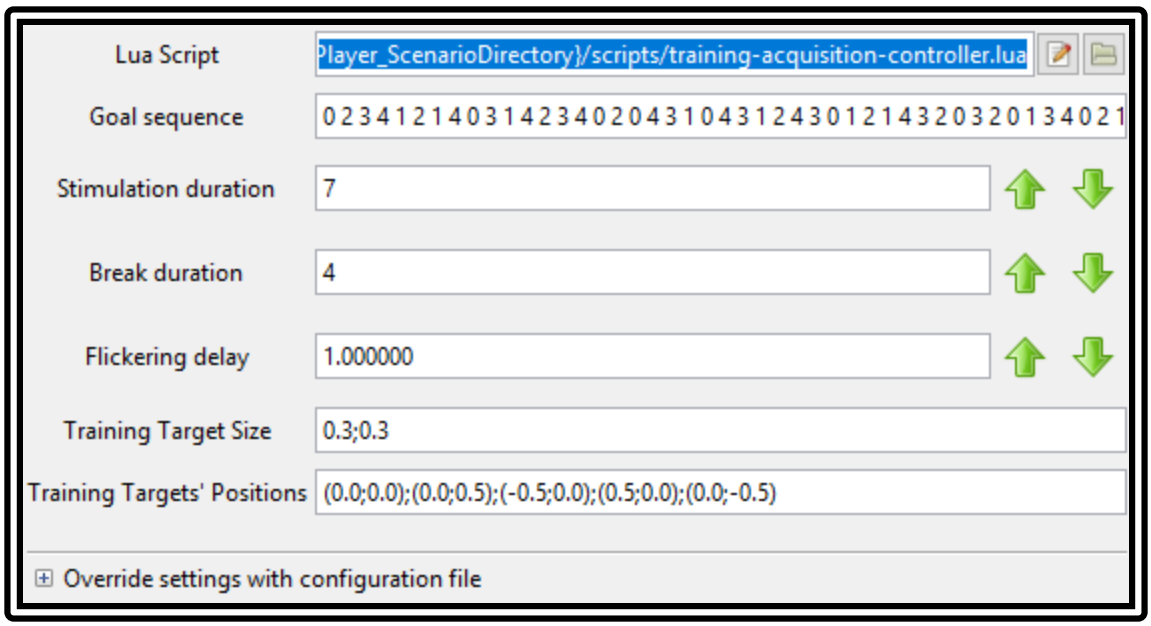

FIGURE 15 SSVEP TRAINING CONTROLLER CONFIGURATION VALUES

The product of this scenario is the recorded file with all the targets marked and classed and can be used for training. The goal sequence shown in Figure 15 is used for training, and it contains 50 goals. It is commonly performed to divide the data into a 30:70 ratio where 30 percent is used for testing and 70 percent is used for training. For simplifying the process, 15 goals are chosen for testing purposes.

4.1.4 CSP training

Common Spatial Pattern (CSP) filters are very important before training the classifier. The scenario creates flexibility in using any set of electrodes. One of the main goals of this scenario is to automatically select the best combination of electrodes for the training. The overall system is visualised in Figure 16.

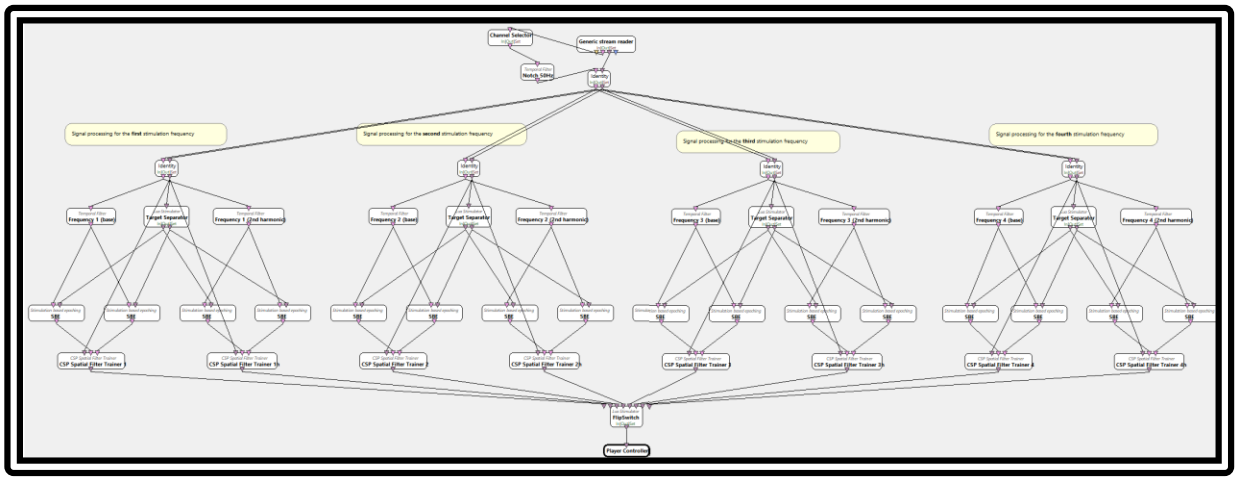

FIGURE 16 CSP TRAINING SCENARIO PRESENTATION

As shown in Figure 17, first the signals recorded in data acquisition will go through some filters, the Channel selector will use the configuration file, and a temporal filter will be used to exclude any signals from other channels and above 50 hertz respectively. For four different frequency target, four different

layers of frequency will be divided where each layer will have two temporal filters (base and second harmonic), for example, the first layer will have frequencies related to the 20-hertz target frequency and the second layer will hold 15 hertz and so on. The target separator box runs the *classifier-training-target-separator* script which will separate targets used to mark one frequency from the others, stimulations related to the outputs will be sent out. The result of this scenario is eight sets of CSP configuration files that will be used in training the classifiers.

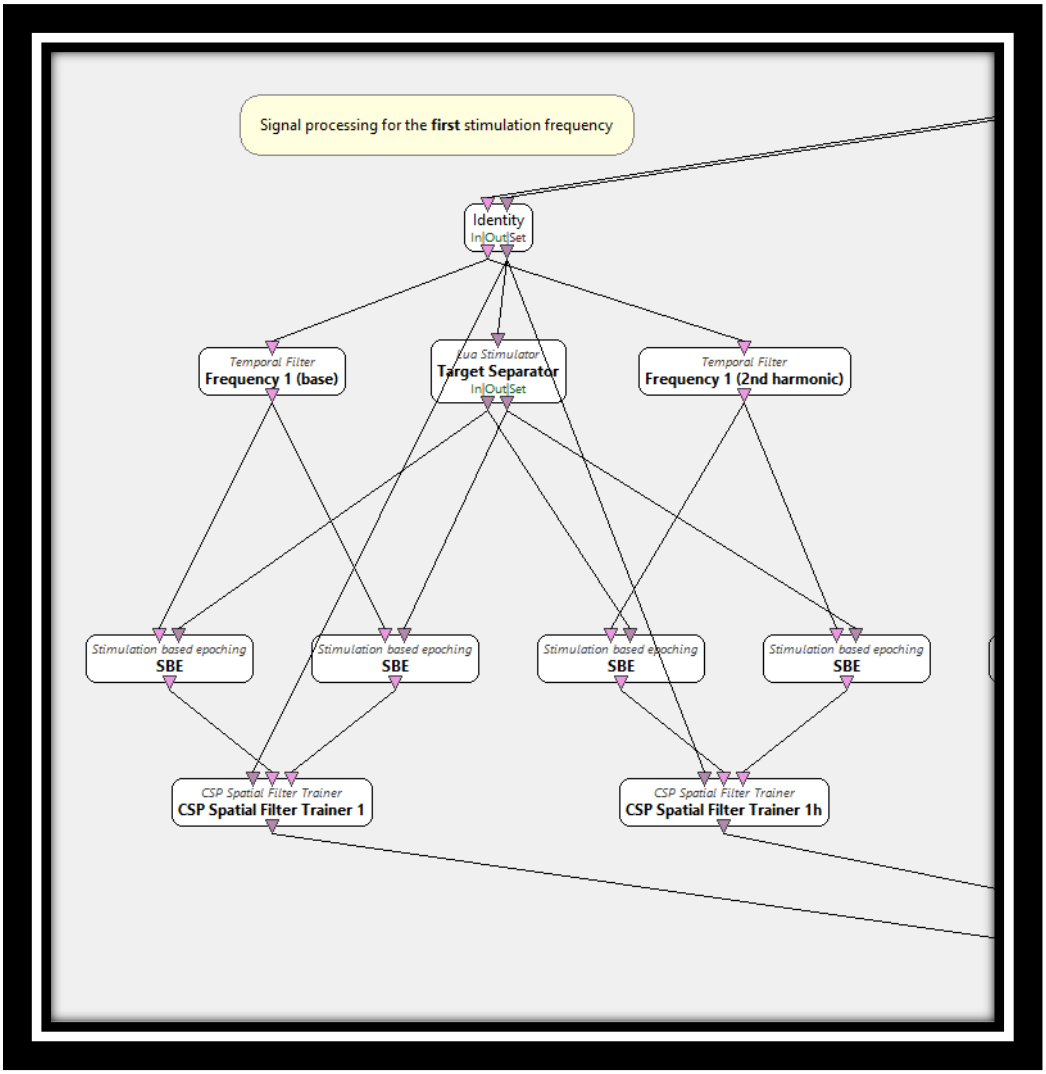

FIGURE 17 CSP TRAINING MAGNIFIED ON PRE-PROCESSING AND FILTER TRAINER FOR TWO HARMONICS OF THE SIGNAL.

4.1.5 Classifier training

The main component where the training of the starts and the classifiers will be outputted to a folder. The recorded signals gathered from the data acquisition will be supplied where signals will be divided into four layers with two harmonics for each signal similarly to CSP training. After the signals went through all the filtering like CSP training scenarios, the filtered signals will now go through spatial filters which are produced by CSP training. Each spatial filter box will filter according to the spatial filter coefficient of that layer. Boxes such as simulation-based epoching and time-based epoching will allow the signals to be epoched for training. The simple DSP box which is a digital signal processing which will optimize or improve the efficiency of the signals by input x to be x*x. The procedure will first optimise the signals by

input x to be x*x and then averages the signals and finally goes through another DSP box which will Log (1+x) for each x input. Before training the frequencies, the preprocessed signals are sent to feature aggregator box where it will be combined into one feature vector. It is mostly used to combine two harmonics of the frequency with being processed by one classifier. The overall scenario can be presented in Figure 18.

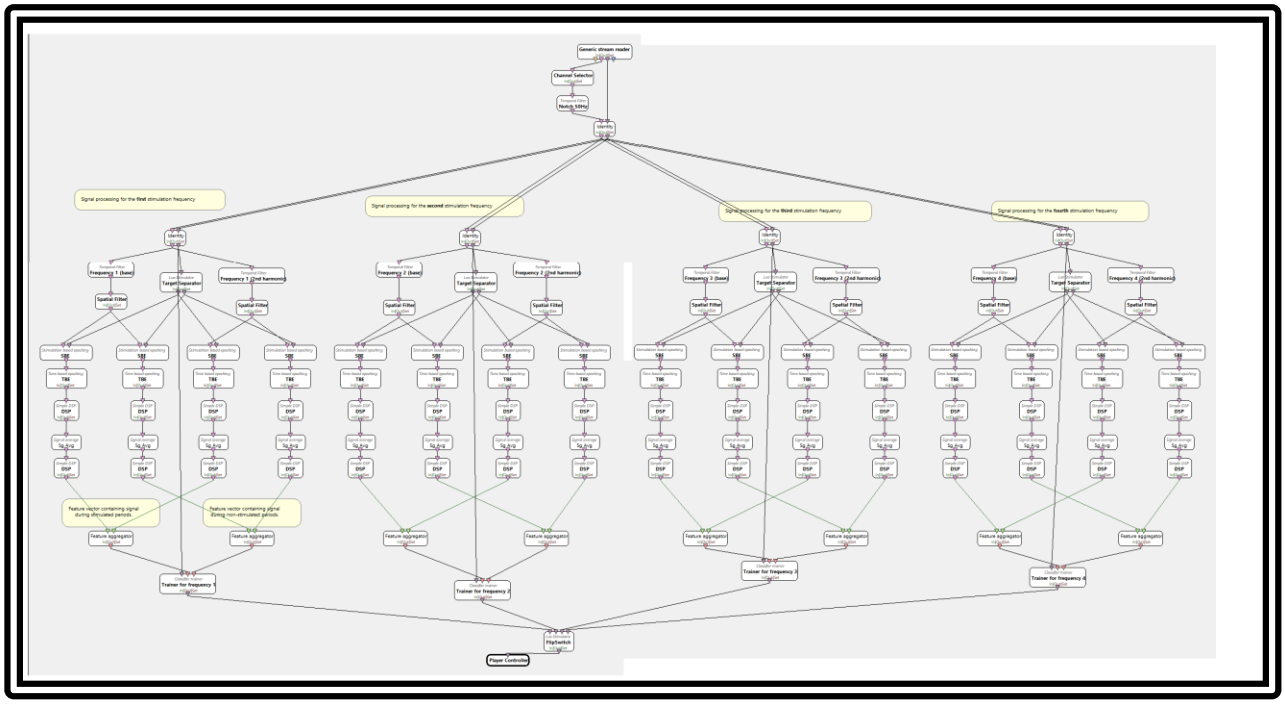

FIGURE 18 CLASSIFIER TRAINING SCENARIO

Each classifier for each layer then will let FlipSwitch script to know that they are done processing and then it will finish running the scenario.

In the classifier box, settings are represented in Figure 19. The current settings are set for the classifier to use the LDA algorithm to train the data. However, Multi-layer Perceptron and SVM are also used to analyse their performance.

FIGURE 19 CLASSIFIER TRAINER BOX SETTING

Figure 20 is a cropped image from Figure 18 which magnified one of the branches of the system where it leads to the classifier trainer. As shown in Figure 20, signals are filtered to be between 19.75 and 20.25. The filtering is configured by the configuration scenario in the Experiment setting box where the tolerance was set to 0.25. Therefore, the signals are filtered between the values mentioned. The second harmonic signal is usually the double of the first harmonic therefore the signals filtered to be in between 39.75 and 40.25. Another filtering is visualised and listed below:

1- Spatial filter configured from the CSP training scenario.

2- Stimulation based epoching.

3- Time-based epoching.

4- Simple DSP applied by x*x.

5- Signal averaging.

6- Another DSP applied by log (1+x).

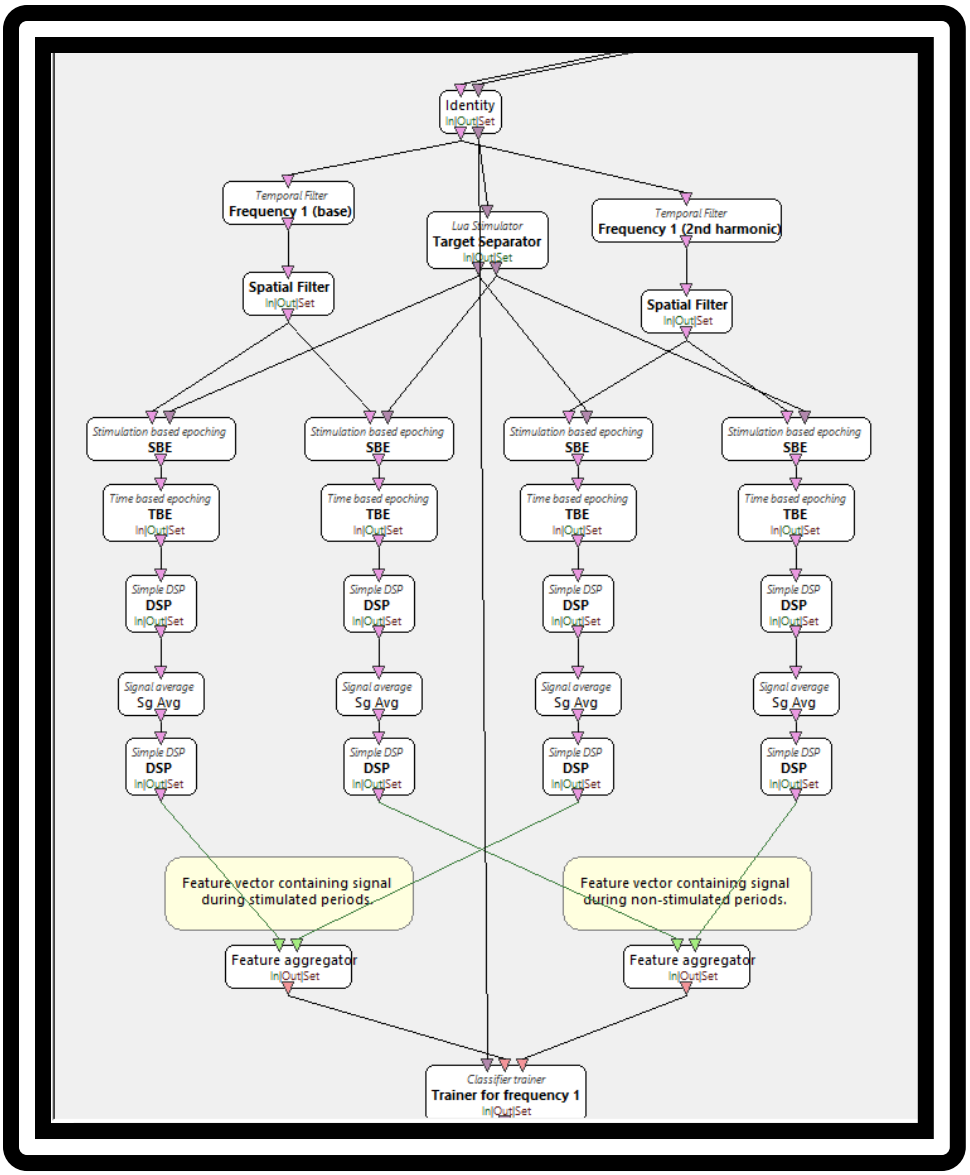

FIGURE 20 PREPROCESSING OF SIGNALS BEFORE REACHING THE CLASSIFIER TRAINER

After each training is done a log of the process and cross-validation test is done, and its accuracy is recorded.

4.1.6 Performance Measurement

In this scenario, the classifiers will be tested with the supplied test data, and the results will be presented as a confusion matrix. The structure in pre-processing the signals that were used in training was used in the performance measurement scenario; however, instead of classifier trainer box, the classifier processor box was used. As shown in Figure 21 classifier processors supply the python script with a streamed matrix which indicates the probability values of the stimulations.

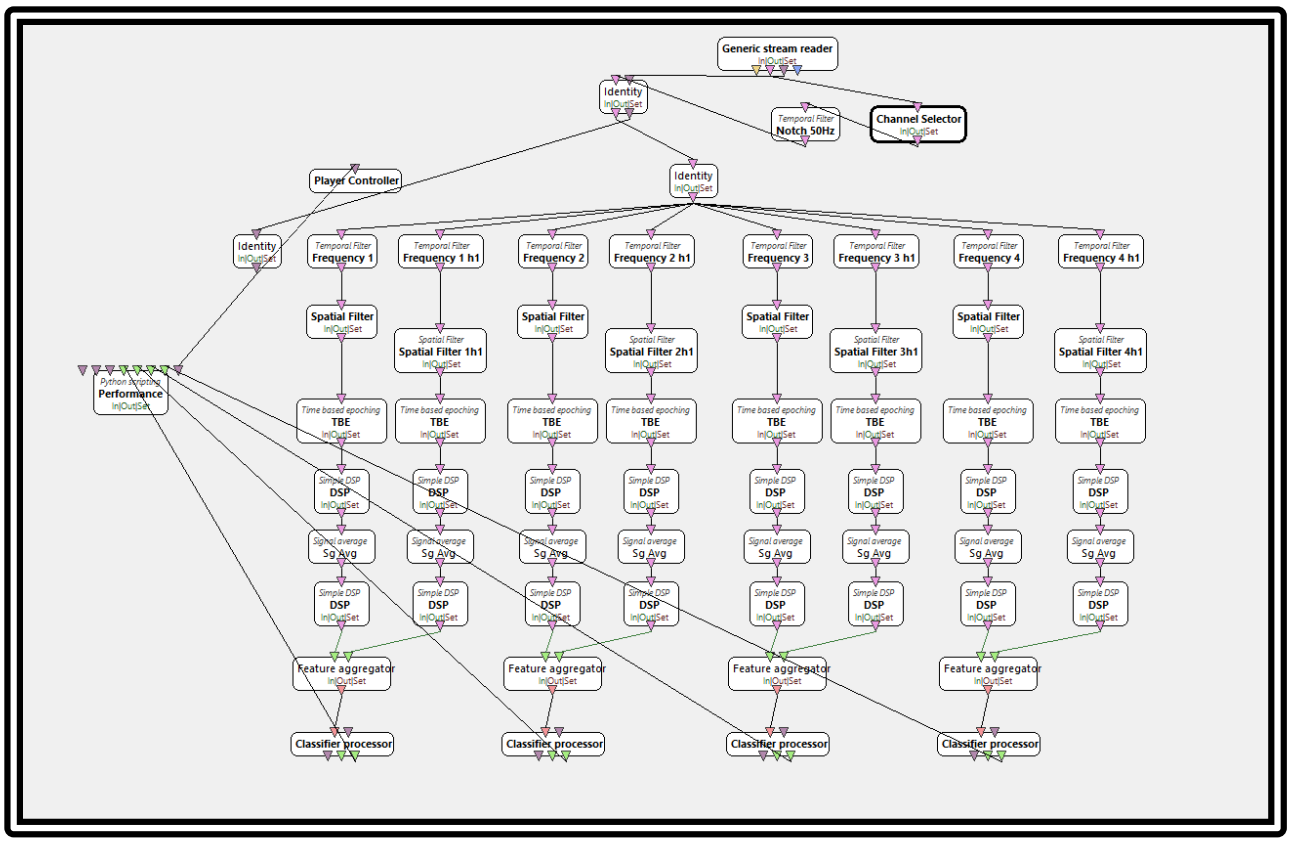

FIGURE 21 PERFORMANCE MEASUREMENT SCENARIO

The implemented python script has a series of inputs that can control the testing. These series of inputs are represented in Figure 22.

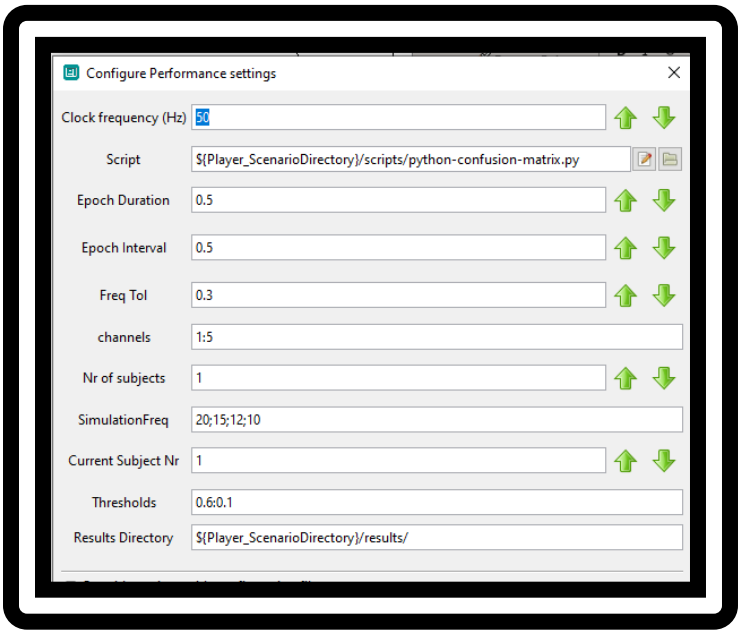

FIGURE 22 PERFORMANCE BOX CONFIGURATION OPTIONS

These series of options are listed below:

1- Clock frequency, clock frequency used in processing the data entered in hertz.

2- Epoch duration.

3- Epoch interval.

4- Frequency tolerance.

5- Channels to gather signals.

6- The number of subjects.

7- Simulation frequencies, for this project, a series of 20, 15, 12 and 10 are used.

8- Current Subject number.

9- Thresholds, for this project, the threshold is set to 0.6 to 0.1.

Furthermore, in the implementation of the performance box, the box uses a python script named python-confusion-matrix. The script uses Sklearn to produce the confusion matrix. First the script will initialize all the values such as:

1- Current label time stop.

2- Current label time starts.

3- Current label.

4- Actual label probabilities.

5- Predicted labels probabilities.

6- The number of stimulations classified.

7- The number of actual stimulations that were classified correctly.

8- A list of class probabilities.

9- Time is taken to predict.

10- Variable to hold the new label.

11- The total number of classified labels.

12- The total number of not classified labels.

13- Current class thresholds.

14- Max probability difference threshold.

All these variables will participate in calculating the confusion matrix.

Briefly explained, the process will first get actual stimulation values and removes it from the data. Then it allows the row of data without the actual stimulation value to be predicted by the classifier. In the next step, the probabilities are stored in a list and checks for any not classified and allocate time to classify the new labels. If the processor cannot predict the label within the allocated time, it will predict the best. The class threshold then will clear the classification hits which contains the maximum probability, the formula below indicates maxPositionDifference will be used for second threshold clearance:

$$maxProbablity = \max(probablity_{list})$$

$$positionOfTheMaxClass = probablity_{list}.index(maxProbablity)$$

$$probablity_{list}.delete(positionOfTheMaxClass)$$

$$maxPositionDifference = positionOfTheMaxClass - \max(probablity_{list})$$

After the probabilities go through the second threshold, the predicted labels will be recorded, and the mean of the time taken to predict will be added to the data recorded. All the data recorded uses the Sklearn's confusion matrix to be set and will be written into a file and will be presented on the debug log screen.

4.1.7 Main Process (online)

The component where the classifiers are used to communicate with other programs through TCP is implemented in this scenario. According to Figure 23 representing the Main process structure, the part for filtering and pre-processing the data is completely the same as the performance measurement scenario with the replacement of Performance box with the Voter box. The voter python script powers the Voter box. The voter python script is quite similar to the performance script where the probabilities must be collected from each processor, and some threshold must filter out redundant probability values. The same formula explained in finding the maximum position difference is obtained and the class with the highest probability value will be obtained from the positionOftheMaxClass and will be supplied to the TCP writer. Stimulations from 0 to 4 will be sent through the TCP port 5678.

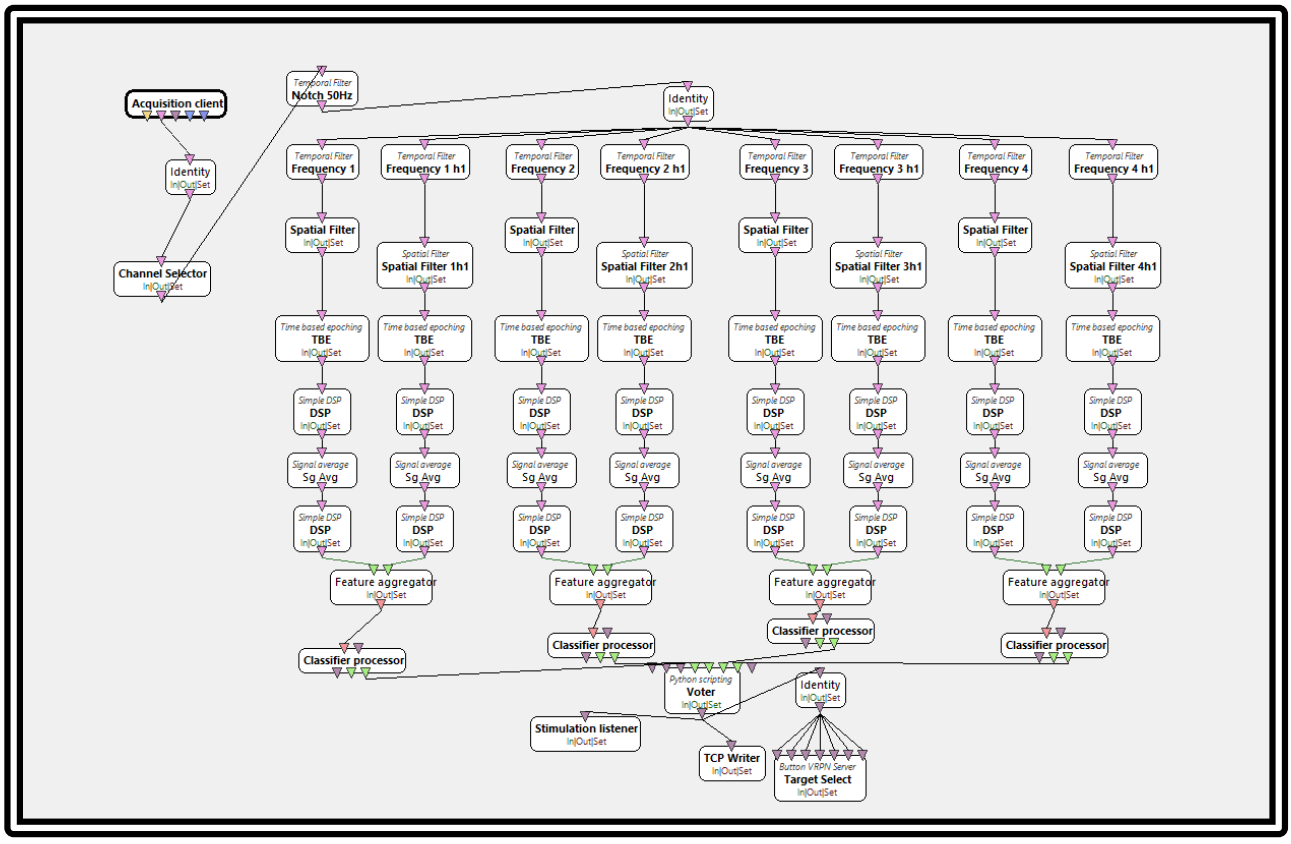

FIGURE 23 MAIN PROCESS (ONLINE) OPENVIBE DESIGN.

The TCP port will hold and carry stimulation codes. These codes are predefined by the OpenVibe's editor and can be shown in Table 3 below:

TABLE 3 COMPLEX ENVIRONMENT'S ASSETS.

OVTK stimulation name	Code in Byte	Code in Integer
OVTK_StimulationId_Label_00	0x00008100	33024
OVTK_StimulationId_Label_01	0x00008101	33025
OVTK_StimulationId_Label_02	0x00008102	33026
OVTK_StimulationId_Label_03	0x00008103	33027
OVTK_StimulationId_Label_04	0x00008104	33028
OVTK_StimulationId_Label_05	0x00008105	33029
OVTK_StimulationId_Label_06	0x00008106	33030
OVTK_StimulationId_Label_07	0x00008107	33031

4.2 Unity-side Implementation

Unity-side implementation was mainly made in creating the experiment and environment's managers and stimulus controllers. These tasks are broken down into smaller tasks and are performed by different classes. These classes can be listed below:

1- *Utilities* class, performing some analysis on calculating the renderer's current frame per second (FPS) and returns the value of the screen's FPS if requested.

2- TCP manager class, this is a helper class where it establishes connections through the dedicated port to be connected to the OpenVibe's Main process scenario.

3- MouseOver, a simple debugging class that checks the mouse or player's facing direction. It activates the action of the object when the player performs the mouse left click. For instance, when the player is facing the light stand and left clicks the mouse, the light will turn on.

4- VR mouse replacement, it is known that using the mouse and VR headset is quite different to navigate, this class will fix the issue with VR headset look direction and helps the stimulus show when the player is looking at the SSVEP powered furniture in the environment.

5- CanvasScript is a class that will allow buttons to appear facing the player. This action is done only if the player is looking at the dedicated furniture.

6- The experiment class is the main class for creating the results of the performance. The class is directly used in *Hove_UI* and supplies the class with capabilities to record data and store them in a list and writes them into a CSV file.

7- Hover_UI class is the main class that will incorporate classes such as the TCP manager, Utilities, and Experiment class. The goal of this class is to make connections with the OpenVibe's Main process scenario and listen to the stimulations; then it will perform the right action according to the stimulations. Moreover, it moves the stimulus to the dedicated areas facing the player. Each stimulus is managed by this class and will flicker.

8- The Java Flasher performs with the same purpose as Hover_UI. The class is compared with Hover_UI, and the best one is picked. The Flasher class is left in the project that is controlled by the Hover_UI for any further implementation.

4.2.1 Utilities

Utilities class has values such as:

1- frame count

2- delta time

3- frame per second

4- refresh rate

Usually, in the update function of Unity, it is classed once per frame. Therefore, it is implemented for the frame count and delta time to increase. The increment of frame count is done by one, and the delta time will increase by the time taken the next update is called. The screens current frame per second is calculated by the formula below when delta time is higher than the $1/refresh\ rate$:

$$fps = \frac{frameCount}{deltaTime}$$

After each time fps is calculated, the frame count is set to zero and delta time is decreased by $1/refresh\ rate$.

There are getter functions implemented for the ease of acquiring frame per second and frame count value.

4.2.2 TCP manager

TCP manager will establish a connection through TCP via 5678 port. The class is supplied with threading to perform any execution in the background and doesn't wait for other commands or other classes to perform. The class uses an enumerator to codify the stimulations received. The enumerator for stimulation is called Stim_code. It dedicates 0 to an unspecified stimulation, and other stimuli such as SL0 to SL7 are calculated by converting the bytes received from the TCP port into a 32-bit integer. For instance, in OpenVibe's main processer class SL0 is coded with 1000000100000000, if this row of bytes converted into a 32-bit integer, it would appear as 33024. Other stimulation codes are presented in Figure 24.

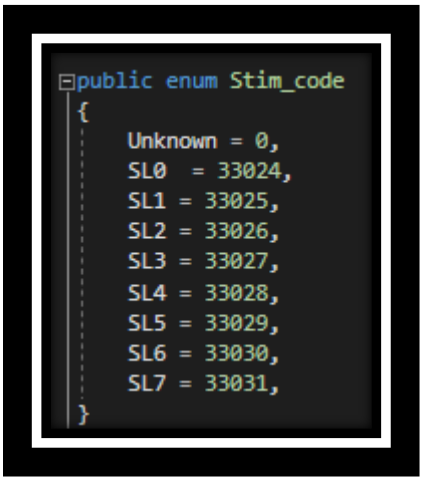

FIGURE 24 ENUM FOR STIMULATION LABELS

The TCP manager class uses the thread to perform the listening function in the background, to establish the connection, the TCP client first is defined to connect to the localhost with the port 5678. An array with a length of 1024 is created. In an infinite loop the RecieveBytes function is called where it will read 1024 bytes of each message at a time as shown in Figure 25. The data required from this array is extracted and converted into 32-bit integer. The converted message then will be codified by the enumerator and will be stored in an accessible variable.

```
try
{
    using (NetworkStream s = tcpClient.GetStream())
    {
        int l;

        while ((l = s.Read(b, 0, b.Length)) != 0)
        {
            Debug.Log("connected");
            var data = new byte[l];
            Array.Copy(b, 0, data, 0, l);


            int code = BitConverter.ToInt32(data, 0);

            mymessage = message_converter(code);
            if (is_debugging)
                Debug.Log("Message recieved: " + mymessage);


              }
    }
```

FIGURE 25 A SNIPPET FROM RECIEVEBYTES FUNCTION IN THE TCP MANAGER CLASS

4.2.3 MouseOver

Mouseover is an important class for debugging and some internal changes in the environment, the class was implemented easily. As shown in Figure 26, it demonstrated the OnMouseUp Unity's inbuilt function, which determines if the cursor has entered the objects collider region. By a simple click the action is performed related to the furniture.

```
private void OnMouseUp()
{

    if (light.activeSelf)
    {
        light.SetActive(false);
    }
    else
    {
        light.SetActive(true);
    }


}
```

FIGURE 26 MOUSEOVER CLASS USING INBUILD UNITY FUNCTION ONMOUSEUP

4.2.4 VR mouse replacement

In a common display, the player can look around using the mouse cursor. The navigation of the looking direction is different in VR. For instance, the player is set to look forward to the game, and if the player moves their head and looks left, the camera will move with the player's head orientation. However, the

body does not rotate unless the player manually controls the body to rotate. Therefore, in the problem-solving of this part of the project, this problem was found and quickly fixed by adding functionality to VR that uses the ray casting from the camera's look directly to the observed object.

The process is done by defining a tag in the editor named *Stimuli-obj*. A ray is beamed using Unity's RayCaster, and the position of this ray is set to the camera's transformation. The ray is beamed at objects while the player is looking around, the ray is checked, and if it collides to an object with the tag defined, as shown in Figure 27, it will trigger Hover_UI's functionality to perform and create stimulus.

```
var ray = Camera.main.ScreenPointToRay(this.transform.position);
RaycastHit hit;

if (Physics.Raycast(ray, out hit))
{
    var s = hit.transform;
    if (s.CompareTag(tag))
    {
        var sr = s.GetComponent<Renderer>();
        if (sr != null)
        {
            s.GetComponent<Hover_UI>().Selected();

        }
        t = s;
    }
}
```

FIGURE 27 RAYCASTING METHOD USED IN VR

4.2.5 CanvasScript

CanvasScript is mainly used for the stimulus objects to be instantiated in the scene and have a specific orientation where it is always facing the player. The transformation used to adjust these orientations is the current main camera in the scene. It is always looking at the player by the code math snippet below:

$$Tranformaton_{Canvas} = position_{canvas} - Rotation_{camera} * Vector3(0,0,-1)$$

$$UpTransoformation_{Canvas} = \ Rotation_{camera} * \ Vector3(0,1,0)$$

Figure 28 shows that the Create_buttons uses a previously made button object and required button configuration to instantiate these buttons. Button configuration only sets the text written on the button.

```
public GameObject Create_buttons(GameObject b, buttonsconfig bc)
{

    GameObject childobj = Instantiate(b,gameObject.transform) as GameObject;

    childobj.GetComponentInChildren<Text>().text = bc.text;



    childobj.transform.SetParent( this.gameObject.transform);

    return childobj;

}
```

FIGURE 28 CREATE_BUTTONS FUNCTION IMPLEMENTED IN CANVASSCRIPT CLASS

4.2.6 Experiment

The experiment class is used for gathering results for any analysis done in chapter 5 Results and Discussion. The experiment class contains an inner class named *Records* for storing performing encapsulation in the required data to be stored. Each *Record* object will have attribute such as:

1- *Timer*, a variable indicating the time taken for that stimuli to be activated.

2- *Button_direction* which uses an enumerator to codify the directions the buttons can move to.

3- *Stim_code* that uses the enumerator from the TCP manager to store the stimulation code for that specific button.

4- *HZ* variable indicating the frequency of the stimuli.

5- *Total_Num_tries* indicating to the number of tries until the stimuli failed to trigger or successfully triggered.

6- *Successful_tries* is used to store the number of successful triggers during the stimulation of that specific button.

7- Achieved indicating if stimulations successfully triggered the button.

When the object is created, the stimulation code or *stim_code* is checked by the *check_stimCode* function. The experiment settings must keep its integrity and not change throughout the experiment. Therefore, the button with a specific direction is given a specific stimulation code.

The setting is described in Table 4:

TABLE 4 EXPERIMENT SETTING FOR STIMULATIONS RELATED TO THE DIRECTION OF THE BUTTONS.

Direction	Stimulation Code	Frequency (HZ)
Down	SL3	12
Left	SL1	20
Right	SL4	10
Up	SL2	15

Some helper functions also added to the *Records* class to manipulate the values such as the timer, number of tries and successful tries.

In the Experiment Class, it uses three different enumerators to generate the name of the CSV file.

These enumerators are:

1- *ClassifierType* enum holds values such as LDA and SVM. NN for neural networks was added but never experimented in the project.

2- *EnvironmentType* enum contains *SimpleEnvironment* and *ComplexEnvironments* to distinguish experiments done in each environment.

3- *DisplayType* enum holds *VR* and *Screen* values for dividing the tests into each enum.

The experiment class will create a list of records where it holds only three or four maximum records.

The *experiment_checker* function will first check if the next stimuli are called and then creates a new *Records* object and return it.

The new experiment will commence by the *Start_experiment* function where it will add the old generated *Records* object into the dedicated list, and after 3 seconds it will create a new *Records* object by the *experiment_checker* method. The exportation to the CSV file is done simply by a loop exploring the *Records* list and calls the *write_into_csv* method write into the file.

Figure 29 shows that the name is created by "DisplayType-EnvironmentType-ClassifierType-Subject Number_of_the_subject.csv".

```
void Export_to_file()
{
    string filename2 = displayType.ToString()+"-"+environmnetType.ToString()+"-"+classifierType.ToString()+"-"+filename + subj_num+".csv";
    StreamWriter outStream = System.IO.File.CreateText(filename2);
    outStream.WriteLine(lineSeperater + "Button Direction" + fieldSeperator + "Frequency" + fieldSeperator + "Successful tries" + fieldSeperator + "Total tries" + fieldSeperator + "Duration till action performed");
    foreach (Records i in records)
    {

        write_into_csv(i, outStream);

    }
    outStream.Close();

}
```

FIGURE 29, NAME GENERATION ILLUSTRATED. THE IMAGE IS TAKEN FROM THE EXPERIMENT CLASS.

4.2.7 Hover_UI

Hover_UI is the main class where if it is attached to any object in the editor, it will convert that object into an SSVEP powered object. It has an inner class named *buttonconfig*, where it holds information about each stimulus button settings. The inner class has configurable attributes such as:

1- The direction of the button.

2- The action of the button defined in *LightAction* enumerator.

3- Frequency of the button in HZ.

4- The text where it is written on the button.

The none configurable attributes include:

1- The delta time between each update is called.

2- Current frame count.

3- The Button object.

The button configuration is set to be completely dynamic and user-friendly in the editor where up to four buttons can be added and can be fully customised. This extend of flexibility is made by serializing the *buttonconfig* class.

As shown in Figure 30, the button list size is set to three. Therefore, three elements 0,1 and 2 is appeared to be customised by the user.

FIGURE 30 BUTTON CONFIGURATION IN THE UNITY EDITOR.

At the start of running the program, objects from Utilities, Experiment, and CanvasScript will be initialised. An option for Java integration is made by running the jar artifact file of the Flasher java which can be used in replacing the stimulus implemented in the unity. A list of buttons will be initialised from the elements mentioned in Figure 30 and uses the CanvasScript object to instantiate new buttons. The program will not allow the buttons to appear or flash if the player does not face the dedicated furniture. After the player faces the furniture, the program will run the *buttons_change_color* method.

```
private void buttons_change_color()
{

    if (Java_integrated)
    {

        dontflash();
    }
    else
    {

        if (Is_shown)
        {
            ObjText.text = ObjString;
            ObjText.color = Color.white;
            foreach (buttonsconfig b in buttons)
            {
                b.stimuli.enabled = true;
                b.stimuli.GetComponentInChildren<Text>().color = Color.Lerp(ObjText.color, Color.white, f_time * Time.deltaTime);

                if (Vector3.Distance(Original_pos, b.stimuli.transform.position) < distance - error)
                {
                    buttons_controller(b);
                }
                else
                {
                    flash = true;
                }
            }


        }
        else
        {

            dontflash();


        }
```

FIGURE 31 BUTTONS_CHANGE_COLOR METHOD DEFINED IN THE HOVER_UI CLASS

As shown in Figure 31, the program will first enable each button to appear in the scene; each button's text is separately fading in to appear. Then the distance of the button's original position is with their current position is calculated and compared with the distance entered for them, this will push the buttons to move towards a direction away from each other and appear in a smooth animation. An example of this animation is illustrated in Figure 32.

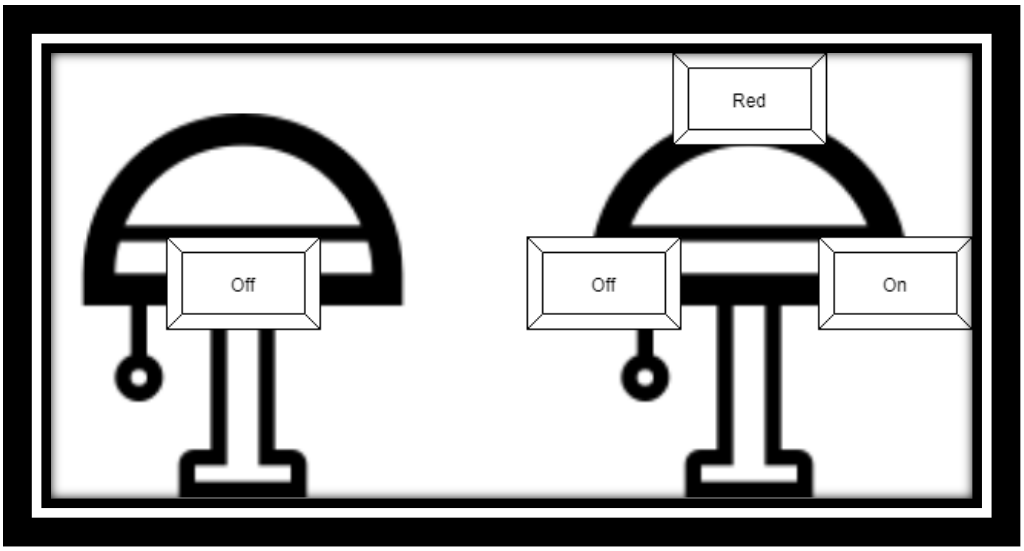

FIGURE 32 BUTTON PLACEMENT AFTER EACH REACHED THE MAXIMUM DISTANCE AWAY FROM THE ORIGINAL POSITION.

Three different flickering methods were tested, and the best one was chosen; these methods can be listed:

1- Frame-based flickering.

2- Time-based flickering.

3- Java-Integrated flickering.

Frame-based flickering is implemented in the *button_flash_fixed* method. The method is illustrated in Figure 33. The method uses the current frame count and compares it with $\frac{1}{Frequency\ of\ the\ button}$. If the current frame is larger or equal to the one over frequency of the button, then it will turn disable or activate the button to disappear or appear respectively.

```
private void button_flash_fixed(buttonsconfig b)
{


    if(b.frame >= 1/b.HZ)
    {

        b.stimuli.enabled = !b.stimuli.enabled;
        b.frame = 0;

    }
```

FIGURE 33 FRAME-BASED FLICKERING USING THE FRAME TO MAKE APPLY THE FLICKERING EFFECT.

Time-based flickering is implemented in the *button_flash_based_on_time* method, as shown in Figure 34. It calculates the next time to flicker by dividing one over flash rate times the delta time. It will apply the flickering effect based on the next calculated time.

```
private void button_flash_based_on_time(buttonsconfig b)
{

    float when_to_flash = b.HZ;
    when_to_flash = 1 / (when_to_flash * Time.deltaTime);

    float current_T = Time.realtimeSinceStartup;

    if (current_T >= b.change_time)
    {

        b.change_time += (current_T + when_to_flash);
        b.stimuli.enabled = !b.stimuli.enabled;
    }
```

FIGURE 34 TIME-BASED FLICKERING METHOD

For each method time and the flicker, the action moment is recorded and analysed by a helper *Kodatriser* python script. The script will plot graphs where it will show the stability of the method.

After testing, the Frame-based flickering had superior results to time-based flickering and java integrated flickering. Therefore, it will be used in the experiment. According to Figure 35, there are some very small unbalanced loadings in Java-based flickering, which may disrupt the pattern. Furthermore, the Time-based flickering had the poorest performance where it could not stabilise after the 8th step. This method may be caused by the Unity update function where it only updates one frame at a time.

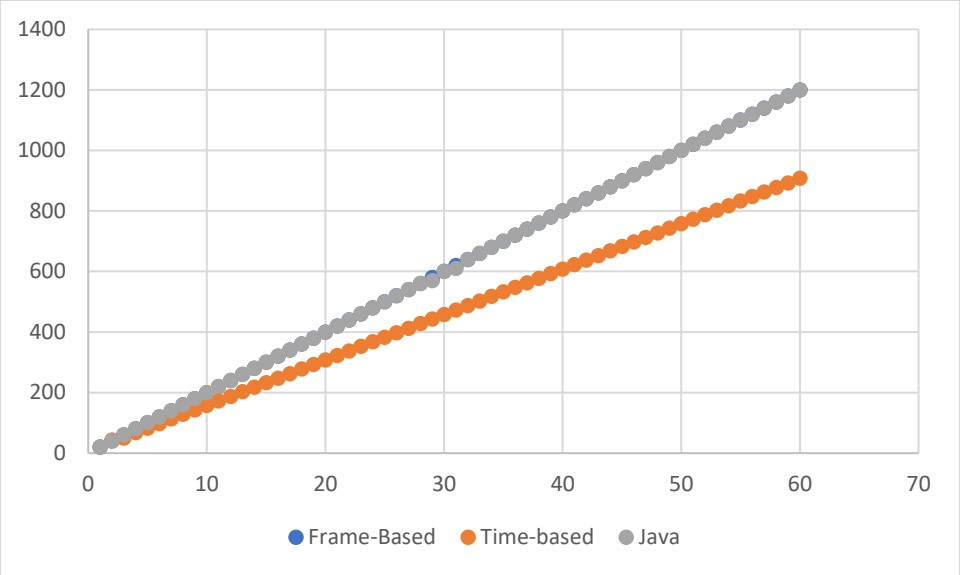

FIGURE 35 FRAME-BASED, JAVA, AND TIME-BASED FLICKERING ARE ANALYSED BY THE PYTHON SCRIPT

Buttons will stop flickering by the method *dontflash*; this method will crossfade the alpha value of the image to 0 within the specified fade time and uses *Color.Lerp* function to fade out the text on the button too. The *dontflash* method will also move the buttons to the original position while fading out.

The buttons are moved back using the *move* method as shown in Figure 36. The function will take the Button object and the duration taken for it to move back to the original position as variables to manipulate the speed the button will move towards the original position.

```
private void move(Button targetButton, float duration)
{

    float elapsed = 0;
    Vector3 start_pos = targetButton.transform.position;

    while (elapsed < duration)
    {
        targetButton.transform.position = Vector3.Lerp(start_pos, Original_pos, elapsed);
        elapsed += Time.deltaTime;




    }

    targetButton.transform.position = Original_pos;



}
```

FIGURE 36 MOVE METHOD FOR MOVING THE BUTTONS TO THE ORIGINAL POSITION WITHIN THE REQUIRED TIME.

Each button contains an attribute that describes the action it will do if triggered. These actions are defined in an enumerator named *LightAction*. *Hover_UI* will use the TCP manager to listen to the messages and will check with the stimulation attributes of each button and will perform the specified action. *Action_from_tcp* is the method that will cover these processes. The stimulations SL1, SL2, SL3, SL4 are received for frequencies such as 20, 15, 12 and 10 respectively. Therefore, a button with 20-hertz frequency will have SL1 to trigger its action.

4.2.8 Java integration

For one of the flickering methods, a java method was written that the *Hove_UI* class could externally call this program and set its buttons and frequencies. This program contains five classes:

1- The *CONSTANTS* class holds all the constant variables.

2- The *BasicView* loads the graphics and allows the screen to be loaded with a black background.

3- The *CustomRec* is a dynamic rectangle generator class where it can create the required rectangles; it also has the capability of the flickering.

4- The *FlasherMain* is the main class where it can take the number of rectangles and the starting frequency as its arguments, and it will create the buttons and flickers them with the provided frequencies. The starting frequency will increase for each button.

5- The *Vec2D* is a two-dimensional position class where it will store any objects positions x and y value in this class, and it is used in the *CustomRec* class for positioning the rectangles.

Chapter 5 – Results and discussion

5.1 Experiment Complications

The experiment was intended for four subjects to participate. Due to some complications and limitations of the time, most of the participants were not able to fulfil all the tests assigned, and some only went through them partially. Therefore, the tests were only performed on one subject, and the mean average of the results are represented in the tables mentioned in this section. One of the issues with the experiment was using both Enobio 20 and the VR headset, which pressured the pins to the scalp making it uncomfortable to wear. Therefore, some factors must be taken into account on VR experiments where the sense of uneasiness can affect the signals and the classification of the result.

5.2 Experiment equipment

Enobio 20 with 20 channels was used to record the raw EEG signal. Enobio 20, as explained in [6], uses drytrodes and NG Geltrode to record the singles where the NG Geltrode requires a small amount of gel is applied to record the signals.

FIGURE 37 ENOBIO 20 HEADGEAR, IMAGE TAKEN FROM [6]

Oculus rift s [35] is one of the newest Oculus virtual reality products; the headset has 2560 by 1440 screen resolution with a refresh rate of 80 HZ.

The experiment was performed on a personal computer with the specifications:

1- Video card: GeForce GTX 1060.

2- CPU: INTEL i7.

3- RAM: 16 gigabytes.

4- Screen Resolution: 1920 by 1080 (60 HZ)

5.3 Experiment Process

As explained in the testing design part of Chapter 3 Design, the subject is assigned first to have their raw EEG signal to be monitored by the Acquisition testing scenario, the monitoring is used to diagnose any problems with the pins or the placement of the pins where it will affect the quality data gathered for classification. Before data acquisition, settings are configured by configuration setting scenario, and the screen refresh rate is set to 60 hertz. The raw signals are gathered by the data acquisition scenario, and the subject is sat through 50 different goals displayed on the screen and flickered for 7 seconds. Frequencies

such as 20, 15, 12, 10 hertz are applied for dedicated goals in the simulation. The raw EEG signals gathered from the sequence of 50 goals simulation were stored in a file distinguished form the test data.

Fifteen goals in the result of 30 percent extracted from the 50 goals are used in the testing session, and the subject is requested to experience these goals in the experiment. The raw EEG signals gathered are saved into a different file from the training file.

The subject is requested to relax while the raw EEG signals are preprocessed and classified by the CSP training and classifier training scenario. The classifiers LDA and SVM are used in the classification, and their performance throughout five partitions of the data is measured and shown in Table 5 and 6.

The classifiers obtained from the classifier training scenario are used to be tested with the test data gathered, and the results are shown in Table 7, 8, 9 and 10.

The subject is set to be tested with a complex and simple environment designed with VR and conventional screen. The results achieved for each classifier. Tables 9 shows the data gathered for this experiment.

5.4 Discussion

LDA training performance was set to be optimistic for the different frequencies. According to Table 5, the highest training-set accuracy achieved was 82.428 percent for 15 frequency. The lowest accuracy achieved was 80.286. The accuracies for all frequencies are above 80 percent. However, in Cross-validation frequency 12 achieved 79.857 percent, which is lower than other frequencies which achieved higher than 80 percent.

TABLE 5 LDA PERFORMANCE IN THE CLASSIFICATION TRAINING.

Frequency	Partition one perf %	Partition two perf %	Partition three perf %	Partition four perf %	Partition five perf %	Cross validation %	Training-set accuracy %
20	78.571	90	70	83.571	83.571	81.142	81.428
15	76.428	82.857	90	72.142	80	80.285	82.428
12	80.714	78.571	86.428	75.714	77.857	79.857	81.142
10	80.714	70	80	90	80	80.142	80.285

SVM training performance was also set to be optimistic, achieving higher than 80 percent throughout the list of frequencies. In comparison to Table 5, SVM could not reach maximum accuracy of LDA classification accuracy, and its highest reached 81.285. The average of training-set accuracy for LDA is 81.32 and for SVM is 80.642. LDA had better performance by 0.67 percent. Moreover, Table 6 shows that in training SVM had poorer performance in Cross-validation and did not reach 80 percent throughout training different frequencies. The highest achieved was 79.857 and the lowest of 78.571. In comparison to Table 5, the average of LDA cross-validation is 80.35 in oppose the SVM cross-validation achieved 79.24.

TABLE 6 SVM PERFORMANCE IN THE CLASSIFICATION TRAINING.

Frequency	Partition one perf %	Partition two perf %	Partition three perf %	Partition four perf %	Partition five perf %	Cross validation %	Training-set accuracy %
20	78.571	90	70	80	80	79.714	80.857
15	68.571	84.285	89.285	72.142	80	78.857	81.285
12	80	80.714	80	75.714	76.428	78.571	80.285

10	80	69.285	80	90	80	79.857	80.142

Tables 7 and 8 were created from testing the LDA classifier with the test data. The LDA performed higher with higher frequencies, and a decline can be seen in the accuracy as the frequency decreases. The highest accuracy achieved was 82 percent where the lowest was 56 percent on 10-hertz frequency.

TABLE 7 LDA CLASSIFIER CONFUSION MATRIX ON THE TEST DATA.

Confusion matrix			
0.	0.172	0.	0.827
0.055	0.222	0.	0.722
0.052	0.394	0.026	0.526
0.	0.439	0.	0.560

TABLE 8 LDA CLASSIFIER ACCURACY ON THE TEST DATA.

Frequency	True Negatives	False Positives	False Negatives	True Positives	Accuracy %
20	0	5	0	24	82
15	2	8	0	26	77
12	2	15	1	20	57
10	0	18	0	23	56

Comparatively, SVM performance had more stability and performed higher than LDA classifier with the highest accuracy of 94 in 20-hertz frequency and lowest of 58 percent in classifying 12-hertz frequency. However, the 10-hertz frequency was classified poorly by LDA, and SVM managed to achieve 24 percent more accuracy in classifying the lowest frequency.

TABLE 9 SVM CLASSIFIER PERFORMANCE ON THE TEST DATA.

Confusion matrix			
0.	0.057	0.	0.942
0.088	0.117	0.088	0.705
0.	0.083	0.333	0.583
0.064	0.161	0.032	0.741

TABLE 10 SVM CLASSIFIER ACCURACY ON THE TEST DATA.

Frequency	True Negatives	False Positives	False Negatives	True Positives	Accuracy %
20	0	2	0	33	94
15	3	4	3	24	79
12	0	2	8	14	58
10	2	5	1	23	80

Table 11 listed eight different experiment settings, and the LDA and SVM classifiers are tested with three different frequencies which are 20, 15, 10. The total number of predictions the classifier has is recorded.

The successful cases indicate if the classifier was successful in predicting the actual frequency if the value is more than 1. The maximum number of successful tries is two for checking if the classifier is not predicting two different frequencies right after another. Time taken in Table 11 shows the duration by which the classifier was successful in predicting the actual frequency. In the complex environment without VR, LDA has performed on lower frequency as it was expected from Table 8, SVM, on the other hand, had no response to frequency 15 which is calculated to have 79 percent accuracy in Table 10.

Furthermore, both classifiers performed well in a simple environment without VR. The VR experiment with the complex environment only had one successful attempt using LDA to classify the highest frequency (20 hertz). However, SVM classification only lacked accuracy during classifying the lowest frequency (10 hertz). LDA performance in use of VR in simple environment achieved a very good result by far with total time taken of 37.44 seconds in classifying all three frequencies. Comparatively, in the same settings SVM took nearly half the time of LDA with total time taken of 16.16 seconds to classify all three frequencies.

TABLE 11 MAIN EXPERIMENT SET OF 8 DIFFERENT EXPERIMENTS WITH DIFFERENT SETTINGS.

Experiment	Button Direction	Frequency	Successful tries	Total tries	Time Taken
Screen-ComplexEnvironmnet-LDA	left	20	2	2	4.931873
	Up	15	2	404	8.963979
	right	12	0	2917	50.01824
Screen-ComplexEnvironmnet-SVM	left	20	2	2	14.72974
	Up	15	0	1771	50.01747
	right	12	2	1204	18.94426
Screen-SimpleEnvironmnet-LDA	left	20	2	267	5.37792
	Up	15	2	568	9.556704
	right	12	2	473	7.972616
Screen-SimpleEnvironmnet-SVM	left	20	2	2	0.0656072
	Up	15	2	973	12.48798
	right	12	2	135	1.02695
VR-ComplexEnvironmnet-LDA	left	20	2	2	0.066126

	Up	15	0	766	50.02257
	right	12	0	509	50.03016
VR-ComplexEnvironmnet-SVM	left	20	2	2	1.846429
	Up	15	2	186	11.40164
	right	12	0	1108	50.03331
VR-SimpleEnvironmnet-LDA	left	20	2	59	8.950642
	Up	15	2	536	21.62804
	right	12	2	21	6.878209
VR-SimpleEnvironmnet-SVM	left	20	2	2	6.724033
	Up	15	2	19	0.4945842
	right	12	2	45	8.9519

Due to the low number of subjects some assumptions from these results will not be as credible as an experiment with a high number of subjects. However, the research findings show that testing on the pre-recorded data had some correlation with the real-time use as the classifiers seemingly performed similarly to the pre-recorded data and classifiers with lower accuracy on a specific frequency performed the same on real-time classification. The SVM classifier had better accuracy on real-time and more stability on pre-recorded data. As seen in the table, the LDA had an obvious decline in accuracy as the frequency of stimulus frequency was lowered to 10 hertz. The complex environment with high-resolution textures had an important impact on classification. The reasons for this impact cannot be measured; the immersion of an environment cannot be measured and quantified. However, it was questioned with the subject about the environment, and they explained the complex environment to be realistic and somehow distracting. The simple environment had a positive impact where VR and screen performed seemingly well. Virtual reality gear had improved the performance in the simple environment where in comparison to the conventional screen, it had higher performance. It can be reasoned that the simple environment had no natural lighting and had no complicated textures where the subject was forced to finish the test with no distraction. The subject described the simple environment as a plain environment and somewhat boring.

To sum up the results, the clear optimal VR integrated environment is the simple environment which has performed fastest and predicted correctly using both LDA and SVM classifiers. It should also be mentioned that the conventional screen used had similar experience and performance. However, the performance of the classifiers was close to their performance with pre-recorded data.

To answer the research questions asked in Chapter 1 in the introduction, the experiment is not quite credible due to the small number of subjects. However, it is shown in Table 11 where simple environment had better result in comparison to the complex environment. Furthermore, VR improved the experience and performance in simple environment, the distraction was noticed during complex environment either with the screen or VR. In analysing the classifiers SVM and LDA, both classifiers had the best outcome in VR and simple environment experiments, but for low performed experiments throughout this list, SVM had more stability than LDA. According to Figure 38, SVM has better performance in all frequencies. However, it can be observed that both classifiers had a decline in accuracy of classifying frequencies lower than 20 hertz, the decline continued to 56 percent for LDA classification of 10-hertz frequencies. Comparatively, SVM had more stability and redeemed itself by achieving 80 percent accuracy in 10-hertz classification.

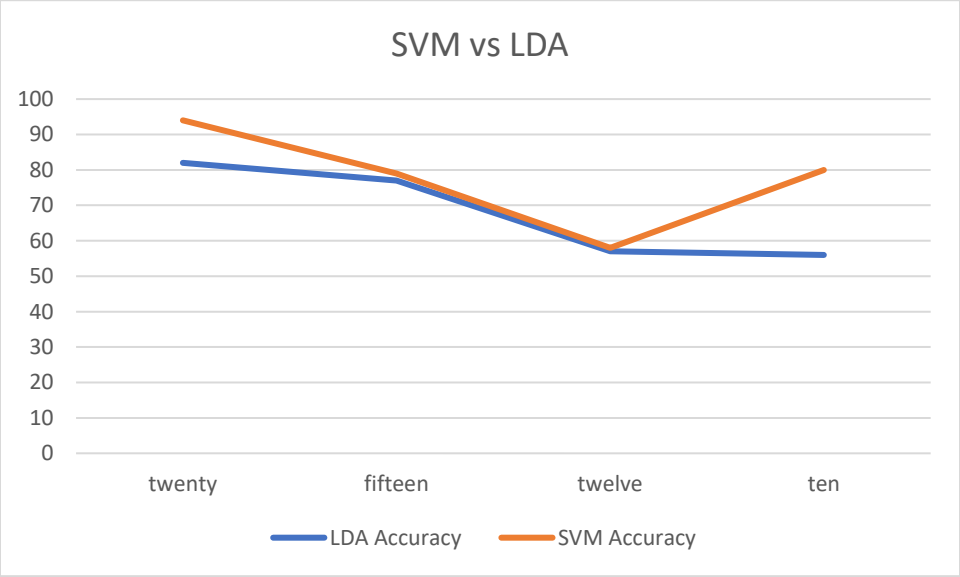

FIGURE 38 SVM ACCURACY COMPARED WITH LDA IN CLASSIFYING STIMULUS FREQUENCIES

To analyse and investigate the reasons why there has been a decline in classifying lower frequencies in our classifiers it is important to look at the training data, the configuration and data acquisition. These have an attribute which defines the goal sequence, the goal sequence was used to indicate which frequency should be used and recorded in the simulation. The goal sequence is shown below:

Goal sequence
: 0 2 3 4 1 2 1 4 0 3 1 4 2 3 4 0 2 0 4 3 1 0 4 3 1 2 4 3 0 1 2 1 4 3 2 0 3 2 0 1 3 4 0 2 1 0 3 1 4 2

Zero shows no stimulation and 1, 2, 3, 4 indicate frequencies 20, 15, 12, 10 respectively.

In the above sequence, there are ten examples for each frequency, therefore the distribution for each frequency was similar, and training data did not favour any specific frequency, and each frequencies had an equal chance to be trained by the classifiers. The test data has the similar attributes as the training data with shorter length, further, as shown in Table 11, SVM had struggled with 15-hertz frequency stimuli and had longer time to succeed and after a few numbers of tries it finally succeed to predict it correctly.

What might improve the classifier's predictions is to apply different filtering and control the DSP with higher factor; this allows the data to be amplified for more and more data to be kept for pattern recognition. For each classifier the number of k-fold could be increased to get more variety of data to be trained.

Chapter 6 – Project Management

Agile methodology was used in managing this project; each week a set of tasks were assigned to be completed and to be tested during the weekend. At the end of testing, the outcomes were analysed, and new set of tasks were made for the following week. Each task is created by considering the time limit of three months to finish the project.

The tasks are first written into a table. The table had its tasks increased over time as new problems emerged. Table 12 show an overview of all the tasks were faced and solved.

TABLE 12 OVERALL TASKS WERE WRITTEN THROUGHOUT THE PROJECT

ID	problems
1	dataset for training the classifier
2	connecting OpenVibe to unity
3	implement the streams to manipulate unity
4	Polish the simple environment
5	Polish the Complex environment
6	polish our results and data gathering
7	Test on VR
8	Test on Screen
9	Complex Environment : Design the Livingroom environment
10	Simple Environment : Design the Livingroom environment
11	Implement appliances in the living room
12	Implement appliances in the bathroom
13	Implement SSVEP stimuli in the living room
14	Allow the user to be moved around the environment using keys
15	Complex Environment : Design the Livingroom environment
16	Complex Environment : Design the Bathroom environment
17	Complex Environment : Design the Bedroom environment
18	make gui stimuli
19	Make an analyser from python to analyse data and show on real-time
20	Find another way to apply SSVEP stimuli with java GUI
21	Make an analyser from python to analyse data and show on real-time
22	OpenVibe :Create Signal tester
23	OpenVibe :Create config settings
24	OpenVibe :Create data acquisition
25	OpenVibe :Create CSP trainer
26	OpenVibe :Create Classifier trainer
27	OpenVibe : Create Main processor
28	OpenVibe :Create Performance measure
29	OpenVibe :Create Confusion matrix
30	Create an experiment on unity
31	Create an experiment on OpenVibe

It was first intended for Jira to be used in the project but there were a technical issue and the account dedicated was not created by the firm and the account was closed to the deadline was received. Therefore, the plans are all organised in EXCEL.

The overall planning is shown in Table 13.

TABLE 13 OVERALL TASKS WERE WRITTEN THROUGHOUT THE PROJECT

Date	Action
22-May	Complex Environment : Design the Livingroom environment
23-May	Simple Environment : Design the Livingroom environment
24-May	Simple Environment : Design the Bedroom environment
25-May	Simple Environment : Design the Bathroom environment
26-May	Implement appliances in the living room
27-May	Implement appliances in the bedroom
28-May	Implement appliances in the bathroom
29-May	Implement SSVEP stimuli in the living room
30-May	Allow the user to be moved around the environment using keys
31-May	Complex Environment : Design the Livingroom environment
1-Jun	Complex Environment : Design the Bedroom environment
2-Jun	Complex Environment : Design the Bathroom environment
3-Jun	make gui stimuli
4-Jun	Meeting
5-Jun	Make an analyser from python to analyse data and show on real-time
6-Jun	Find another way to apply SSVEP stimuli with java GUI
7-Jun	Find another way to apply SSVEP stimuli with java GUI
8-Jun	Find another way to apply SSVEP stimuli with java GUI
9-Jun	Make an analyser from python to analyse data and show on real-time
10-Jun	Find another way to apply SSVEP stimuli with java GUI
11-Jun	Family problem : away
12-Jun	
13-Jun	
14-Jun	
15-Jun	
16-Jun	
17-Jun	
18-Jun	Meeting
19-Jun	Find another way to apply SSVEP stimuli with java GUI
20-Jun	Make an analyser from python to analyse data and show on real-time
21-Jun	OpenVibe :Create Signal tester
22-Jun	OpenVibe :Create config settings
23-Jun	OpenVibe :Create data acquisition

24-Jun	OpenVibe :Create Classifier trainer
25-Jun	OpenVibe :Create CSP trainer
26-Jun	OpenVibe :Create Classifier trainer
27-Jun	OpenVibe : Create Main processor
28-Jun	OpenVibe :Create Perfomance measure
29-Jun	Birthday plan
30-Jun	OpenVibe :Create Confusion matrix
1-Jul	OpenVibe :Create Signal tester
2-Jul	OpenVibe :Create config settings
3-Jul	OpenVibe :Create data acquisition
4-Jul	OpenVibe :Create Classifier trainer
5-Jul	OpenVibe :Create CSP trainer
6-Jul	OpenVibe :Create Classifier trainer
7-Jul	OpenVibe : Create Main processor
8-Jul	OpenVibe :Create Performance measure
9-Jul	Polish the simple environment
10-Jul	Polish the simple environment
11-Jul	Polish the simple environment
12-Jul	Polish the simple environment
13-Jul	Polish the Complex environment
14-Jul	Polish the Complex environment
15-Jul	Polish the Complex environment
16-Jul	Polish the Complex environment
17-Jul	Polish the Complex environment
18-Jul	Polish the Complex environment
19-Jul	Openvibe :Create Signal tester
20-Jul	Openvibe :Create config settings
21-Jul	Openvibe :Create data acquisition
22-Jul	Openvibe :Create CSP trainer
23-Jul	Openvibe :Create Classifier trainer
24-Jul	Openvibe : Create Main processor
25-Jul	Openvibe :Create Performance measure
26-Jul	Openvibe :Create Confusion matrix
27-Jul	Test on VR
28-Jul	Test on Screen
29-Jul	polishing finishes
30-Jul	Create an experiment on unity
31-Jul	Create an experiment on unity
1-Aug	Create an experiment on unity
2-Aug	Create an experiment on Openvibe
3-Aug	Create an experiment on Openvibe

4-Aug	Create an experiment on Openvibe
5-Aug	Create an experiment on Openvibe
6-Aug	Create an experiment on Openvibe
7-Aug	Write up + polishing
8-Aug	
9-Aug	
10-Aug	
11-Aug	
12-Aug	
13-Aug	
14-Aug	
15-Aug	
16-Aug	
17-Aug	
18-Aug	
19-Aug	
20-Aug	
21-Aug	
22-Aug	
23-Aug	
24-Aug	
25-Aug	
26-Aug	
27-Aug	

It was intended to complete one set of tasks each week. At the beginning of the new sprint, the previous tasks' implementations are tested, and a new set of tasks are assigned for the coming week, this cycle was helpful with supervision every Tuesday at a weekly meeting.

Chapter 7 - Conclusions and recommendation

In conclusion, the project was a success with the main experiment raising more questions and the research questions asked were answered. The project has shown, with immersion there may be an improvement with feeling and seeing the world. However, the classifiers performed poorly in classifying the brain signals for each frequency. There can be a few reasons for this; one being that the VR used may have affected the refresh rate which caused the stimulus to flicker incorrectly. The other relates to the comment where the subject explained the VR and the complex environment to be distracting and exciting to look at. This may have caused some distraction in focusing on the stimulus. Furthermore, it has been proven that a simple environment with dull and simple colours affected the experience of the subject which led to better results in classification and performance. Furthermore, SVM has proven that it has more stability than LDA in lower frequencies and has higher accuracy overall.

In the implementation of this project, there are four different languages used, one is Lua that has never been practiced before and was learned during this project. The project has two aspects of BCI and game development and there can be done some improvements; either the environments or the technical side of the BCI where more classifiers can be used or other models to be tested. In the environment, more SSVEP powered appliances can be implemented such as shower, sink, microwave or a complete menu for a washing machine or a dishwasher. For the technical BCI side of the project, more models can be used to improve the flaws explained about the classifier trainers and the old models can be improved by changing the DSP values or change the filtering of signals in pre-processing. The simulation used for data acquisition currently uses an OpenVibe simulation program where it can be replaced with a simulation to make it similar to the environment. The simulation environment is set to have a similar stimuli design as the environment. This method may improve the classifiers where a two-dimensional environment with a black background is quite different from the three-dimensional environment with textures and graphics are dedicated to showing in-world objects.

References

[1] Y. Wang, M. Nakanishi, Y. Te Wang, and T. P. Jung, "Enhancing detection of steady state visual evoked potentials using individual training data," *2014 36th Annu. Int. Conf. IEEE Eng. Med. Biol. Soc. EMBC 2014*, pp. 3037–3040, 2014.

[2] A. T. Berg *et al.*, "Revised terminology and concepts for organization of seizures and epilepsies: Report of the ILAE Commission on Classification and Terminology, 2005-2009," *Epilepsia*, vol. 51, no. 4, pp. 676–685, 2010.

[3] Y. Chae, J. Jeong, and S. Jo, "Toward brain-actuated humanoid robots: Asynchronous direct control using an EEG-Based BCI," *IEEE Trans. Robot.*, 2012.

[4] M. S. Al-Quraishi, I. Elamvazuthi, S. A. Daud, S. Parasuraman, and A. Borboni, "Eeg-based control for upper and lower limb exoskeletons and prostheses: A systematic review," *Sensors (Switzerland)*. 2018.

[5] P. V. Mahrad, "BCI implemented in virtual worlds using VR."

[6] NeuroElectrics, "Neuroelectrics User Manual Enobio," *Neuroelectrics User Man.*, vol. 2.0, pp. 1–36, 2016.

[7] Y. Renard *et al.*, "OpenViBE: An open-source software platform to design, test, and use brain-computer interfaces in real and virtual environments," *Presence Teleoperators Virtual Environ.*, 2010.

[8] Z. Işcan and V. V Nikulin, "Steady state visual evoked potential (SSVEP) based brain-computer interface (BCI) performance under different perturbations," *PLoS One*, vol. 13, no. 1, pp. 1–17, 2018.

[9] F. Lotte, M. Congedo, A. Lécuyer, F. Lamarche, and B. Arnaldi, "A review of classification algorithms for EEG-based brain-computer interfaces," *Journal of Neural Engineering*. 2007.

[10] E. P. Widmaier, H. Raff, and K. T. Strang, "Regulation of Organic Metabolism and Energy Balance," *Vander's Hum. Physiol. Mech. Body Funct.*, 2014.

[11] R. Srinivasan, "Methods to Improve the Spatial Resolution of EEG," *Int. J. Bioelectromagn.*, 1999.

[12] S. Murakami and Y. Okada, "Contributions of principal neocortical neurons to magnetoencephalography and electroencephalography signals," *J. Physiol.*, 2006.

[13] Halliday and Resnick, *Fundamental of Physics 10th Edition*. 2015.

[14] D. Purves, *3,4 Neuroscience Third Edition*. 2004.

[15] Lbonnet, "SSVEP: Steady-State Visual-Evoked Potentials," 2011. [Online]. Available: http://openvibe.inria.fr/steady-state-visual-evoked-potentials/.

[16] A. Heydarian, J. P. Carneiro, D. Gerber, B. Becerik-Gerber, T. Hayes, and W. Wood, "Immersive virtual environments: Experiments on impacting design and human building interaction," *Rethink. Compr. Des. Specul. Counterculture - Proc. 19th Int. Conf. Comput. Archit. Des. Res. Asia, CAADRIA 2014*, no. May, pp. 729–738, 2014.

[17] Robert A. Baron;, Mark S. Rea;, and Susan G. Daniels, "Effects of indoor lighting (illuminance and spectral distribution) on the performance of cognitive tasks and interpersonal behaviors: The potential mediating role of positive affect," *Motiv. Emot.*, 1992.

[18] Y. P. Lin, Y. Wang, C. S. Wei, and T. P. Jung, “Human neuroscience assessing the quality of steady-state visual-evoked potentials for moving humans using a mobile electroencephalogram headset,” *Front. Hum. Neurosci.*, 2014.

[19] G. Lisi, M. Hamaya, T. Noda, and J. Morimoto, “Dry-wireless EEG and asynchronous adaptive feature extraction towards a plug-and-play co-adaptive brain robot interface,” in *Proceedings - IEEE International Conference on Robotics and Automation*, 2016.

[20] R. T. Schirrmeister *et al.*, “Deep learning with convolutional neural networks for EEG decoding and visualization,” *Hum. Brain Mapp.*, 2017.

[21] V. J. Lawhern, A. J. Solon, N. R. Waytowich, S. M. Gordon, C. P. Hung, and B. J. Lance, “EEGNet: A compact convolutional neural network for EEG-based brain-computer interfaces,” *J. Neural Eng.*, 2018.

[22] N. S. Kwak, K. R. Müller, and S. W. Lee, “A convolutional neural network for steady state visual evoked potential classification under ambulatory environment,” *PLoS One*, 2017.

[23] J. Thomas, T. Maszczyk, N. Sinha, T. Kluge, and J. Dauwels, “Deep learning-based classification for brain-computer interfaces,” in *2017 IEEE International Conference on Systems, Man, and Cybernetics, SMC 2017*, 2017.

[24] A. L. Goldberger *et al.*, “PhysioBank, PhysioToolkit, and PhysioNet: components of a new research resource for complex physiologic signals.,” *Circulation*, 2000.

[25] C. Guger, C. Holzner, C. Grönegress, G. Edlinger, and M. Slater, “Brain-computer interface for virtual reality control,” in *ESANN 2009 Proceedings, 17th European Symposium on Artificial Neural Networks - Advances in Computational Intelligence and Learning*, 2009.

[26] C. G. Coogan and B. He, “Brain-Computer Interface Control in a Virtual Reality Environment and Applications for the Internet of Things,” *IEEE Access*, vol. 6, pp. 10840–10849, 2018.

[27] Unity Technology, “Unity 3D,” *Unity Technology*, 2018. .

[28] Mohelm97, “Simple Home Stuff.” .

[29] A. V. Education, “Water Fx Particles.” [Online]. Available: https://assetstore.unity.com/packages/vfx/particles/environment/water-fx-particles-48580.

[30] OPTIMESH, “Croissants Pack.” [Online]. Available: https://assetstore.unity.com/packages/3d/props/food/croissants-pack-112263.

[31] ARCHVIZPRO, “ArchVizPRO Interior Vol.2.” [Online]. Available: https://assetstore.unity.com/packages/3d/environments/archvizpro-interior-vol-2-50665.

[32] S. KROKIDANA, “Kitchen Creation Kit.” [Online]. Available: https://assetstore.unity.com/packages/3d/environments/kitchen-creation-kit-2854.

[33] 3DIZ-ART, “Table with chairs x3 Free.” [Online]. Available: https://assetstore.unity.com/packages/3d/props/furniture/table-with-chairs-x3-free-101246.

[34] S. MAREVOY, “Realistic Furniture and Interior Props Pack.” [Online]. Available: https://assetstore.unity.com/packages/3d/props/furniture/realistic-furniture-and-interior-props-pack-120379.

[35] Oculus, “Oculus rift s.” [Online]. Available: https://www.oculus.com/rift-s/?locale=en_GB.

Appendices

Unity Side

CanvasScript.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class CanvasScript : MonoBehaviour
{
    public Camera camera;

    // Start is called before the first frame update
    void Start()
    {
        camera = GameObject.FindGameObjectWithTag("MainCamera").GetCompo-
nent<Camera>();

    }

    // Update is called once per frame
    void Update()
    {
        transform.LookAt(transform.position - camera.transform.rotation *
Vector3.back, camera.transform.rotation * Vector3.up);

    }


    public GameObject Create_buttons(GameObject b, buttonsconfig bc)
    {

        GameObject childobj = Instantiate(b,gameObject.transform) as
GameObject;

        childobj.GetComponentInChildren<Text>().text = bc.text;



        childobj.transform.SetParent( this.gameObject.transform);

        return childobj;

    }
}
```

Experiment.cs

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.IO;
using UnityEngine;


public class Records
{
    public float Timer;
    public Direction button_direction;
    public Stim_code stim_Code;
    public float HZ;
    public int total_Num_tries;
    public int successful_tries;
    public bool achieved;

    public Records(Direction direction, float HZ)
    {
        Timer = 0;

        this.button_direction = direction;

        achieved = false;
        this.HZ = HZ;
        this.total_Num_tries = 0;
        check_stimCode();

    }

    void check_stimCode()
    {
        switch (button_direction)
        {

            case Direction.down:
                stim_Code = Stim_code.SL3;
                break;
            case Direction.left:
                stim_Code = Stim_code.SL1;
                break;
            case Direction.right:
                stim_Code = Stim_code.SL4;
                break;
            case Direction.Up:
                stim_Code = Stim_code.SL2;

                break;
        }
    }

        public void TimerUpdate()
    {
```

```
        Timer += Time.deltaTime;

    }
    public void increase_tries()
    {

        total_Num_tries++;
    }
    public void increase_ST()
    {

        successful_tries++;
    }
    }
public enum ClassifierType
{
       LDA = 1,
       SVM = 2,
       NN = 3
}
public enum EnvironmnetType
{
    SimpleEnvironmnet = 1,
    ComplexEnvironmnet = 2
}
public enum DisplayType
{
    VR = 1,
    Screen = 2
}

public class Experiment : MonoBehaviour
{

    private List<Records> records;
    private string filename;
    private int subj_num = 1;
    private int[] Hzs = { 20, 15, 12, 10 };
    public bool Is_debugging = false;
    public bool operation = true;
    public bool next_exp;
    public EnvironmnetType environmnetType;
    public ClassifierType classifierType;
    public DisplayType displayType;

    // Start is called before the first frame update
    void Start()
    {
        records = new List<Records>();
        filename = "Subject_";

    }
    int called = 0;
```

```
   private Records experiment_cheker( )
    {
        Records newrec = null;

        switch (called)
        {

            case 1:
                newrec = new Records(Direction.left, Hzs[called-1]);
                Debug.Log("Starting to record, Please look at the button
on the left");
                break;

            case 2:
                newrec = new Records(Direction.Up, Hzs[called-1]);
                Debug.Log("Starting to record, Please look at the button
on the top");
                break;
            case 3:
                newrec = new Records(Direction.right, Hzs[called-1]);
                Debug.Log("Starting to record, Please look at the button
on the right");
                break;




        }

        if(called > 3)
        {


        }

        return newrec;


    }


    public Records Start_experiment(Records currentrec)
    {

        if(called > 0 && currentrec != null )
        {
            add_records(currentrec);

        }
            called++;

        if (called > 3)
        {
            Export_to_file();
```

```
            operation = false;

        }
        if (operation)
        {
            StartCoroutine(waitTime(3));

            Records newrec = experiment_cheker();

            return newrec;
        }
        else
        {
            return null;
        }


    }

    IEnumerator waitTime(float seconds)
    {
        yield return new WaitForSeconds(seconds);
    }
    void add_records(Records rec)
    {

        records.Add(rec);




    }

    void Export_to_file()
    {
        string filename2 = displayType.ToString()+"-"+environmnet-
Type.ToString()+"-"+classifierType.ToString()+"-"+filename +
subj_num+".csv";
        StreamWriter outStream = System.IO.File.CreateText(filename2);
        outStream.WriteLine(lineSeperater + "Button Direction" +
fieldSeperator + "Frequency" + fieldSeperator + "Successful tries" +
fieldSeperator + "Total tries" + fieldSeperator + "Duration till action
performed");
        foreach (Records i in records)
        {

            write_into_csv(i, outStream);

        }
        outStream.Close();

    }
```

```
    private char lineSeperater = '\n';
    private char fieldSeperator = ',';
    void write_into_csv(Records record, StreamWriter outStream)
    {
        try
        {


            string filename2 = filename+ subj_num;

            if (Is_debugging)
            {
                Debug.Log("Starting to write into the file " + filename2);

            }
            outStream.WriteLine(lineSeperater + record.button_direc-
tion.ToString() + fieldSeperator + record.HZ + fieldSeperator + rec-
ord.successful_tries + fieldSeperator + record.total_Num_tries +
fieldSeperator + record.Timer);

            //TextAsset file = Resources.Load<TextAsset>(filename2);
           // Debug.Log(lineSeperater + record.button_direction.ToString()
+ fieldSeperator + record.HZ + fieldSeperator + record.successful_tries +
fieldSeperator + record.total_Num_tries + fieldSeperator + record.Timer);

            //File.AppendAllText(getPath()  + filename2 + ".txt", lineSe-
perater + record.button_direction.ToString()+fieldSeperator+ record.HZ
+fieldSeperator+ record.successful_tries+ fieldSeperator +record.to-
tal_Num_tries+fieldSeperator+ record.Timer);


#if UNITY_EDITOR
            UnityEditor.AssetDatabase.Refresh();
#endif

        }

        catch (Exception e)
        {

            Debug.Log("could not write into file" + filename+ subj_num+
e.Message);



        }


    }
    private static string getPath()
    {
#if UNITY_EDITOR
        return Application.dataPath;
#elif UNITY_ANDROID
```

```
return Application.persistentDataPath;
#elif UNITY_IPHONE
return GetiPhoneDocumentsPath();
#else
return Application.dataPath;
#endif
    }
    // Update is called once per frame
    void Update()
    {

    }
}
```

Hover_UI

```
using System.Collections;
using System.Collections.Generic;
using System.Diagnostics;
using UnityEngine;
using UnityEngine.UI;

public enum Direction { Up, down, left, right}
public enum LightAction { Red,Blue,Green,OFF_ON, Yellow, Magneta, Cyan ,
Unknown }

[System.Serializable]
public class buttonsconfig
{

    public Direction direction;
    public LightAction action;
    public float HZ;
    public string text = "";
    [System.NonSerialized]
    public float dt = 0;
    [System.NonSerialized]
    public float frame = 0;
    [System.NonSerialized]
    public Button stimuli;
    public double change_time = 0;

}

public class Hover_UI : MonoBehaviour
{
    private int LS_frame = 0, CS_frame = 0;
    private float next_sec;
    private Stim_code stim;
    private Tcp_manager tcp;
    public buttonsconfig[] buttons;
    public bool Java_integrated;
    public string ObjString;
    public Text ObjText;
    public float f_time;
    public bool Is_shown;
```

```
    Utilities Util;
    Experiment experiment;
    public bool is_experimenting;
    private Vector3 Original_pos;
    private float distance = 2f;
    float speed = 2f;
    float error = 0.1f;
    private bool flash = false;
    private Process foo;
    public Button ButtonPrefab;
    public GameObject light;

    //shoulnt be multiple
    int[] frame_delay = { 10, 26 };
    private bool running = false;

    // TODO : actual value of the frequency must be checked. 10 seconds,
BUILD TOO
    // For more than 2 flashings:
    // one way is to use a separate screen
    // 3 hrtz and ghost images with multiples....
    // Start is called before the first frame update
    void Start()
    {
        next_sec = (Time.time * 1000) + 1000;
        tcp = GameObject.Find("GameScripts").GetComponent<Tcp_manager>();

        if (light == null)
        {
            light = gameObject.transform.GetChild(0).gameObject;




        }
        Util = GameObject.FindGameObjectWithTag("Util").GetComponent<Util-
ities>();
        experiment = GameObject.FindGameObjectWithTag("Util").GetCompo-
nent<Experiment>();
        experiment.next_exp = true;
        ObjText = GameObject.Find("Text").GetComponent<Text>();
        ObjText.color = Color.clear;

        CanvasScript c = gameObject.GetComponentInChildren<CanvasScript>()
;

        Original_pos = c.transform.position;
        foo = new Process();
        foo.StartInfo.FileName = "C:/Users/Mahrad/Desk-
top/CE901/ce901_pisheh_var_m/Flasher/out/arti-
facts/Flasher_jar/Flasher.jar";
        foo.StartInfo.Arguments = "2 15";
        foo.StartInfo.WindowStyle = ProcessWindowStyle.Normal;
```

```
        foreach (buttonsconfig b in buttons)
        {


            b.stimuli = c.Create_buttons(ButtonPrefab.gameObject, b).Get-
Component<Button>();


        }


    }


    public void increase_dt()
    {
        foreach (buttonsconfig s in buttons)
        {
            s.dt += Time.deltaTime;

        }

    }


    private bool Stim_checker(int i)
    {
        if (i <= buttons.Length)
        {
            return true;

        }
        else return false;


    }

    private void if_exp_increase_ST(Stim_code stim)
    {

        if (is_experimenting)
        {
            if (currentrec != null)
            {
                currentrec.increase_tries();
                if (currentrec.stim_Code.Equals(stim))
                {
                    currentrec.increase_ST();
                    currentrec.achieved = true;


                }
            }
```

```
    }


}
private LightAction action_from_tcp(Stim_code s)
{
    LightAction action = LightAction.Unknown;
    switch (s)
    {
        case Stim_code.Unknown:
            break;

        case Stim_code.SL1:

            if_exp_increase_ST(s);
            if (Stim_checker(1)) action = buttons[0].action;
            break;
        case Stim_code.SL2:

            if_exp_increase_ST(s);
            if (Stim_checker(2)) action = buttons[1].action;
            break;
        case Stim_code.SL4:

            if_exp_increase_ST(s);
            if (Stim_checker(3)) action = buttons[2].action;
            break;




        case Stim_code.SL3:
            action = LightAction.Unknown;
            break;

        case Stim_code.SL5:
            action = LightAction.Unknown;

            break;
        case Stim_code.SL6:
            action = LightAction.Unknown;

            break;
        case Stim_code.SL7:
            action = LightAction.Unknown;

            break;
        default:
            action = LightAction.Unknown;
            break;

    }
```

```
    return action;
}



public void  turn_light()
{
    if (light.activeSelf)
    {
        light.SetActive(false);
    }
    else
    {
        light.SetActive(true);
    }
}
public void lit_action(LightAction action)
{

    switch (action)
    {
        case LightAction.Red:
            light.GetComponent<Light>().color = Color.red;

            break;
        case LightAction.Blue:
            light.GetComponent<Light>().color = Color.blue;

            break;
        case LightAction.Green:
            light.GetComponent<Light>().color = Color.green;

            break;
        case LightAction.OFF_ON:
            turn_light();
            break;
        case LightAction.Yellow:
            light.GetComponent<Light>().color = Color.yellow;

            break;
        case LightAction.Magneta:
            light.GetComponent<Light>().color = Color.magenta;

            break;
        case LightAction.Cyan:
            light.GetComponent<Light>().color = Color.cyan;

            break;
        case LightAction.Unknown:
            break;
    }
}
```

```
    Records currentrec= null;
    public void if_flash()
    {

        if (flash)
        {

            if (is_experimenting )
            {
                if (experiment.next_exp)
                {
                    currentrec = experiment.Start_experiment(currentrec);
                    if (currentrec == null) { is_experimenting = false;
return; }
                    experiment.next_exp = false;
                }

                else{

                    if (currentrec.Timer > 50 || currentrec.achieved) {
experiment.next_exp = true; currentrec.achieved = false; }
                }
                currentrec.TimerUpdate();
            }



            stim = tcp.mymessage;

            lit_action(action_from_tcp(stim));


            foreach (buttonsconfig s in buttons)
            {
                s.frame += Time.deltaTime;

                // button_flash_fixed(s); //Frame Base flickering
                button_flash_based_on_time(s); //Time based flickering

            }



        }


    }
    // Update is called once per frame
    void Update()
    {

        Fade_Func();
        increase_dt();
        if_flash();
```

```
}


private void button_flash_based_on_time(buttonsconfig b)
{

    float when_to_flash = b.HZ;
    when_to_flash = 1 / (when_to_flash * Time.deltaTime);

    float current_T = Time.realtimeSinceStartup;

    if (current_T >= b.change_time)
    {

        b.change_time += (current_T + when_to_flash);
        b.stimuli.enabled = !b.stimuli.enabled;
    }


}
private void button_flash_fixed(buttonsconfig b)
{


    if(b.frame >= 1/b.HZ)
    {

        b.stimuli.enabled = !b.stimuli.enabled;
        b.frame = 0;

    }




}


private void button_flash(buttonsconfig b)
{

    if(b.dt >= b.HZ / Util.get_fps())
    {

        b.stimuli.enabled = !b.stimuli.enabled;
        b.dt = 0;

    }



}
public void Selected()
```

```
    {
        Is_shown = true;

    }
    public void notSelected()
    {
        Is_shown = false;
    }
    private void OnMouseOver()
    {


            Is_shown = true;



    }

    private void OnMouseUp()
    {
        if (Java_integrated)
        {

            foo.Start();

        }

    }

        private void OnMouseExit()
    {


            Is_shown = false;


    }


    private void buttons_controller(buttonsconfig b)
    {




            switch (b.direction)
            {
                case Direction.Up:
                    b.stimuli.transform.position += b.stimuli.transform.up
* Time.deltaTime * speed;
                    break;
                case Direction.down:
                    b.stimuli.transform.position -= b.stimuli.transform.up
* Time.deltaTime * speed;
```

```
                    break;
                case Direction.left:
                    b.stimuli.transform.position -= b.stimuli.trans-
form.right * Time.deltaTime * speed;
                    break;
                case Direction.right:
                    b.stimuli.transform.position += b.stimuli.trans-
form.right * Time.deltaTime * speed;
                    break;
            }



    }

    private void buttons_change_color()
    {

        if (Java_integrated)
        {

            dontflash();
        }
        else
        {

            if (Is_shown)
            {
                ObjText.text = ObjString;
                ObjText.color = Color.white;
                foreach (buttonsconfig b in buttons)
                {
                    b.stimuli.enabled = true;
                    b.stimuli.GetComponentInChildren<Text>().color =
Color.Lerp(ObjText.color, Color.white, f_time * Time.deltaTime);

                    if (Vector3.Distance(Original_pos, b.stimuli.trans-
form.position) < distance - error)
                    {
                        buttons_controller(b);
                    }
                    else
                    {
                        flash = true;
                    }
                }


            }
            else
            {

                dontflash();
```

```
            }
        }
    }

    private void  dontflash()
    {
        foreach (buttonsconfig b in buttons)
        {
            b.stimuli.enabled = false;
            b.stimuli.GetComponent<Image>().CrossFadeAlpha(0, f_time *
Time.deltaTime, false);
            b.stimuli.GetComponentInChildren<Text>().color =
Color.Lerp(ObjText.color, Color.clear, f_time * Time.deltaTime);
            if (Vector3.Distance(Original_pos, b.stimuli.transform.posi-
tion) > 0.5f)
            {

                move(b.stimuli, 0.5f);
            }



            if (Vector3.Distance(Original_pos, b.stimuli.transform.posi-
tion) > 0.5f)
            {

                move(b.stimuli, 0.5f);
            }
        }
        ObjText.color = Color.Lerp(ObjText.color, Color.clear, f_time *
Time.deltaTime);
        flash = false;
    }
        private void Fade_Func()
    {


        buttons_change_color();
    }


    private void move(Button targetButton, float duration)
    {

        float elapsed = 0;
        Vector3 start_pos = targetButton.transform.position;

        while (elapsed < duration)
        {
            targetButton.transform.position = Vector3.Lerp(start_pos,
Original_pos, elapsed);
            elapsed += Time.deltaTime;
```

```
        }

        targetButton.transform.position = Original_pos;



    }




    }
```

MouseOver.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MouseOver : MonoBehaviour {

      // Use this for initialization

      public GameObject light;
      void Start () {

            if (light == null)
            {
                  light = gameObject.transform.GetChild(0).gameObject;




            }

      }

      // Update is called once per frame
      void Update () {

      }

      private void OnMouseUp()
      {

            if (light.activeSelf)
            {
                  light.SetActive(false);
            }
            else
            {
```

```
                light.SetActive(true);
            }


      }
}
```

Tcp_manager.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading;
using System.Net.Sockets;
using System.Text;
using System;

public enum Stim_code
{
    Unknown = 0,
    SL0  = 33024,
    SL1 = 33025,
    SL2 = 33026,
    SL3 = 33027,
    SL4 = 33028,
    SL5 = 33029,
    SL6 = 33030,
    SL7 = 33031,
}

public class Tcp_manager : MonoBehaviour
{

    public int PORT = 5678;
    private Thread CRT; // client recieved thread
    private TcpClient tcpClient;
    public Stim_code mymessage;
    public bool is_debugging = true;

    // Start is called before the first frame update
    void Start()
    {
        connect();
    }

    // Update is called once per frame
    void Update()
    {

    }
    private void OnDestroy()
    {
        tcpClient.Close();
        CRT.Abort();
    }
```

```
private void connect()
{

    try
    {

        CRT = new Thread(new ThreadStart(listening));
        CRT.IsBackground = true;
        CRT.Start();
        if (is_debugging)
        Debug.Log("Client--> Successfully connected");
    }

    catch (Exception e) {

        if (is_debugging)
            Debug.Log("Client--> Cannot connect");


    }


}

private Stim_code message_converter(int code)
{

    Stim_code a;

    if (Enum.IsDefined(typeof(Stim_code), code))
        a = (Stim_code)code;
    else
        a = Stim_code.Unknown;

    return a;
}
private void RecieveBytes(Byte[] b)
{
    try
    {
        using (NetworkStream s = tcpClient.GetStream())
        {
            int l;

            while ((l = s.Read(b, 0, b.Length)) != 0)
            {
                Debug.Log("connected");
                var data = new byte[l];
                Array.Copy(b, 0, data, 0, l);


                int code = BitConverter.ToInt32(data, 0);
```

```
                    mymessage = message_converter(code);
                    if (is_debugging)
                        Debug.Log("Message recieved: " + mymessage);


                      }
            }

        }
        catch (Exception e)
        {

                Debug.Log("Client--> Cannot recieve bytes"+ e);


        }

    }
    private void listening()
    {
        try
        {
            tcpClient = new TcpClient("localhost", PORT);

            Byte[] b = new Byte[1024];
            if (is_debugging)
                Debug.Log("Client-->  listening to localhost at port: "+
PORT);
            while (true)
            {
                RecieveBytes(b);
                if (tcpClient.Connected)
                {
                    Debug.Log("connected");
                }


            }



        }
        catch(SocketException e)
        {
            if (is_debugging)
                Debug.Log("Socket exception occured: check :>" + e);
        }


    }
}
```

Utilities.cs

```
using System.Collections;
```

```
using System.Collections.Generic;
using UnityEngine;

public class Utilities : MonoBehaviour
{
    int frameCount = 0;
    float dt, fps = 0;
    float rate = 4;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        frameCount++;
        dt += Time.deltaTime;
        if(dt > 1.0 / rate)
        {
            fps = frameCount / dt;
            frameCount = 0;
            dt -=  1 / rate;

        }

    }
    public float get_fps()
    {
        return fps;
    }
    public float get_frameCount()
    {
        return frameCount;
    }
}
```

VR_Mouse_replacement.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class VR_Mouse_replacement : MonoBehaviour
{

    [SerializeField] private string tag = "Stimuli_obj";
    // Start is called before the first frame update

    private Transform t;

    void Start()
    {
```

```
    }

    // Update is called once per frame
    void Update()
    {


        if (t != null)
        {
            t.GetComponent<Hover_UI>().notSelected();
            t = null;

        }
        var ray = Camera.main.ScreenPointToRay(this.transform.position);
        RaycastHit hit;

        if (Physics.Raycast(ray, out hit))
        {
            var s = hit.transform;
            if (s.CompareTag(tag))
            {
                var sr = s.GetComponent<Renderer>();
                if (sr != null)
                {
                    s.GetComponent<Hover_UI>().Selected();

                }
                t = s;
            }
        }

    }
}
```

BCI side

classifier-training-flipswitch.lua

```
f_count = 0
switched_count = 0
f_list = {}


Generated_file_path = "/plugins/stimulation/lua-stimulator-stim-codes.lua"
function initialize(box)
      dofile(box:get_config("${Path_Data}") .. Generated_file_path)

      f_count = box:get_input_count()

      for i = 1, f_count do
            f_list[i] = false
      end
end
```

```
function uninitialize(box)
end


function write_flip(i, j)

      io.write("Flip ", i, " of ", j, " switched\n")

end
function stimulation_alter(box)


      while box:keep_processing() and switched_count < f_count do
            for i = 1, f_count do

                  if box:get_stimulation_count(i) > 0 then

                        box:remove_stimulation(i,1)

                        if not f_list[i] then

                              f_list[i] = true
                              switched_count = switched_count +1

                              write_flip(i,f_count)

                        end
                  end
            end
            box:sleep()

      end
end
function process(box)

      stimulation_alter(box)
      box:send_stimulation(1, OVTK_StimulationId_Label_00, box:get_cur-
rent_time())
end
```

classifier-training-target-separator.lua

```
Element_target = {}
non_Element_target = {}
stimulation_check_number = 0
generated_file_path = "/plugins/stimulation/lua-stimulator-stim-codes.lua"

function initialize(box)
      dofile(box:get_config("${Path_Data}") .. generated_file_path)
```

```
	get_settings(box)

end

function get_settings(box)


	s_non_Element_target = box:get_setting(3)

	for i in s_non_Element_target:gmatch("%d+") do
		non_Element_target[i + 0] = true
	end

	s_Element_target = box:get_setting(2)
	for i in s_Element_target:gmatch("%d+") do
		Element_target[i + 0] = true
	end


	stimulation_check_number = _G[box:get_setting(4)]



end


function uninitialize(box)
end

function process(box)

	is_finished = false

	while  not is_finished and box:keep_processing()  do

		current_time = box:get_current_time()

		while box:get_stimulation_count(1) > 0 do

			stimulation_code, stimulation_date, stimulation_duration
= box:get_stimulation(1, 1)
			box:remove_stimulation(1, 1)

			if stimulation_code >= OVTK_StimulationId_Label_00 and
stimulation_code <= OVTK_StimulationId_Label_1F then

				received_stimulation = stimulation_code -
OVTK_StimulationId_Label_00

				if Element_target[received_stimulation] ~= nil then
					box:send_stimulation(1, stimula-
tion_check_number, current_time)
				elseif non_Element_target[received_stimulation] ~=
nil then
```

```
                              box:send_stimulation(2, stimula-
tion_check_number, current_time)
                        end

                elseif stimulation_code == OVTK_StimulationId_Experiment-
Stop then
                        is_finished = true
                end
            end

            box:sleep()

      end

end
```

Experiment_settings.lua

```
S_freq = {}
Frequency_NUM = 0

Target_L_Color = {}
Target_D_Color = {}
target_training_Cap = {}
target_training_pos = {}

epoch_time = nil
epoc_delay = nil
freq_tolerance = nil
CONSTANT_LUA_GEN =  "/plugins/stimulation/lua-stimulator-stim-codes.lua"

Chs = nil

function initialize(box)



      dofile(box:get_config("${Path_Data}") ..CONSTANT_LUA_GEN)

      Get_settings(box)



end

function uninitialize(box)
end

--Main process function
function process(box)

      while  box:get_stimulation_count(1) == 0 and box:keep_processing()
do
```

```
            box:sleep()
      end

      box:log("Info", box:get_config("Running Configuration for the exper-
iment"))
      box:log("Info", box:get_config("Writing..... to  '${CustomConfigura-
tionPrefix${OperatingSystem}}-ssvep-demo${CustomConfigurationSuffix${Oper-
atingSystem}}'"))

      if (Write_Config_to_file(box) == false) then return false end
      if (Temporal_config(box) == false)  then return false end
      if (Time_based_epoch(box) == false) then return false end
      if(Channel_selector(box) == false) then return false end




      -- notify the scenario that the configuration process is complete
      box:send_stimulation(1, OVTK_StimulationId_TrainCompleted,
box:get_current_time() + 0.2, 0)

end

-- Get the Values from the table and input --
function Get_settings(box)

      epoch_time = box:get_setting(5)
      epoc_delay = box:get_setting(6)
      freq_tolerance = box:get_setting(7)
      Chs  = box:get_setting(8)

      for i in box:get_setting(4):gmatch("%d+[.]?%d*") do
            table.insert(S_freq, i)
            Frequency_NUM = Frequency_NUM + 1
      end
            for i in box:get_setting(3):gmatch("%d+") do
            table.insert(Target_D_Color, i)
      end
      for i in box:get_setting(2):gmatch("%d+") do
            table.insert(Target_L_Color, i)
      end


end

-- Write to exp config file
function Write_Config_to_file(box)
      myFile = assert(io.open(box:get_config("${CustomConfigurationPre-
fix${OperatingSystem}}-ssvep-demo${CustomConfigurationSuffix${Operat-
ingSystem}}"), "a"))

      success = true
```

```
	success = success and myFile:write("SSVEP_TargetLightColourRed = ",
Target_L_Color[1] / 100, "\n")
	success = success and myFile:write("SSVEP_TargetLightColourGreen =
", Target_L_Color[2] / 100, "\n")
	success = success and myFile:write("SSVEP_TargetLightColourBlue = ",
Target_L_Color[3] / 100, "\n")
	success = success and myFile:write("SSVEP_TargetDarkColourRed = ",
Target_D_Color[1] / 100, "\n")
	success = success and myFile:write("SSVEP_TargetDarkColourGreen = ",
Target_D_Color[2] / 100, "\n")
	success = success and myFile:write("SSVEP_TargetDarkColourBlue = ",
Target_D_Color[3] / 100, "\n")

	for i=1,Frequency_NUM do
		success = success and myFile:write("SSVEP_Frequency_", i, " =
", string.format("%g", S_freq[i]), "\n")
	end

	myFile:close()

	if (success == false) then
		box:log("Error", box:get_config("Couldnt Write  '${CustomCon-
figurationPrefix${OperatingSystem}}-ssvep-demo${CustomConfigurationSuf-
fix${OperatingSystem}}'  config file" ))
		return false
	else  return true
	end

end

-- create config files for temporal filters
function Temporal_config(box)




	scenario_path = box:get_config("${Player_ScenarioDirectory}")

	box:log("Info", "Frequency Count : '" ..Frequency_NUM.. "'")

	for i=1,Frequency_NUM do
		myFile_name = scenario_path .. string.format("/configura-
tion/temporal-filter-freq-%d.cfg", i)

		myFile = io.open(myFile_name, "w")

		if myFile == nil then
			box:log("Error", "Cannot write to [" .. myFile_name ..
"]")
			return false
		end
			box:log("Info", "Stimulation Frequency:
'"..S_freq[i].."'")
```

```
      box:log("Info", " tolerance against the Stimulation freq =
'"..tostring(S_freq[i] - freq_tolerance).."'")
            success = true
            success = success and myFile:write("<OpenViBE-SettingsOver-
ride>\n")
            success = success and myFile:write("<SettingValue>Butter-
worth</SettingValue>\n")
            success = success and myFile:write("<SettingValue>Band
pass</SettingValue>\n")
            success = success and myFile:write("<SettingValue>4</Set-
tingValue>\n")
            success = success and myFile:write(string.format("<Set-
tingValue>%g</SettingValue>\n", tostring(S_freq[i] - freq_tolerance)))
            success = success and myFile:write(string.format("<Set-
tingValue>%g</SettingValue>\n", tostring(S_freq[i] + freq_tolerance)))
            success = success and myFile:write("<Set-
tingValue>0.500000</SettingValue>\n")
            success = success and myFile:write("</OpenViBE-SettingsOver-
ride>\n")

            myFile:close()

            if (success == false) then
                  box:log("Error", box:get_config("Write error"))
                  return false
            end

      myfilename = "${Player_ScenarioDirectory}/configuration/temporal-
filter-freq-%dh1.cfg"
      myFile = assert(io.open(string.format(box:get_config( myfilename) ,
i ), "w"))

            myFile:write("<OpenViBE-SettingsOverride>\n")
            myFile:write("<SettingValue>Butterworth</SettingValue>\n")
            myFile:write("<SettingValue>Band pass</SettingValue>\n")
            myFile:write("<SettingValue>4</SettingValue>\n")
            myFile:write(string.format("<SettingValue>%s</Set-
tingValue>\n", tostring(S_freq[i] * 2 - freq_tolerance)))
            myFile:write(string.format("<SettingValue>%s</Set-
tingValue>\n", tostring(S_freq[i] * 2 + freq_tolerance)))
            myFile:write("<SettingValue>0.500000</SettingValue>\n")
            myFile:write("</OpenViBE-SettingsOverride>\n")

            myFile:close()

            box:log("Info", "Writing into the file [" ..myfilename..."]")

      end

end

-- create configuration file for time based epoching--

function Time_based_epoch(box)
```

```
	myFile_name = scenario_path .. "/configuration/time-based-epoch-
ing.cfg";

	box:log("Info", "Writing file '" .. myFile_name .. "'")

	myFile = io.open(myFile_name, "w")
	if myFile == nil then
		box:log("Error", "Cannot write to [" .. myFile_name .. "]")
		return false
	end

	success = true
	success = success and myFile:write("<OpenViBE-SettingsOverride>\n")
	success = success and myFile:write(string.format("<Set-
tingValue>%g</SettingValue>\n", tostring(epoch_time)))
	success = success and myFile:write(string.format("<Set-
tingValue>%g</SettingValue>\n", tostring(epoc_delay)))
	success = success and myFile:write("</OpenViBE-SettingsOverride>\n")

	myFile:close()

	if (success == false) then
		box:log("Error", box:get_config("Write error"))
		return false
	else  return true end
end

function Channel_selector(box)

	myfilename = scenario_path .."/configuration/channel-selector.cfg"
		box:log("Info", "Writing file '" .. myfilename .. "'")

	myFile = io.open(myfilename, "w")

	success = true

	success = success and myFile:write("<OpenViBE-SettingsOverride>\n")
	success = success and myFile:write(string.format("<Set-
tingValue>%s</SettingValue>\n", Chs))
	success = success and myFile:write(string.format("<Set-
tingValue>%s</SettingValue>\n", "Select"))
	success = success and myFile:write(string.format("<Set-
tingValue>%s</SettingValue>\n", "Smart"))
	success = success and myFile:write("</OpenViBE-SettingsOverride>\n")

	myFile:close()

	if(success == false) then
	box:log("Error", "Writing into File [..myfilename.."]" )

	return false
	else return true
```

```
	end



end
```

Monitor_settings.lua

```
vrpn_host = nil
vrpn_port = nil
--Screen Referesh rate--
Screen_RF = nil

file_dir = "${CustomConfigurationPrefix${OperatingSystem}}-ssvep-
demo${CustomConfigurationSuffix${OperatingSystem}}"
CONSTANT_LUA_GEN =  "/plugins/stimulation/lua-stimulator-stim-codes.lua"



function initialize(box)
	dofile(box:get_config("${Path_Data}") ..CONSTANT_LUA_GEN)
	Screen_RF = box:get_setting(2)
end

function uninitialize(box)
end

function process(box)

	box:log("Info", box:get_config("Generating ".. file_dir))

	myfile = assert(io.open(box:get_config(file_dir), "w"))



	if (loadFile(myfile,box) == false) then

		box:log("Error", box:get_config("Couldnt Write"))
	end

	myfile:close()

	box:send_stimulation(1, OVTK_StimulationId_TrainCompleted,
box:get_current_time() + 0.2, 0)
end



function loadFile(myfile,box)
```

```
      box:log("Info", "Attempting to write in the SSVEP_Screen-
RefreshRate.config")
      success = true
      success = success and myfile:write("SSVEP_ScreenRefreshRate = ",
Screen_RF, "\n")

      if(success == false) then

            return false

      else
            return true

      end

end
```

python-confusion-matrix.py

```
import pickle
import collections
import numpy as np
from sklearn.metrics import confusion_matrix

class mod_OVBox(OVBox):


    def __init__(self):
        OVBox.__init__(self)

        self.time_predicted = 0
        self.gen_label = False
      #meanDetectTimeNewStim
        self.mean_dtns = 0
        self.num_T_class = 0
        self.is_debugged = False
        self.thresh_of_class = 0.5
        self.maximum_probility_diff_in_Thresh = 0.25
        self.debug_pred_nothing = False
        self.neg_class = 0

        self.time_stop_cl = 0
        self.time_start_cl = 0
        self.cl = -1

        deque_max_lenght = 1
        self.class_probablities = [collec-
tions.deque(maxlen=deque_max_lenght),
                           collections.deque(maxlen=deque_max_lenght),
                           collections.deque(maxlen=deque_max_lenght),
                           collections.deque(maxlen=deque_max_lenght)]

        self.labels_prob_act = []
```

```
        self.labels_prob_pred = []
        self.num_stims_class = 0
        self.num_stims_act = 0




    def initialize(self):
        thresh_string_list = self.setting['Thresholds']
        thresh_list = thresh_string_list.split(':')
        if thresh_list:
            self.maximum_probility_diff_in_Thresh = float(thresh_list[1])
            self.thresh_of_class = float(thresh_list[0])



    def add_to_matrix(self, num_input, num_class):
        probMatrix = self.input[num_input].pop()  # id_probMatrix = 0
stimulated, id_Matrix = 1 is non-stimulated
        self.class_probablities[num_class].append([probMatrix[0],
self.getCurrentTime()])
        if probMatrix[0] > self.thresh_of_class:
            return True
        else:
            return False

    def get_classification_probablity(self, inputNr, classNr):
        for i in range(len(self.input[inputNr])):
            if self.add_to_matrix( inputNr, classNr):
                return True

        return False


    def calculate_new_attributes(self,stim):
        self.num_stims_act += 1
        self.time_start_cl = stim.date + 1.0
        self.cl = stim.identifier - 33024
        self.time_stop_cl = self.time_start_cl + 7.0
        self.gen_label = True


    def if_OVStimset(self,act_chunk):
        if type(act_chunk) == OVStimulationSet:
            for i in range(len(act_chunk)):
                stimulation = act_chunk.pop()
                if self.is_debugged:
                    print 'Received stim on input 7', stimulation.identi-
fier - 33025, ' at', stimulation.date, 's'
```

```
                if (stimulation.identifier > 33024 and stimulation.identi-
fier < 33030):
                    self.calculate_new_attributes(stimulation)
                    if self.is_debugged:
                        print 'Total classified: ' + str(self.num_T_class)



    def probablity_alter(self,all_probablities):
        max_probablity = max(all_probablities)
        max_position = all_probablities.index(max_probablity)
        del all_probablities[max_position]
        maxProb2 = max(all_probablities)
        maxPosDif = max_probablity - maxProb2
        if self.is_debugged:
            print "maxPosDif: " + str(maxPosDif) + "max_position: " +
str(max_position)
        return max_probablity, max_position, maxPosDif

    def if_predict_condition(self,maxPosDif):
        if (self.cl is not -1) and  maxPosDif >= self.maximum_probil-
ity_diff_in_Thresh and (self.time_start_cl <= self.time_predicted) and
(self.time_predicted <= self.time_stop_cl):
            return True
        return False




    def Hit_class_process(self,Hit_class, all_probablities):
        if True in Hit_class:  # Clear first threshold
            for i in range(4):
                for j in self.class_probablities[i]:
                    if j[0] > self.thresh_of_class:
                        all_probablities[i] += j[0]
            if self.is_debugged:
                print "all_probablities: " + str(all_probablities)

            max_probablity , max_position , maxPosDif = self.proba-
blity_alter(all_probablities)

            self.time_predicted = self.getCurrentTime()

            if self.if_predict_condition(maxPosDif):
                self.num_T_class += 1
                self.labels_prob_pred.append(max_position + 1)
                self.labels_prob_act.append(self.cl)


                temp_mean = self.time_predicted - self.time_start_cl
                if self.gen_label:
                    self.mean_dtns += temp_mean
```

```
                    self.gen_label = False
                    self.num_stims_class += 1
                    if self.is_debugged:
                        print "number  of stims is " +
str(self.num_stims_class)
                        print "predicted class was " + str(max_position +
1) + " label was: " + str(self.cl)
      # Process loop for classification

 # Mark as unclassified if not classified in allotted time
    def if_gen_lab(a,all_probablities):


        if  a.gen_label and (a.getCurrentTime() > a.time_stop_cl) :
            a.neg_class += 1
            a.gen_label = False

            if a.is_debugged:
                print "couldnt classify!"

            if (a.cl is not -1) and a.debug_pred_nothing :
                for i in range(4):
                    for j in a.class_probablities[i]:
                        all_probablities[i] += j[0]

                max_pos = all_probablities.index(max(all_probablities))

                a.labels_prob_act.append(self.cl)
                a.labels_prob_pred.append(max_pos + 1)
                a.num_stims_class += 1
                a.mean_dtns += 7
                a.num_T_class += 1
        return all_probablities

    def process(self):
        for i in range(len(self.input[7])):  #act process
            a = self.input[7].pop()
            self.if_OVStimset(a)

        all_probablities = [0.0, 0.0, 0.0, 0.0]
        all_probablities = self.if_gen_lab(all_probablities)

        Hit_class = [self.get_classification_probablity(inputNr=3,
classNr=0),
                     self.get_classification_probablity(inputNr=4,
classNr=1),
                     self.get_classification_probablity(inputNr=5,
classNr=2),
                     self.get_classification_probablity(inputNr=6,
classNr=3)]
        self.Hit_class_process(Hit_class,all_probablities)
```

```
        return



    def gather_data_to_write(self):
        file_name = self.setting['Current Subject Nr']
        path_file = self.setting['Results Directory']
        num_subjs = int(self.setting['Nr of subjects'])


        list_settings = {'ch': self.setting['channels'],
                         'epDur': self.setting['Epoch Duration'],
                         'epInt': self.setting['Epoch Interval'],
                         'freqTol': self.setting['Freq Tol'],
                         'simFreq': self.setting['SimulationFreq'],
                         'num_subjsects': num_subjs,
                         'currentSubjNr': file_name,
                         'classTresh': self.thresh_of_class,
                         'maximum_probility_diff_in_Thresh': self.maxi-
mum_probility_diff_in_Thresh}
        return file_name, path_file, num_subjs, list_settings

    def write_into_file(self,file_name, path_file, num_subjs, list_set-
tings):
        settingsFileName = ''
        for key, value in list_settings.items():
            if (key != 'simFreq'):
                settingsFileName += str(value)[2:]

        writeFile = path_file + 'prev' + settingsFileName + 'subject' +
str(file_name)
        data = {'settings': list_settings,
                'actual': self.labels_prob_act,
                'predicted': self.labels_prob_pred,
                'detectTime': self.mean_dtns,
                'stims': {'stimsNrClassified': self.num_stims_class,
                          'stimsNrActual': self.num_stims_act,
                          'num_T_class': self.num_T_class,
                          'totalTime': self.num_stims_act * 7,
                          'neg_class': self.neg_class}
                }

        pickle.dump(data, open(writeFile, 'wb'))
        print 'File saved to ' + writeFile

    # Write results to a file
    def uninitialize(self):
        if self.num_stims_class != 0:
            self.mean_dtns /= self.num_stims_class
            print "mean time for first classification: " +
str(self.mean_dtns)
```

```
        cmSklearn = confusion_matrix(self.labels_prob_act, self.la-
bels_prob_pred)
        cmSklearn = cmSklearn.astype('float') / cmSklearn.sum(axis=1)[:,
np.newaxis]
        print "Confusion matrix ------------------------------"
        print cmSklearn
        returned = confusion_matrix(self.labels_prob_act, self.la-
bels_prob_pred).ravel()
        F1 = returned[0:4]
        F2 = returned[4:8]
        F3 = returned[8:12]
        F4 = returned[12:16]
        ACC = [F1,F2,F3,F4]
        i = [20,15,12,10]
        j = 0
        for v in ACC:
            print("Frequency:",i[j])
            j +=1
            tn, fp, fn,tp = v
            print("True Negatives: ",tn)
            print("False Positives: ",fp)
            print("False Negatives: ",fn)
            print("True Positives: ",tp)
            Accuracy = (tn+tp)*100/(tp+tn+fp+fn)
            print(("Accuracy {:0.2f}%:").format(Accuracy))
            Precision = tp/(tp+fp)
            print(("Precision {:0.2f}").format(Precision))
            Recall = tp/(tp+fn)
            print(("Recall {:0.2f}").format(Recall))
            f1 = (2*Precision*Recall)/(Precision + Recall)
            print("F1 Score {:0.2f}".format(f1))
            Specificity = tn/(tn+fp)
            print("Specificity {:0.2f}".format(Specificity))
        file_name, path_file, num_subjs, list_settings=
self.gather_data_to_write()
        self.write_into_file(file_name, path_file, num_subjs, list_set-
tings)


box = mod_OVBox()
```

start-stimulator.lua

```
delay = 1
function initialize(box)
      s_delay = box:get_setting(2);

      io.write(string.format("Delay : [%s]\n", s_delay))

      if (s_delay:find("^%d+$") ~= nil) then
            delay = tonumber(s_delay)
      else
            io.write("[ERROR] The parameter should be a numeric value\n")
```

```
	end

end

function uninitialize(box)
end

function process(box)
	box:send_stimulation(1, 0x00008001, delay, 0)
end
```

training-acquisition-controller.lua

```
-- training aquisition controller
Sq = {}
cycles_num = 0

stim_time = nil
break_time = nil
flashing_rate = nil

t_width = nil
t_height = nil

t_pos = {}
t_nums = {}

stimulationLabels = {
	0x00008100,
	0x00008101,
	0x00008102,
	0x00008103,
	0x00008104,
	0x00008105,
	0x00008106,
	0x00008107
}

function process_settings(box)

	for t in s_Sq:gmatch("%d+") do

		table.insert(Sq, t)
		cycles_num = cycles_num +1

	end

	if (s_break_time:find("^%d+[.]?%d*$") ~= nil) then
		break_time = tonumber(s_break_time)
		box:log("Info", string.format(">>>Break time or durtation :
[%s]", s_break_time))
	else
```

```
            box:log("Error", "problem with the parameter in the break du-
ration \n")
            error()
      end
      if (s_flashing_rate:find("^%d+[.]?%d*$") ~= nil) then
            flashing_rate = tonumber(s_flashing_rate)
            box:log("Info", string.format("Flashing rate : [%s]", s_flash-
ing_rate))
      else
            box:log("Error", "problem with the parameter in the Flashing
rate \n")
            error()
      end

      if (s_stim_time:find("^%d+[.]?%d*$") ~= nil) then
            stim_time = tonumber(s_stim_time)
            box:log("Info", string.format(">>>Stimulation time : [%g]",
stim_time))
      else
            box:log("Error", "problem with the parameter in the stimula-
tion duration \n")
            error()
      end
      if s_width ~= nil and s_height ~= nil then
            box:log("Info", string.format("Target shape : width = %g,
height = %g", t_width, t_height))
      else
            box:log("Error", "problem with the parameter in the Traget
Shape \n")
            error()
      end

      t_nums = 0

      for s_target_x, s_target_y in s_targetPositions:gmatch("(-
?%d+[.]?%d*);(-?%d+[.]?%d*)") do
            box:log("Info", string.format("Target %d : x = %g y = %g",
t_nums, tonumber(s_target_x), tonumber(s_target_y)))
            table.insert(t_pos, {tonumber(s_target_x), tonumber(s_tar-
get_y)})
            t_nums = t_nums + 1

      end

end
function load_settings(box)

      -- load the goal Sq
      s_Sq = box:get_setting(2)
      -- get the duration of a stimulation Sq
      s_stim_time = box:get_setting(3)
      -- get the duration of a break between stimulations
      s_break_time = box:get_setting(4)
```

```
	-- get the delay between the appearance of the marker and the start
of flickering
	s_flashing_rate = box:get_setting(5)
	-- get the target size
	s_targetSize = box:get_setting(6)
	-- get the targets' positions
	s_targetPositions = box:get_setting(7)



	s_width, s_height = s_tar-
getSize:match("^(%d+[.]?%d*);(%d+[.]?%d*)$")
	t_width = tonumber(s_width)
	t_height = tonumber(s_height)




end

-- create the configuration file for the stimulation-based-epoching
--this file is used during classifier training only
function config_maker_sbp(box)

	myfile_name = box:get_config("${Player_ScenarioDirectory}/configura-
tion/stimulation-based-epoching.cfg")
	myFile = io.open(myfile_name, "w")
	box:log("Info", "Writing into [" .. myfile_name .. "]")
	if myFile == nil then
		box:log("Error", "Cannot write to [" .. myfile_name .. "]")
		return false
	end

	myFile:write("<OpenViBE-SettingsOverride>\n")
	myFile:write("   <SettingValue>", stim_time, "</SettingValue>\n")
	myFile:write("   <SettingValue>", flashing_rate, "</Set-
tingValue>\n")
	myFile:write("   <SettingValue>OVTK_StimulationId_Target</Set-
tingValue>\n")
	myFile:write("</OpenViBE-SettingsOverride>\n")
	myFile:close()


end
-- create the configuration file for the training program
function config_maker_tp(box)

	file_name = "${CustomConfigurationPrefix${OperatingSystem}}-ssvep-
demo-training${CustomConfigurationSuffix${OperatingSystem}}"

	myFile = io.open(box:get_config(file_name),"w")
	box:log("Info","Writing into [..file_name..]")
```

```
	if myFile == nil then
		box:log("Error", "Cannot write to [".. file_name..."]")
		return false
	end

	myFile:write("SSVEP_TargetCount = ", t_nums, "\n")
	myFile:write("SSVEP_TargetWidth = ", t_width, "\n")
	myFile:write("SSVEP_TargetHeight = ", t_height, "\n")

	for target_index, position in ipairs(t_pos) do

		myFile:write("SSVEP_Target_X_", target_index - 1, " = ", posi-
tion[1], "\n")
		myFile:write("SSVEP_Target_Y_", target_index - 1, " = ", posi-
tion[2], "\n")
	end

	myFile:close()

end
function initialize(box)


	dofile(box:get_config("${Path_Data}") .. "/plugins/stimulation/lua-
stimulator-stim-codes.lua")

	load_settings(box)

	process_settings(box)

	if(config_maker_sbp(box) == false) then  return false end
	if (config_maker_tp(box) == false) then return false end



end

function uninitialize(box)
end

function process(box)

	while box:keep_processing() and box:get_stimulation_count(1) == 0 do
		box:sleep()
	end

	current_time = box:get_current_time() + 1

	box:send_stimulation(1, OVTK_StimulationId_ExperimentStart, cur-
rent_time, 0)

	current_time = current_time + 2

	for i,j in ipairs(Sq) do
```

```
            box:log("Info", string.format("Goal no %d is %d at %d", i, j,
current_time))
            -- mark goal
            box:send_stimulation(2, OVTK_StimulationId_LabelStart + j,
current_time, 0)
            -- wait for flashing_rate seconds
            current_time = current_time + flashing_rate
            -- start flickering
            box:send_stimulation(1, OVTK_StimulationId_VisualStimulation-
Start, current_time, 0)
            -- wait for stim_time seconds
            current_time = current_time + stim_time
            -- unmark goal and stop flickering
            box:send_stimulation(1, OVTK_StimulationId_VisualStimula-
tionStop, current_time, 0)
            -- wait for break_time seconds
            current_time = current_time + break_time
      end

      box:send_stimulation(1, OVTK_StimulationId_ExperimentStop, cur-
rent_time, 0)

      box:sleep()
end
```

voter.py

```
import collections
# CONSTANTS
Max_len = 1
probsAll = [0.0, 0.0, 0.0, 0.0]
number_of_stims = 4

class MyOVBox(OVBox):
    def __init__(self):
        OVBox.__init__(self)


        self.prob_classes = [collections.deque(maxlen=Max_len),
                             collections.deque(maxlen=Max_len),
                             collections.deque(maxlen=Max_len),
                             collections.deque(maxlen=Max_len)]


    def get_thresholds(self,that):
        Thresholds = self.setting['Thresholds'].split(':')
      if Thresholds:
            self.maxProbDiffThresh = float(Thresholds[1])
            self.classThresh = float(Thresholds[0])
      else:
          self.classThresh = 0.5
          self.maxProbDiffThresh = 0.25
```

```
    def initialize(self):
        self.get_thresholds(self)



    def get_prob(self, num_input, num_class):
      size  = len(self.input[num_input])
        for i in range(size):
            pm = self.input[num_input].pop()
          current_time = self.getCurrentTime()
          current_pm = pm[0]
            self.prob_classes[num_class].append([current_pm, cur-
rent_time])
            if pm[0] > self.classThresh:
                return True
        return False

    def clear_thresh(self, hits, sent_probs):
        if True in hits:
            for stim in range(number_of_stims):
           for p in self.prob_classes[stim]:
                    state = p[0];
                if state > self.classThresh:
                        sent_probs[stim] += state

          mp = max(sent_probs)
          mps = sent_probs.index(mp)
          del sent_probs[mps]
          mp2 = max(sent_probs)
          diff = mp - mp2
          if(diff >= self.maxProbDiffThresh):
                self.tcp_writer(mps)

      return

    def tcp_writer(self,mps):
        stimul_set = OVStimulationSet(self.getCurrentTime(), self.getCur-
rentTime() + 1. / self.getClock())
        stimul_set.append(OVStimulation(33025 + mps, self.getCur-
rentTime(), 0.))
        self.output[0].append(stimul_set)
        return
    def process(self):
        probsAll = [0.0, 0.0, 0.0, 0.0]
        hit_inClass = [self.get_prob(num_input=3, num_class=0),
                    self.get_prob(num_input=4, num_class=1),
                    self.get_prob(num_input=5, num_class=2),
                    self.get_prob(num_input=6, num_class=3)]

        self.clear_thresh(hit_inClass, probsAll)

      return
```

```
    def uninitialize(self):
        return

box = MyOVBox()
```

Java Side

BasicView.java

```
import javax.swing.JComponent;
import java.awt.*;

public class BasicView extends JComponent{

    FlasherMain flasher;
    public BasicView(){




    }

    public void  setup(FlasherMain flasherMain){
        flasher = flasherMain;

    }

    @Override
    public void paintComponent(Graphics g0){


        Graphics2D g = (Graphics2D) g0;

        g.setColor( Color.black);

        g.fillRect(0,0,getWidth(), getHeight());

        if (flasher != null) {
            for (CustomRec b : flasher.Buttons) {
                if (b != null)
                    b.draw(g);

            }
        }
        repaint();
    }



}
```

Constants.java

```
public class CONSTANTS {

    enum RecID{
```

```
        one ,
        two ,
        three ,
        four

    }

   static Vec2D size  =  new Vec2D(500,300);
    static int STARTING_HZ = 40;
}
```

CustomRec.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;

public class CustomRec implements ActionListener {

    CONSTANTS.RecID id;
    Vec2D pos;
    Color color;
    int width = CONSTANTS.size.x / 2, height = CONSTANTS.size.y/2;
    private boolean isFlashing=  true;
    private final Timer timer;
    private  int HZ = 60;
    private long next_sec = System.currentTimeMillis() + 1000;
    private int LS_frame = 0 , CS_frame = 0;
    public CustomRec(CONSTANTS.RecID id, int HZ){
        this.id = id;
        this.HZ = HZ;
        pos = new Vec2D(0,0);
        setupRec();
        this.timer =  new Timer(1000/HZ,  this);
        timer.start();
    }

    private void setupRec(){

        switch (id) {
            case one:
                pos.set(0,0);
                color = Color.RED;
                break;
            case two:
                pos.set(width +10, 0);
                color = Color.RED;
                break;
            case three:
                pos.set(0, height+10);
                color = Color.RED;
                break;
```

```
            case four:
                pos.set(width+ 10, height +10);
                color = Color.RED;
                break;
        }

    }

    public void draw(Graphics2D g){


        long current_T = System.currentTimeMillis();

        if(current_T > next_sec){

            next_sec += 1000;
            LS_frame = CS_frame;
            CS_frame = 0;
        }

        CS_frame++;

        g.setColor(color);
        if (isFlashing){
            g.fillRect(pos.x, pos.y, width, height);
            if(LS_frame != 0) {

                write_in_csv((current_T / 1000) % 60, LS_frame);
            }
            }

        else clear(g);

    }

    private void write_in_csv(long T, int fps){

        try{

            FileWriter CSV_file = new FileWriter("Ana-
lyser/Java_FrameFSP_Analyser.csv", true);

            CSV_file.append(T + "," + fps);
            CSV_file.append("\n");

            CSV_file.flush();
            CSV_file.close();


        } catch (IOException e) {

            System.out.println(e.fillInStackTrace());
        }
```

```
    }
    public void clear(Graphics2D g){
        g.clearRect(pos.x,pos.y,width,height);

    }

    @Override
    public void actionPerformed(ActionEvent e) {
        isFlashing = !isFlashing;
    }
}
```

FlasherMain.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.ActionListener;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

public class FlasherMain extends JFrame {



    List<CustomRec> Buttons;
    public Component comp;
    int StartingHZ = 0;
    int num = 0;
    public FlasherMain(Component comp, int num, int StartingHZ) {
        Buttons = new ArrayList<>();
        this.comp = comp;
        this.num = num;
        this.StartingHZ = StartingHZ;
            setup();

    }

    private void setup() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        pack();
        setVisible(true);
        getContentPane().add(BorderLayout.CENTER, comp);
        setBackground(Color.BLACK);
        setSize(CONSTANTS.size.x+20, CONSTANTS.size.y+50);
        setLocationRelativeTo(null);
        if(num == 0)
         num = Check();

        if(num != 0){

            add_squares(num);
```

```
        }
        //Buttons.add(new CustomRec(CONSTANTS.RecID.one, 120));
        repaint();

    }

    private void add_squares(int n){

        int st = CONSTANTS.STARTING_HZ;

        if(StartingHZ != 0) st = StartingHZ;
        for (int i = 0; i < n; i++){

            System.out.println(i);
            Buttons.add(new CustomRec(CONSTANTS.RecID.values()[i],st+ (5 *
i) ));

        }


    }

    private int Check(){

        try{

            FileReader fileReader = new FileReader("instructions.txt");
           char [] a = new char [4];


            fileReader.read(a);


            return Character.getNumericValue(a[0]);


    } catch (FileNotFoundException e) {
            return 0;
        } catch (IOException e1) {
            return 0;
        }

    }

        public static void main(String[] args){
        int Num = 0;
        int StartingHZ= 0;
        BasicView basicView = new BasicView();
        if (args.length > 0){

            Num = Integer.valueOf(args[0]);
            StartingHZ = Integer.valueOf(args[1]);
```

```
            FlasherMain flasherMain = new FlasherMain(basicView, Num,
StartingHZ);
            basicView.setup(flasherMain);

        }
        else {
            FlasherMain flasherMain = new FlasherMain(basicView, 0, 0);
            basicView.setup(flasherMain);
        }



    }


}
```

Vec2D.java

```
public class Vec2D {

    public int x,y;



    public Vec2D(int x, int y ){
        this.x = x;
        this.y = y;


    }


    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public void setX(int x) {
        this.x = x;
    }

    public void setY(int y) {
        this.y = y;
    }
    public void set(int x, int y){

        this.x  = x;
        this.y = y;
    }
}
```