

# UI Components

Class Detection Visualizer: This text visualizer shows the detected class (for example, r as run) by the classifier (Openvibe) at runtime, Used class (for example, ? represents unclear class) for participating in voting, and all the used classes for participating in voting (for example, last three detected classes used for voting, r??).

**Speed Visualizer:** This text visualizer shows the current speed level, actual speed in Unity scale, and the Max speed level of the current condition. Speed levels and Max Speed are defined in the AppSet.txt file and also can be controlled or redefined in the trial file. Currently, speed levels are defined from 1 to 9 using the parameter “star\_speed\_au\_level = 9” in the AppSet.txt file. AppSet.txt file is under the “ConfigFiles” folder of the root directory of the application.

**Speed Tester:** These buttons are used to check/monitor the speed level so that experimenters can decide what could be the Max speed and how many levels should be there. Initially the system takes the max speed level and the number of speed levels (say, 1..9) from the configuration file AppSet.txt. The experimenter has the option to see all the speed levels and the actual visual speed of the stars movement by manually clicking “Slower” or “Faster” buttons. Or, clicking the “Demo Speed” button to monitor iteratively all the speed levels starting from 1 to Max speed level.

**Online Speed:** After calibration and before running the experiment, experimenters can see how good the classification is and how well the star movement follows the brain signal. To activate the online demo, experimenters need to click the button “Online Speed Test” and click the button “Online Speed Test Done” to finish the online demo. If the experimenters and the participant finds the system is not following well or the calibration is not good enough then they can redo the calibration task, and repeat the online demo test. If they find it is good enough to run the experiment then click the “Experiment 2” button to proceed with the experiment trials.

**Calibration:** Click the “Calibration” button to run the calibration module which presents visual stimulus with instructions and provides necessary markers/triggers to the openvibe data recorder scenario.

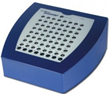
**Experiment:** After successful calibration and online demo test, experimenters can start the actual experiment trials by clicking the button “Experiment 2”. Do not confuse it with the name Experiment 2, the number 2 here is our index to identify which experiment it is from the list of all experiments.

**Break and Exit:** These buttons used to pause the experiment or exit the application. The button “Tea Break” is added here so that the participant can take a break / rest for a while.

**Recenter:** This button used to recenter the HMD at any time. If the participant finds HMD is not aligned in center then he/she can ask the experimenter to recenter the HMD. In this case participants were asked to straight their head position and orientation, and then click the button “Recenter”.

**N.B:** Note that all the UI buttons are always active. This is done so that experimenters have the option to control the system easily, in case of system crash or manual break. If we used to hide/deactivate some buttons/functionality, for example, after calibration if the calibration button deactivated and you found that you need to redo the calibration then you need to restart reinitializing the system, which is inconvenient.

# Installing Devices

**EEG Amp:** In our case, we used Brainvision’s QuickAmp amplifier to collect the EEG data. You can use other devices of your preferences. Connect the EEG amplifier where Openvibe is installed. Then configure the Opevibe Acquisition server with the current EEG amplifier.

**Openvibe and Experiment PC:**

#### Connection

1. They can be the same PC. In this case, you can set the parameter “eeg\_marker\_ip = localhost”of the AppSet.txt file.
2. If they are separate PCs then they need to be in the same network. The easiest way is to connect them using LAN cable and manually configure the IP address. Then you can set the parameter “eeg\_marker\_ip = 192.168.0.2”of the AppSet.txt file. The ip address here is our given address, you can use your own. The connection is required to transfer data and signals between the Openvibe and the application. The IP address is required to transfer marker/trigger signal from the application to the Openvibe acquisition server using TCP/IP protocol. The data from the openvibe scenarios are collected in the application using Lab Streaming Layer (LSL) protocol.

# Run Application

**Procedural steps:**

1. After getting ready with the EEG electrodes, run NA\_Record\_training\_data scenario.
2. Click Calibration to start calibration and follow the instructions shown on the HMD.
3. After completing the calibration task, in openvibe run the NA\_0\_TrainCSPFilters.xml scenario to adjust the filtering parameters.
4. Then in openvibe run NA\_1\_TrainCLassifier.xml to train the classifier.
5. Now you can see the online speed test, whether star speed is following the brain signal or not. To see that click Online Speed Test and click on Online Speed Test Done when you are done.
6. Now we are ready to run the experiment. It is better to restart the application so that it can reinitialize the IP and connection functionality.
   1. In openvibe, run NA\_2\_TestOnData.xml. This scenario used to classify data online and send the result to the application.
   2. Then click the button Experiment 2 and follow the trial instruction shown on the HMD

At any time to recenter the view, click the button Recenter

**N.B:** Very important, please rename the Traning.gdf and Online.gdf and copy those to another directory before running another session with NA\_Record\_training\_data and NA\_2\_TestOnData. This is so that you do not need to rename the data file manually in the scenario. Or in other words, all the scenarios used either Training.gdf or Online.gdf as default file names. Each time you run the NA\_Record\_training\_data.xml or NA\_2\_TestOnData.xml overwrites the files.

# Application/Code design

## Calibration

The participant is asked to imagine two different tasks, *Run* or *Relax* (Currently, named as Walk) immediately after seeing the visual stimuli while speed of the star movement remains static with default minimum velocity. The participants were instructed to imagine the task until they see the stop visual stimuli. The tasks are randomized. Each task has a fixed duration which can be setted using the parameter “calibration\_task\_duration = 8” of the AppSet.txt. The number represents the duration in seconds. Value of the parameter “calibration\_task\_interval = 4” defines the interval between two trials. The parameter “calibration\_task\_repetition = 20” defines how many repetitions each task should present. There is an option to change the images of the visual stimuli. In this case, you can change the images of the directory named “Images” under the ConfigFiles folder. Keep the name of the images as it is, since the source code by default uses those names.

**Functions**

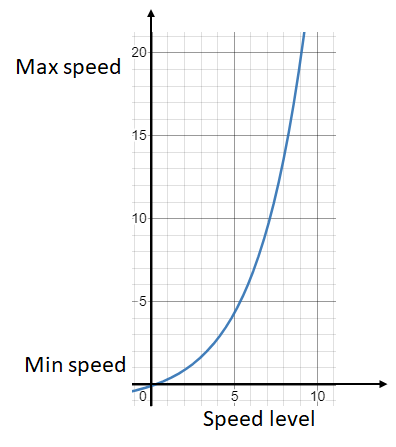
During the start of the application, the calibration module generates a randomized task list (Run/Walk/Relax) with a given number of repetition of each condition.

When the button “Calibration” is clicked it starts the calibration task (function named “Calibrate” in the Calibrate.cs). The detailed functional steps are:

1. It initiates a tcp/ip link with the Openvibe to send markers/triggers.
2. Displays task instruction. You have an option to change the text, use the NeuroCalibration\_Instruction.txt file (under the ConfigFiles directory) for this purpose.
3. Starts presenting visual stimuli according to the sequence of the task list for a given duration and sends corresponding markers/triggers to the openvibe.
4. At the end of the tasks, sends a marker/trigger to the openvibe to inform the task is done, and now can be run the classification training scenario.

## Online Demo

After training the classifier you have an option to test the classifier behavior and the corresponding star speed whether it is following the imaginary movement of the participant or not. By default the maximum (star\_speed\_max = 20) and minimum (star\_speed\_min = 0.1) speed and the speed level (star\_speed\_au\_level = 9) are taken from the AppSet.txt. During the application initialization, the speed values for the corresponding speed level are calculated using an exponential logarithmic function considering these minimum speed, maximum speed, and the number of speed levels. For example, say there are 1 to 9 speed levels, and taking the maximum and minimum speed the speed values for each level can be shown as the following figure. To smoothing up the star speed and filtering online detection of imaginary run/walk/relax we added a voting algorithm where the winner class used to update the star speed. Online demo functionality is implemented in the NeuroAdaptiveVR.cs.



**Exponential Speed**

This is used to define the unity speed values for each of the corresponding speed levels. Everytime classifier detects a run, the system speeds up the star movement logarithmically so that the user has a feeling of truely speeding it up, and vice versa for the relax/walk detection. This functionality is implemented in the function “UpdateExponentialSpeed()” in NeuroAdaptiveVR.cs.

To calculate the exponential speed level we used the following formula:

1. Calculate 10 base log values for each level, or in a simple way:
   1. For i=0 to star\_speed\_au\_level
      1. log10base[i] = Mathf.Pow(10f, (1 + i / (star\_speed\_au\_level - 1f)));
2. Scale down the log10base to 0 to 1.
   1. For i=0 to star\_speed\_au\_level
      1. scale\_0to1[i] = (log10base[i] - 10f) / 90f
3. Scale up to the maximum and minimum speed
   1. For i=0 to star\_speed\_au\_level
      1. exp2\_au\_speeds[i] = minSpeed + (scale\_0to1 \* (maxSpeed - minSpeed));

**Voting Algorithm**

This is implemented in a function named Exp2\_WinnerClass() in NeuroAdaptiveVR.cs. When the experimenter clicks the button “Online Speed Test” it enables the system to collect classifier data from the openvibe, and encoded it using three different characters “r”, “w”, and “?”. Where, “r” =run, “w”= walk/relax, and “?”= unclear. This is implemented in the function OnlineSpeedTest\_PrcoessOpenvibeData(). The system uses a stack to store the detected class characters. The size of the stack is determined from the parameter “number\_of\_classifiers\_to\_use\_in\_vote = 3” in AppSet.txt. The voting algorithm counts each category of the detected class and finds the maximum to declare the winner class. This winner class is used to update the star speed.

**Update star speed**

This operation is implemented in the function Exp2\_Set\_au\_speed\_level(winner\_class), which takes the winner class symbol, and updates the star speed. As mentioned above, the exponential function is used to generate the speed values and stored in an speed array. The index of the array is similar to speed level. We used simple logic to implement this function:

1. Each time it takes a run “r” symbol it increments the speed level index, and the corresponding speed value of the speed array is applied to update the star speed.
2. Each time it takes a walk/relax “w” symbol it decrements the speed level index and updates the star speed.
3. Each time it takes an unclear “?” symbol it is forced to set to the default index, either incrementing or decrementing the speed level index by 1 and updates the star speed accordingly.

## Experiment

Experiment’s logic is implemented in NeuroAdaptiveVR.cs. The trial conditions are defined in the trial file (for example, 1\_Trials\_Exp2.txt). The trials files are located at application\_directory/ConfigFiles/Trials/.

When the button “Experiment 2” is clicked it starts executing trials. The detailed functional steps are:

1. It initiates a tcp/ip link with the Openvibe to send markers/triggers.
2. It inilitizes required parameters and star speed.
3. Loads trials from the given trial file. The trial file is determined by the ID of the participant.
4. Execute trials and log data.

There is an option to define the max star speed in each trial, this is due to vari the star speed depending on different experimental conditions. Therefore, the exponential speed algorithm is executed once for each trial.

The voting algorithm and update star speed works similarly. However, the experiment also has the option to follow the star speed according to the earlier recorded similar trial. For a recorded condition, a class detected by the openvibe should be ignored and need to use an earlier recorded class, therefore, we implemented another function named exp2\_ProcessAndCollectOpenvibeData() to handle this situation.

**Test Experiment (offline)**

To test Online Demo or the Experiment 2 with the existing data you can run the NA\_Simulator.xml scenario on the Openvibe and then click the “Online Speed Test” button or “Experiment 2” button. The EEG data was recorded for a few minutes (less than 5minutes).

# Target Integration

# AppSet.txt

It contains all the necessary parameters to set the application behavior. Here we describe briefly those parameters which are used but not discussed earlier.

**default\_au\_speed\_level\_index = 0**

Set this parameter to -1 if you want the default speed level to be in the middle. For example, in a scale of 1 to 9 if you want the default starting speed level for each trial should be 5 then put -1 to this parameter. Otherwise, put 0 to start from the first index, or any other number between 0 to 8 of your preferences. The entire system has a tendency to round the speed level toward this default level when “?” or unclear states are detected.

**class\_run\_threshold = 0.3**

Set this parameter to define the threshold of the class run.

**class\_walk\_threshold = 0.7**

Set this parameter to define the threshold of the class walk/relax.

**calibration\_clipart\_duration = 0.5**

Determines the display duration of the walk/run/stop icon.

**calibration\_change\_speed\_on\_task = false**

If true then changes speed during calibration task, faster while running and slower while walking/relaxing.

**sliderSpeed = 0.01**

Used to determine the slider speed (For answering likert items).

**sliderFastMoveActivatingThresholdTime = 0.5**

While keeping pressing this duration determines when to start updating the slider value faster.

**sliderFastMoveSensitivityInSeconds = 0.033**

Used to determine the sensitivity of the slider (For answering likert items), how many numbers per second.

**eeg\_marker\_port = 15361**

Port number to communicate with openvibe.