

# System Design Document

## [Old ver.] World Tour System(WTS)

Developer: Sanghum Woo (sanghumwoo@gmail.com)

System version: 1.0

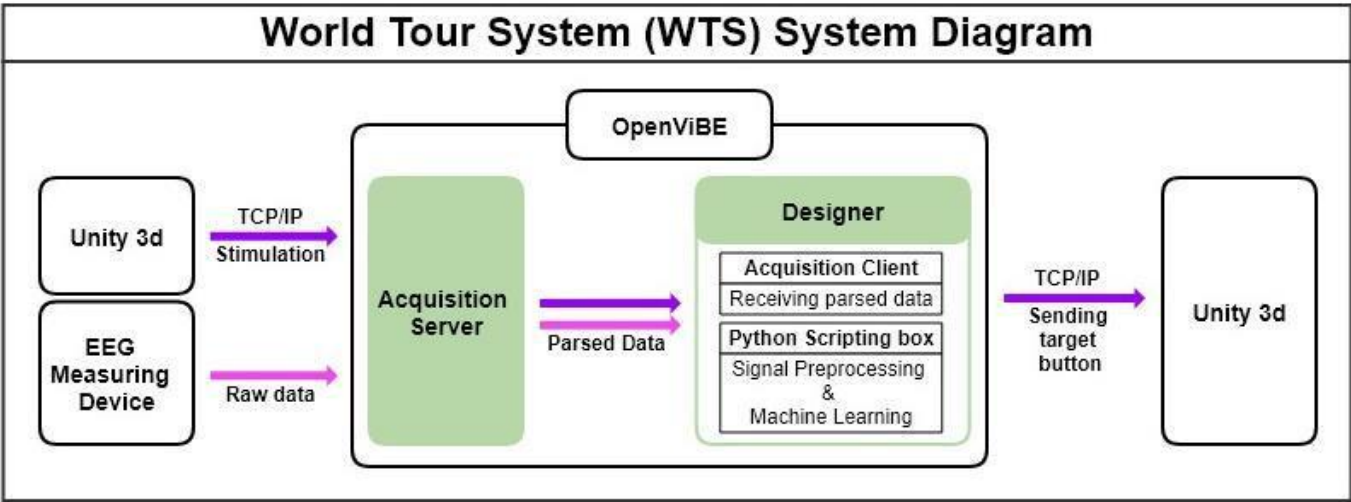
Latest reviewed: 11 March 2020



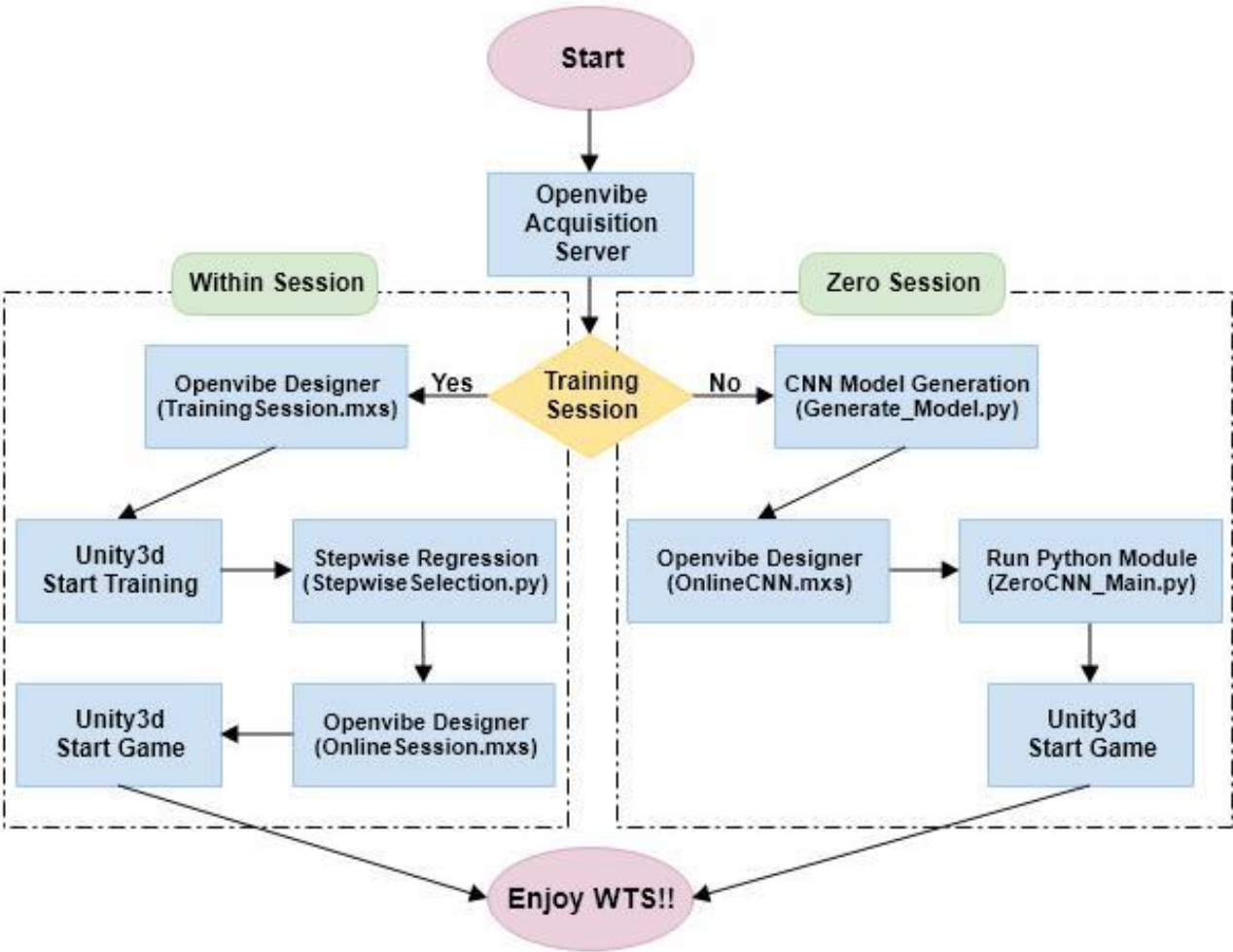
# Contents

<b>1. System Diagram and Flowchart -----</b>	<b>3</b>
<b>2. Within Session used SWLDA -----</b>	<b>4</b>
A. Training Session	
B. Online Session	
<b>3. Zero training Session used CNN -----</b>	<b>9</b>
A. CNN Model Generation Process	
B. Online Session	
<b>4. Trouble Shooting -----</b>	<b>14</b>
A. Python 2.7 Library import	
B. joblib.load	
C. pip operation	
D. Required Library	
<b>5. How to Start -----</b>	<b>16</b>

System Diagram



System Flowchart



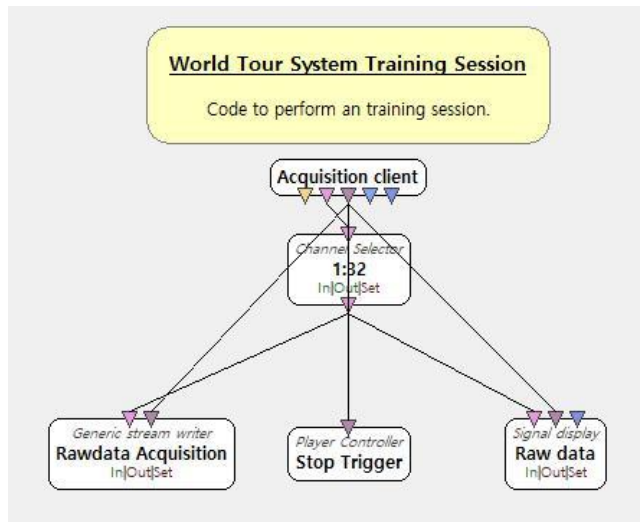
## Within Session used SWLDA (File path: /Within)

The system extracts significant features from user data through training sessions and then trains them to generate LDA classifiers and use it to predict target buttons.

### ① Training Session

#### OpenViBE designer

Designer code: **TrainingSession.mxs** (File path: /Training)



**Description:** EEG data and stimulation coming from Acquisition client are written into Generic stream writer.

**Other information:** At the end of the training session, the trigger fires to terminate the designer. The stored data is converted into Mat files in Python code, processed by signal processing, and stepwise feature selection is performed.

**Rawdata Acquisition:** Because the file name is saved with a timestamp followed by the file name, the file does not overlap when saved with each scene change.

#### Python module

Python code: **StepwiseSelection.py** (File path: /SourceCode)

**Code Description:** Convert ov file saved through Openvibe Designer to mat file and then convert it to feature vector by signal processing. After that, the significant features are extracted through the stepwise selection process, and the LDA is constructed and saved as a file.

#### Progression order

##### 1. Load ov file and convert it to mat file. And load the mat file and store it in variables.

```
##Generate Preprocessing Training data
ctime = datetime.today().strftime("%m%d_%H%M")
SelectedF_path = 'C:/Users/wldk5/WorldSystem/Within/StepWise/Features/' + ctime + 'SelectedFeatures.pickle'
Classifier_path = 'C:/Users/wldk5/WorldSystem/Within/StepWise/Classifiers/' + ctime + 'Classifier.pickle'

channelNum = 32
downsampleRate = 4

ov_Path = "C:/Users/wldk5/WorldSystem/Within/Training/Data/"
current_list = []
current_list = sorted(glob.glob(ov_Path + '*.ov'), key=os.path.getmtime, reverse=True)
ovfile_name = current_list[0]
matfile_name = current_list[0][:-3] + ".mat"

print("current ov file path:", current_list[0])
eng = matlab.engine.start_matlab()
k = eng.convert_ov2mat(ovfile_name, matfile_name)
mat = hdf5storage.loadmat(matfile_name)
channelNames = mat['channelNames']
eegData = mat['eegData']
samplingFreq = mat['samplingFreq']
samplingFreq = samplingFreq[0,0]
stims = mat['stims']
channelNum = channelNames.shape
channelNum = channelNum[1]
eegData = np.transpose(eegData)
```

## ■ Variables

**ctime:** System automation is achieved by saving the file name of Classifier and Selected features with the current time.

**current\_list:** The list is saved in order of the most recently modified file in the folder where the ov file is stored. It is used to load the file stored in the 0th index.

## ■ Call Matlab function in Python

### ① Run this code on Matlab script

```
cd (fullfile(matlabroot,'extern','engines','python'))
```

```
system('python setup.py install --prefix "C:\ProgramData\Anaconda3\envs\world"')
```

\* It's my python path. Write down your python path here. I run Matlab with administrator account and run this code.

### ② Matlab engine start and function call

```
eng = matlab.engine.start_matlab()
```

```
k = eng.convert_ov2mat(ov file path', 'mat file path')
```

\* To use a specific Matlab function, the function file must exist in the working directory where the Python code exists!!! (ex. convert\_ov2mat.m is a function that must exist in the python code dir.)

## 2. After the EEG preprocessing process, it is converted into feature vectors.

```
##Preprocessing process
eegData = Downsampling(eegData, downsampleRate)
samplingFreq = samplingFreq/4
sampleNum = eegData.shape[1]

#Common Average Reference
eegData = Re_referencing(eegData, channelNum, sampleNum)

#Bandpass Filter
eegData = butter_bandpass_filter(eegData, 0.5, 10, samplingFreq,4)

#Epoching
epochSampleNum = int(np.floor(0.4 * samplingFreq))
offset = int(np.floor(0.2 * samplingFreq))
baseline = int(np.floor(0.6 * samplingFreq))
[EpochsT, NumT] = Epoching(eegData, stims, 1, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[EpochsN, NumN] = Epoching(eegData, stims, 0, samplingFreq, channelNum, epochSampleNum, offset, baseline)

#DownsamplingEpoch
[EpochsT, EpochsN, epochSampleNum] = DownsamplingEpoch(EpochsT, EpochsN, downsampleRate)

#Convert to feature vector
featureNum = channelNum*epochSampleNum
[FeaturesT, FeaturesN] = Convert_to_featureVector(EpochsT, NumT, EpochsN, NumN, featureNum)
TrainData = np.concatenate((FeaturesT, FeaturesN))
TrainLabel = np.concatenate((np.ones((NumT,1)).astype(int), np.zeros((NumN,1)).astype(int))).ravel()
```

## ■ Preprocessing process

Downsampling [2048 -> 512] → Re-referencing(CAR) → bandpass[0.5, 10] → Epoching[200 600] → BaselineCorrection[200 600] → DownsamplingEpoch[512 -> 128]

## 3. Through the stepwise selection process, we select the meaningful features and store the index information in the array, and then construct and save the LDA based on the train data reflecting this.

```
lda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
```

**shrinkage** is a tool to improve estimation of covariance matrices in situations where the number of training samples is small compared to the number of features. By setting shrinkage into 'auto', it automatically determines the optimal shrinkage parameter in an analytic way following the lemma introduced by Ledoit and Wolf. Note that currently shrinkage only works when setting the solver parameter to 'lsqr' or 'eigen'.



```
joblib.dump(lda, Classifier_path, protocol=2)
with open(SelectedF_path, 'wb') as f:
    pickle.dump(SelectedFeatures, f, protocol=2)
```

★★★ This system is implemented to run as Python 3.6 module except Python 2.7 module which runs in Openvibe.

Therefore, by setting **protocol** into 2, the saved files can be loaded in Openvibe.

**stepwise\_selection** is little bit tuned because of the depreciated function matters. In order to generalize the performance of SWLDA, I obtained a 96% accuracy by performing 10 cross validations on P300 speller training data of 55 subjects.

```
#Feature Selection process
x = np.arange(featureNum)
Data = pd.DataFrame(TrainData, columns=x)
SelectedFeatures = stepwise_selection(Data, TrainLabel)
print('resulting features:')
SelectedFeatures = np.sort(SelectedFeatures)
print(SelectedFeatures)
SWTrainData = TrainData[:,SelectedFeatures]

#Saving LDA classifier and Selected features
lda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
lda.fit(SWTrainData, TrainLabel)
joblib.dump(lda, Classifier_path, protocol=2)
with open(SelectedF_path, 'wb') as f:
    pickle.dump(SelectedFeatures, f, protocol=2)
```

```
def stepwise_selection(X, y,
                      initial_list=[],
                      threshold_in=0.01,
                      threshold_out = 0.05,
                      verbose=True):
    """ Perform a forward-backward feature selection
    based on p-value from statsmodels.api.OLS
    Arguments:
```

\* The original author of *stepwise\_selection* is:

<https://datascience.stackexchange.com/users/24162/david-dale>

#### 4. The saved files are as follows.

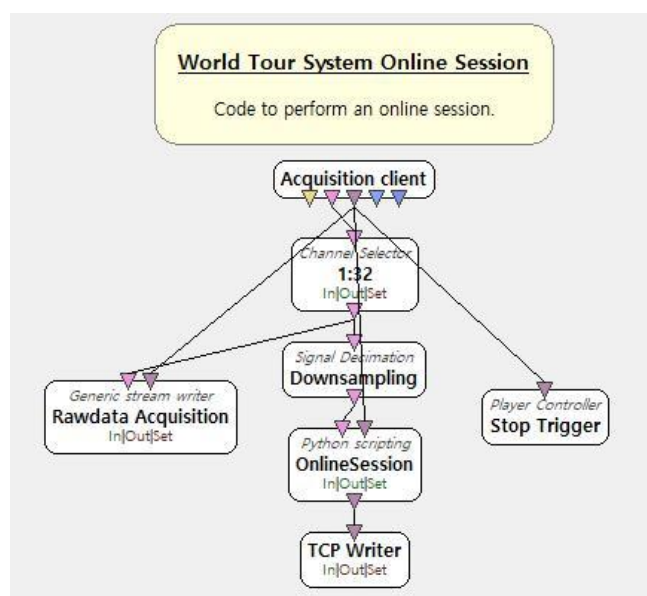
지 > WorldSystem > Within > StepWise > Classifiers	
이름	수정한 날짜
1021_1927Classifier.pickle	2019-10-21

지 > WorldSystem > Within > StepWise > Features	
이름	수정한 날짜
1021_1927SelectedFeatures.pickle	2019-10-21

## ② Online Session

### OpenViBE designer

Designer code: **OnlineSession.mxs** (File path: /Online)



**Description:** EEG data and stimulation coming from Acquisition client are preprocessed with Python code applied inside Python scripting box to detect target button and send it to Unity3d by using TCP/IP.

#### Other information:

Each time the scene changes, a stop trigger is triggered to restart a designer. By storing raw data, it can be used for other studies.

Due to online performance issues, the EEG data is downscaled to 512Hz through downsampling into the Python Scripting box.

### Python module

Python code: **WorldOnline.py** (File path: /SourceCode)

**Code Description:** This module runs on the Openvibe designer's python scripting box. It receives **eegData** and **stims** from Designer, save the variables, and pass it to **classify function** of ProcessingWorld module.

\* Before viewing this description, I recommend that you study at the following link <http://openvibe.inria.fr/tutorial-using-python-with-openvibe/>

## ■ Detailed explanation

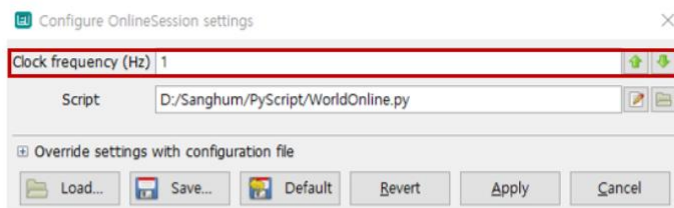
- ① **sys.path.append:** Adding the path of **ProcessingWorld** on the system path so that import the function is possible.

```
import sys
sys.path.append("C:/Users/wldk5/WorldSystem/SourceCode")
import ProcessingWorld
```

- ② **initialize function:** Called when the designer starts. It is declared as a global variable so that it can be used in *process function*. However, it does not seem to act as an intact global variable. I found that you must declare it within the *process function* to use it.

```
def initialize(self):
    print('Python initialize function started')
    global eegData, stims, trigger, filename
    eegData = np.zeros((32,1))
    stims = np.zeros((1,3))
    trigger = 0.
```

- ③ **process function:** Called repeatedly as much as the clock specified in the box configuration when the designer is running. If the clock frequency is 1, it is called once per second. The pink inverted triangle is the branch that receives or sends EEG data, and the purple inverted triangle is the branch that receives or sends stimulations.



```
def process(self):
    global eegData, stims, trigger, filename
    #Signal acquisition
    for chunkIndex in range( len(self.input[0]) ):
        if(type(self.input[0][chunkIndex]) == OVSignalHeader):
            self.signalHeader = self.input[0].pop()
        elif(type(self.input[0][chunkIndex]) == OVSignalBuffer):
            chunk = self.input[0].pop()
            signalRaw = np.array(chunk).reshape(tuple(self.signalHeader.dimensionSizes))
            eegData = np.append(eegData,signalRaw,axis=1)
    #Stimulation acquisition
    for chunkIndex in range( len(self.input[1]) ):
        chunk = self.input[1].pop()
        if(type(chunk) == OVStimulationSet):
            for stimIdx in range(len(chunk)):
                stim=chunk.pop();
                x = np.array([[stim.date,stim.identifier,0.0]])
                stims = np.append(stims, x, axis=0)
                trigger = stim.identifier
```

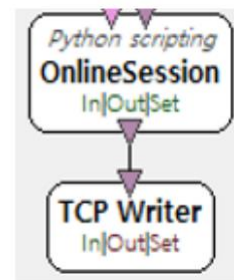
Therefore, **self.input[0]** stores the EEG signal, and **self.input[1]** stores the stimulation information.

During the process, it continues to store the stimulation number in **trigger** and run the following branch when the number is 7. It sends **eegData** and **stims** to the ProcessingWorld module's **classify function** and receive the result of detecting the target button in **result**. The stimulation is then sent to **TCP Writer** to send the target button to unity application.

```

if(trigger == 7.):
    print('got here')
    trigger = 0.
    samplingFreq = 512
    channelNum = 32
    stims = np.delete(stims,0,0)
    with open(filename,'wb') as f:
        pickle.dump([eegData, stims, samplingFreq, channelNum], f) # Saving eeg data
    result = ProcessingWorld.classify(eegData, stims, samplingFreq, channelNum)
    stimSet = OVStimulationSet(0., 0.)
    stimSet.append(OVStimulation(result, self.getCurrentTime(), 0.))
    self.output[0].append(stimSet)

```



Python code: **ProcessingWorld.py** (File path: /SourceCode)

**Code Description:** Code used to load LDA classifier and feature index information saved during the training session, preprocess online data and detect target button.

### ■ Detailed explanation

- ① **glob.glob + sorted + indexing:** Load the most recently modified classifier and feature index information.

```

def classify(eegData, stims, samplingFreq, channelNum): ### Online Preprocessing code
    #Load lda classifier and selected features
    SelectedF_path = 'C:/Users/wldk5/WorldSystem/Within/StepWise/Features/'
    Classifier_path = 'C:/Users/wldk5/WorldSystem/Within/StepWise/Classifiers/'

    current_list = []
    current_list = sorted(glob.glob(SelectedF_path + '*.pickle'), key=os.path.getmtime, reverse=True)
    SelectedF_real = current_list[0]
    with open(SelectedF_real, 'rb') as f:
        SelectedFeatures = pickle.load(f)

    current_list2 = []
    current_list2 = sorted(glob.glob(Classifier_path + '*.pickle'), key=os.path.getmtime, reverse=True)
    Classifier_real = current_list2[0]
    lda = LinearDiscriminantAnalysis(solver='lsqr', shrinkage='auto')
    lda = joblib.load(Classifier_real)

```

- ② **Preprocess:** Re-referencing(CAR) → bandpass[0.5, 10] → Epoching[200 600] → BaselineCorrection[200 600]  
→ DownsamplingEpoch[512 -> 128]

```

##EEG Preprocessing
sampleNum = eegData.shape[1]

downsampleRate = 4

#Common Average Reference
eegData = Re_referencing(eegData, channelNum, sampleNum)

#Bandpass Filter
eegData = butter_bandpass_filter(eegData, 0.5, 10, samplingFreq,4)

#Epoching
epochSampleNum = int(np.floor(0.4 * samplingFreq))
offset = int(np.floor(0.2 * samplingFreq))
baseline = int(np.floor(0.6 * samplingFreq))
[Epochs1, Num1] = Epoching(eegData, stims, 1, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[Epochs2, Num2] = Epoching(eegData, stims, 2, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[Epochs3, Num3] = Epoching(eegData, stims, 3, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[Epochs4, Num4] = Epoching(eegData, stims, 4, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[Epochs5, Num5] = Epoching(eegData, stims, 5, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[Epochs6, Num6] = Epoching(eegData, stims, 6, samplingFreq, channelNum, epochSampleNum, offset, baseline)

#Downsampling Epochs
[Epochs1, Epochs2, Epochs3, Epochs4, Epochs5, Epochs6, epochSampleNum] = DownsamplingOnlineEpoch(Epochs1, Epochs2, Epochs3, Epochs4, Epochs5, Epochs6, downsampleRate)
samplingFreq = samplingFreq/4

```

- ③ **Predict the target button:** After that, a new vector is formed by converting the feature vector to reflect the feature index information previously loaded called **SelectedFeatures**. The target button is detected through the lda predict result that takes these vectors as an input (**voting mechanism**). And return the result.



```

#Convert to feature vector
featureNum = channelNum*epochSampleNum
[Features1, Features2, Features3, Features4, Features5, Features6] = Online_Convert_to_featureVector(Epochs1, Epochs2, Epochs3, Epochs4, Epochs5, Epochs6, Num1, featureNum)

#Classification process
result = np.zeros((1,6))
Features1 = Features1[:,SelectedFeatures]
Features2 = Features2[:,SelectedFeatures]
Features3 = Features3[:,SelectedFeatures]
Features4 = Features4[:,SelectedFeatures]
Features5 = Features5[:,SelectedFeatures]
Features6 = Features6[:,SelectedFeatures]

but1_pred = lda.predict(Features1)
but2_pred = lda.predict(Features2)
but3_pred = lda.predict(Features3)
but4_pred = lda.predict(Features4)
but5_pred = lda.predict(Features5)
but6_pred = lda.predict(Features6)

result[0,0] = np.sum(but1_pred)
result[0,1] = np.sum(but2_pred)
result[0,2] = np.sum(but3_pred)
result[0,3] = np.sum(but4_pred)
result[0,4] = np.sum(but5_pred)
result[0,5] = np.sum(but6_pred)

answer = np.argmax(result) + 1
return answer

```

## Zero training Session used CNN (File path: /Zero)

The system has no training session, and it works by using the CNN model in an online session that predict the target button.

### ① CNN Model Generation Process

**Description:** This process converts 55 p300 speller data into text files through python preprocessing. Since we processed the signal with python code in WTS, we wanted to construct a CNN model with a same process.

### Raw Data

Access to data resides in the BCI LAB.

If you want to use it, please contact us by e-mail: [minkyuahn@handong.edu](mailto:minkyuahn@handong.edu)

**Data description:** 55 P300 speller train data

**Data format:** mat file

**Consist of:**

- eegData(32, sample number), samplingFreq(512), nontargetStim(1,750), targetStim(1,150)
- It is divided into two files for each subject. (ex. S01, S01\_2)

### Python Module

**Python code:** [Generate\\_Model.py](#) (File path: /SourceCode)

**Code Description:** Python preprocessing process on 55 P300 speller data and CNN model generation process on the data. (Briefly: Load Matlab data → Python preprocessing → Save the data into txt file<Two type files: All, Individual>)

### Progression Order

1. Specify the file path and name to be saved after signal processing and create an array filled with zeros to fit the data size.

```

saveTargetroot = 'C:/Users/wldk5/WorldSystem/Zero/CNNdata/55P300data/PythonData/alltarget.out'
saveNontargetroot = 'C:/Users/wldk5/WorldSystem/Zero/CNNdata/55P300data/PythonData/allnontarget.out'
root = 'C:/Users/wldk5/WorldSystem/Zero/CNNdata/55P300data/MatlabData/S'
filename2 = '_2'
filename = ''
alltarget = np.zeros((300*55,32,51))
allnontarget = np.zeros((1500*55,32,51))







```

## 2. Pass the parameters of the function from the first subject to the 55th subject in the loop.

```

for i in np.arange(1,56):
    filename = ''
    if(i<10):
        filename = root + '0' + str(i)
    else:
        filename = root + str(i)
    [alltarget[300*(i-1):300*i, :, :], allnontarget[1500*(i-1):1500*i, :, :]] = GenerateP300Data(filename)
    print("subject {0} is preprocessed".format(str(i)))

```

 S01  
 S01\_2  
 S02  
 S02\_2  
 S03  
 S03\_2

## 3. GenerateP300Data











### ■ Detailed explanation

- ① In a loop, signal processing is performed on two separate files for a subject. First, load the mat file and save the data in variables.

```

def GenerateP300Data(filename):
    target = np.zeros((300,32,51))
    nontarget = np.zeros((1500,32,51))
    for i in np.arange(1,3):
        if (i==2):
            filename = filename + '_2'
        mat = hdf5storage.loadmat(filename)
        eegData = mat['eegData']
        samplingFreq = mat['samplingFreq'][0,0]
        stimsN = mat['stimsN']
        stimsT = mat['stimsT']
        channelNum = 32
        sampleNum = eegData.shape[1]

```

 S01  
 S01\_2  
 S02  
 S02\_2  
 S03  
 S03\_2  
 S04  
 S04\_2  
 S05  
 S05\_2

- ② After that, preprocessing is performed and stored in **target** and **nontarget**.

**Preprocessing process:** Re-referencing(Common Average Reference) → Bandpass[0.5, 10] → Epoching[200 600] → BaselineCorrection[200 600] → DownsamplingEpoch[512 -> 128]

```

#Common Average Reference
eegData = Re_referencing(eegData, channelNum, sampleNum)

#Bandpass Filter
eegData = butter_bandpass_filter(eegData, 0.5, 10, samplingFreq,4)
# FFTPlotFull(eegData, samplingFreq)
# # FFTPlot(eegData, samplingFreq, 15)

#Epoching
epochSampleNum = int(np.floor(0.4 * samplingFreq))
offset = int(np.floor(0.2 * samplingFreq))
baseline = int(np.floor(0.6 * samplingFreq))
[EpochsT, NumT] = Epoching(eegData, stimsT, samplingFreq, channelNum, epochSampleNum, offset, baseline)
[EpochsN, NumN] = Epoching(eegData, stimsN, samplingFreq, channelNum, epochSampleNum, offset, baseline)
# EpochPlot(EpochsT, EpochsN, epochSampleNum, 0.2, 0.6, 30)

#Downsampling
downsampleRate = 4
samplingFreq = samplingFreq/4
[EpochsT, EpochsN, epochSampleNum] = DownsamplingEpoch(EpochsT, EpochsN, downsampleRate)

target[150*(i-1):150*i, :, :] = EpochsT
nontarget[750*(i-1):750*i, :, :] = EpochsN

```

- ③ **target** and **nontarget** for a subject are stored in text files within the function and returned to form **alltarget** and **allnontarget**.

```

tar = filename + 't.out'
ntar = filename + '_nt.out'

with open(tar, 'w') as f:
    f.write('# Array shape: {0}\n'.format(target.shape))
    for data_slice in target:
        np.savetxt(f, data_slice)
        f.write('# New slice\n')

[alltarget[300*(i-1):300*i, :, :], allnontarget[1500*(i-1):1500*i, :, :]] = GenerateP300Data(filename)
print("subject {0} is preprocessed".format(str(i)))

with open(ntar, 'w') as fl:
    fl.write('# Array shape: {0}\n'.format(nontarget.shape))
    for data_slice in nontarget:
        np.savetxt(fl, data_slice)
        fl.write('# New slice\n')

return [target, nontarget]

```

- ④ After that, **alltarget** and **allnontarget** are also saved as text files.

```

with open(saveTargetroot, 'w') as outfile:
    outfile.write('# Array shape: {0}\n'.format(alltarget.shape))
    for data_slice in alltarget:
        np.savetxt(outfile, data_slice)
        outfile.write('# New slice\n')
a = np.loadtxt(saveTargetroot)
print(a.shape)
a = a.reshape((300*55, 32, 51))
print(np.all(a == alltarget))

with open(saveNontargetroot, 'w') as outfile:
    outfile.write('# Array shape: {0}\n'.format(allnontarget.shape))
    for data_slice in allnontarget:
        np.savetxt(outfile, data_slice)
        outfile.write('# New slice\n')
b = np.loadtxt(saveNontargetroot)
print(b.shape)
b = b.reshape((1500*55, 32, 51))
print(np.all(b == allnontarget))

```

- ⑤ Data Description

- A. Target epoch (300,32,51): S01\_t.out ~ S55\_t.out
- B. Nontarget epoch (1500,32,51): S01\_nt.out ~ S55\_nt.out
- C. All Target epoch (300\*55, 32, 51): alltarget.out
- D. All Nontarget epoch (1500\*55, 32, 51): allnontarget.out

- ☐ allnontarget.out
- ☐ alltarget.out
- ☐ S01\_nt.out
- ☐ S01\_t.out
- ☐ S02\_nt.out
- ☐ S02\_t.out

## 4. CNN Model generation process

### ■ Detailed explanation

- ① **trainingData** is created by concatenating **alltarget** and **allnontarget**. Then create a **label** accordingly.

```

trainingData = np.concatenate((alltarget, allnontarget), axis = 0);
trainingData = np.reshape(trainingData, (1800*54, 1, 32, 51));

label = np.concatenate((np.ones(300*54), np.zeros(1500*54)), 0);
label = np_utils.to_categorical(label, 2);

```

- ② I created a sequential model, added what I needed, and compiled it.

(This is a transfer learning of the existing research named *P300 speller Zero CNN training* developed by Jongmin Lee who is a MS Course Student in Prof. Min Kyu Ahn's BCI Lab in Handong Global University, so I did not modify the model.)

```

##Generating CNN model
model = Sequential();
#model.add(AveragePooling2D(pool_size=(1, 4), strides=(1,4))) # this was added
model.add(Conv2D(51, kernel_size=(1, 25), data_format='channels_first', input_shape=(1, 32, 51)))
model.add(BatchNormalization())
model.add(Conv2D(51, (32, 1), data_format='channels_first'))
model.add(BatchNormalization())
model.add(Flatten())
model.add(Dense(2))
model.add(Activation('softmax'))

#model = multi_gpu_model(model, gpus=2);
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam');

```

- ③ After constructing validation data and setting patience of early stopping, I fitted the model.

```

data = trainingData;
randIdx = np.random.permutation(54*1800);
trainIdx = randIdx[0:int(1800*54*0.95)];
valIdx = randIdx[int(1800*54*0.95):54*1800];

trainData = data[trainIdx,:,:,:];
trainLabel = label[trainIdx];
valData = data[valIdx,:,:,:];
valLabel = label[valIdx];

early_stopping = EarlyStopping(patience = 3);

fittedModel = model.fit(trainData, trainLabel, epochs=10, validation_data=(valData, valLabel), callbacks=[early_stopping]);

```

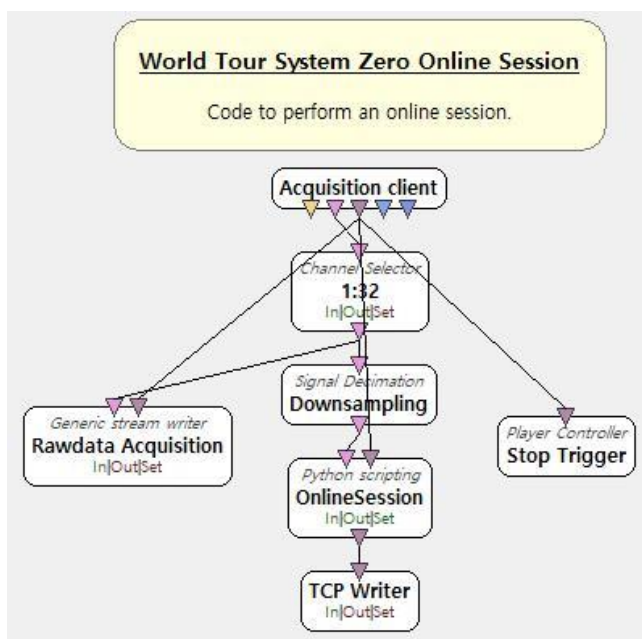
- ④ Finally, the model was saved in specified path and the code ended.

```
model.save('C:/Users/wldk5/WorldSystem/Zero/ZeroModel/ZeroCNN.h5')
```

## ② Online Session

### OpenViBE designer

Designer code: **OnlineCNN.mxs** (File path: /Online)



**Description:** EEG data and stimulation coming from Acquisition client are preprocessed with Python code applied inside Python scripting box to detect target button and send it to Unity3d by using TCP/IP.

#### Other information:

Each time the scene changes, a stop trigger is triggered to restart a designer. By storing raw data, it can be used for other studies.

Due to online performance issues, the EEG data is downscaled to 512Hz through downsampling into the Python Scripting box.

### Python Module

Python code: **WorldOnlineCNN.py** (File path: /SourceCode)

**Code Description:** This module runs on the Openvibe designer's python scripting box. It communicates with **ZeroCNN\_Main** module through txt file. Detailed description will be mentioned below.

#### ■ Detailed explanation

- ① **sys.path.append:** Adding the path of **ProcessingWorld** on the system path so that import the function is possible.

```

import sys
sys.path.append("C:/Users/wldk5/WorldSystem/SourceCode")
import ProcessingWorld

```

- ② **initialize function:** If you look at the code, you will see the variables where the path to the txt file is stored. These variables are the paths where the txt files that play the role of each name are saved. As **start\_txt** is saved, it serves



as a trigger to notify the start of the online session to the **ZeroCNN\_Main** module.

```
def initialize(self):
    print('Python initialize function started')
    global eegData, stims, trigger, eegData_txt, stims_txt, start_txt, result_txt
    eegData = np.zeros((32,1))
    stims = np.zeros((1,3))
    trigger = 0.
    #filename = 'C:/Users/Ahn-Lab/Documents/SelectedFeatures.pickle' # Training data
    eegData_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/eegData.out'
    stims_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/stims.out'
    start_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/start.out'
    result_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/result.out'
    ProcessingWorld.start_txt_trigger(start_txt)
```

- ③ **process function:** Since the front part of the process function is the same as the WTS Online session code mentioned above, I omitted the description. During the process, it continues to store the stimulation number in **trigger** and run the following branch when the number is 7.

Then it passes **eegData** and **stims** to **save\_data function** that saves these as a txt file. After that, the function waits for 8 seconds using the delay function. And through the **load\_result function**, the txt file containing the result value is loaded and the result value is stored in the **result**. The stimulation is then sent to **TCP Writer** to send the target button to unity application.

```
if(trigger == 7.):
    print('got here')
    trigger = 0.
    stims = np.delete(stims,0,0)
    ProcessingWorld.save_data(eegData, stims, eegData_txt, stims_txt)
    ProcessingWorld.delay(8)
    result = ProcessingWorld.load_result(result_txt)
    stimSet = OVStimulationSet(0., 0.)
    stimSet.append(OVStimulation(result, self.getCurrentTime(), 0.))
    self.output[0].append(stimSet)
```

### Python code: ZeroCNN\_Main.py (File path: /SourceCode)

**Code Description:** This code was written in the **Python3 module** (*Many problems occurred in implementing the CNN Model with theano, so it was implemented to separate and work with other modules.*). While operating during an online session, it predicts the target button using the CNN model and save it as a txt file. The saved txt file is read by the Python code named **WorldOnlineCNN**.

### Progression Order

1. It loads the CNN Model created earlier and saves the value in each variable according to the path of the txt file created to communicate with **WorldOnlineCNN**.

```
def main():
    #Load cnn model and predict result
    model = load_model('C:/Users/wldk5/WorldSystem/Zero/ZeroModel/ZeroCNN.h5')
    # global file_exist, file1, file2, channelNum
    eegData_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/eegData.out'
    stims_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/stims.out'
    start_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/start.out'
    result_txt = 'C:/Users/wldk5/WorldSystem/Zero/CNNtemp/result.out'
```

2. There are four while loops in this module. I will explain in order.

#### ■ While loops

- ① **First loop:** This makes the entire code run in an infinite loop.
- ② **Second loop:** It continuously checks whether a txt file corresponding to the **start\_txt** exists and exits the loop if it is found.

- ③ **Third loop:** Since it takes time for the txt files for *eegData* and *stims* to be generated in the *WorldOnlineCNN* module, It improves the entire system performance even further by waiting for 35 seconds.

**If statement that operates after the loop:** If the file exists in the *result\_txt* value, it is not the first session, so I created an if statement to remove the existing files.

- ④ **Fourth loop:** It continuously checks whether there are files stored in *eegData\_txt* and *stims\_txt*, and when it is confirmed, loads the values in the corresponding txt file and stores them in *eegData* and *stims*.

```
while True:
    #load text file
    while True:
        if os.path.isfile(start_txt):
            break
    start_time = time.time()

    while(time.time() - start_time < 35):
        pass
    if os.path.isfile(result_txt):
        os.remove(eegData_txt)
        os.remove(stims_txt)
        os.remove(result_txt)

    while True:
        if os.path.isfile(eegData_txt) & os.path.isfile(stims_txt):
            processing_time = time.time()
            os.remove(start_txt)
            eegData = np.loadtxt(eegData_txt, delimiter = ",")
            stims = np.loadtxt(stims_txt, delimiter = ",")
            break
```

3. When six epochs are generated by signal processing, weights for each epoch are generated through the predict function of the CNN model. It works by voting mechanism that takes the maximum value after summing each saved weight. After that, *result* is stored in *result\_txt*.

```
a1 = model.predict(Epochs1)
a2 = model.predict(Epochs2)
a3 = model.predict(Epochs3)
a4 = model.predict(Epochs4)
a5 = model.predict(Epochs5)
a6 = model.predict(Epochs6)
```

→

```
result[0,0] = np.sum(a1[:,1])
result[0,1] = np.sum(a2[:,1])
result[0,2] = np.sum(a3[:,1])
result[0,3] = np.sum(a4[:,1])
result[0,4] = np.sum(a5[:,1])
result[0,5] = np.sum(a6[:,1])
```

→

```
answer = np.argmax(result) + 1
answer = [answer]
np.savetxt(result_txt, answer)
```

## Trouble Shooting

### 1. Python 2.7 library import

I am using Anaconda3 now, and Opnenvibe works by loading the Python 2.7 module. That's why you need to install the same library in Python 2.7.

In my case, there is a pip file in C:\Python27\Scripts, so I went in and proceeded to install the library.

Like this: C:\Python27\Scripts>pip install numpy

### 2. joblib.load

- ① **ImportError:** No module named sci-kit learn error

This will be happened about sklearn version conflict. Matching the package version will fix it.

- ② **Scikit-learn, Joblib version conflict:**

```

3 Messages
Clear DEBUG BENCH TRACE INF WARNING IMPORTANT WARNING ERROR FATAL Show pop-up on Warnings and Errors
Search for :
Python initialize function started
[ INF ] At time 78.703 sec <Box algorithm::(0x00007cf0, 0x0000663a) aka OnlineSession> got here
Traceback (most recent call last):
  File "C:/Program Files (x86)/openvibe-2.1.0//share/openvibe/plugins/python/openvibe.py", line 173, in realProcess
    self.process()
  File "C:/Users/wldk5/WorldSystem/Within/SourceCode/WorldOnline.py", line 45, in process
    result = ProcessingWorld.classify(eegData, stims, samplingFreq, channelNum)
  File "C:/Users/wldk5/WorldSystem/SourceCode/ProcessingWorld.py", line 164, in classify
    lda = joblib.load(Classifier_real)
  File "C:/Python27/lib/site-packages/sklearn/externals/joblib/numpy_pickle.py", line 598, in load
    obj = _unpickle(fobj, filename, mmap_mode)
  File "C:/Python27/lib/site-packages/sklearn/externals/joblib/numpy_pickle.py", line 526, in _unpickle
    obj = unpickler.load()
  File "C:/Python27/Lib/pickle.py", line 864, in load
    dispatch[key](self)
  dispatch[key](self)
KeyError: '\x00'

```

Anaconda cloud Joblib 0.14.0 link: <https://anaconda.org/conda-forge/joblib>

Execute **Anaconda Powershell Prompt** and enter the command: `conda install -c conda-forge joblib`

And execute **Command Prompt** and enter the python27/Script path and then: `pip install scikit-learn==0.20.1`

The solution to this was pretty banal: Without being aware of it I was using the version of `joblib` in `sklearn.externals.joblib` for the pickling, but a newer version of `joblib` for unpickling the object. The problem was resolved when I used the newer version of `joblib` for both tasks.

### 3. pip operation

```

C:\Python27\Scripts>pip freeze
Traceback (most recent call last):
  File "c:\python27\lib\runpy.py", line 174, in _run_module_as_main
    "__main__", fname, loader, pkg_name)
  File "c:\python27\lib\runpy.py", line 72, in _run_code
    exec code in run_globals
  File "C:\Python27\Scripts\pip.exe\__main__.py", line 9, in <module>
TypeError: 'module' object is not callable

```

Installing new pip via easy\_install will solve it.

```

C:\Python27\Scripts>easy_install pip
Searching for pip
Best match: pip 19.3.1
Adding pip 19.3.1 to easy-install.pth file
Installing pip-script.py script to c:\python27\Scripts
Installing pip.exe script to c:\python27\Scripts
Installing pip.exe.manifest script to c:\python27\Scripts
Installing pip3-script.py script to c:\python27\Scripts
Installing pip3.7.exe script to c:\python27\Scripts
Installing pip3.7.exe.manifest script to c:\python27\Scripts
Installing pip3-script.py script to c:\python27\Scripts
Installing pip3.exe script to c:\python27\Scripts
Installing pip3.exe.manifest script to c:\python27\Scripts

Using c:\python27\lib\site-packages
Processing dependencies for pip
Finished processing dependencies for pip

C:\Python27\Scripts>pip freeze
DEPRECATION: Python 2.7 will reach the end of its life on Jan
n't be maintained after that date. A future version of pip wi
support in pip, can be found at https://pip.pypa.io/en/late
WARNING: Could not generate requirement for distribution -ip
p==19.3.1: Expected W:(abcd...)
numpy==1.16.4
scipy==1.2.2

```

### 4. Required Library (File path: /PythonPackageInfo)

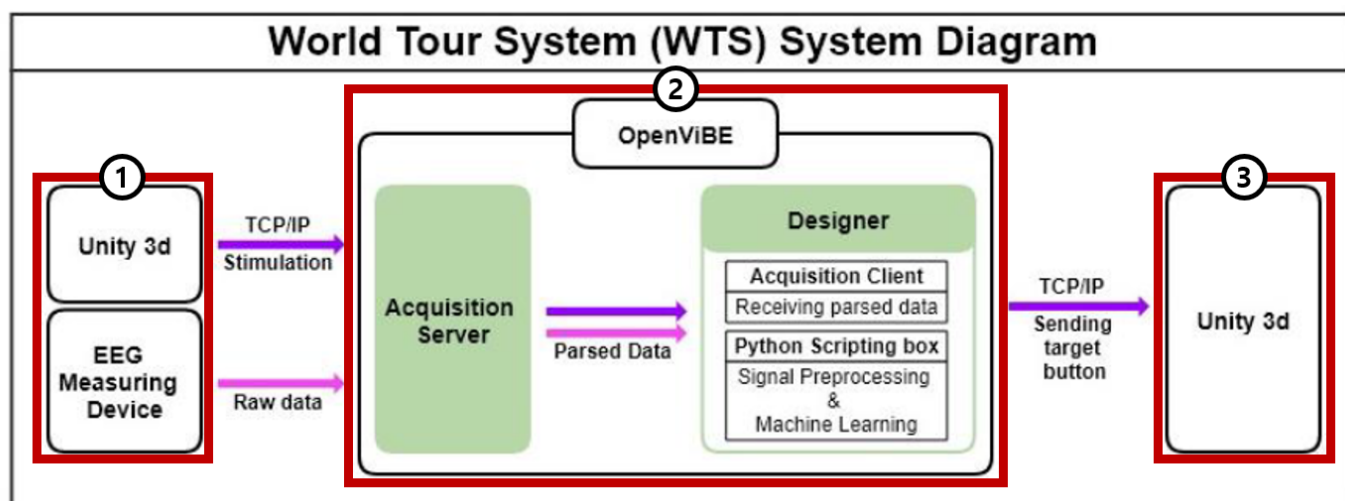
#### Python 3.6 Module

scikit-learn, hdf5storage, tensorflow, keras, joblib, statsmodels, pandas, pickleshare, matplotlib, scipy, numpy

#### Python 2.7 Module

scikit-learn, joblib, scipy, numpy

## How to Start



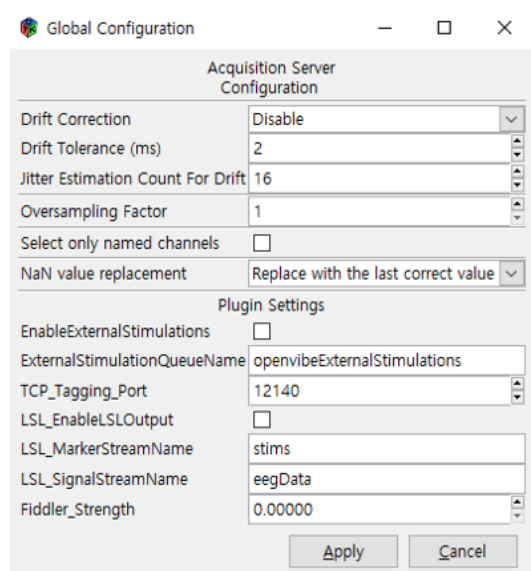
### 1. Simultaneously received brain waves and button signals through Openvibe Acquisition Server.

**Brain wave:** Measured in Biosemi equipment. Acquisition Server is built to receive this signal as a digital value.

**Button signal:** The button blinks over the UI of Unity 3d, and the blinking signal passes over TCP/IP communication to Acquisition Server.



- ① Select the brainwave measuring device driver after running Openvibe Acquisition Server



- ② Preferences - Specify TCP\_Tagging\_Port.  
(Must match the communication port inside Unity)

```
theSender = new StimulusSender();
theSender.open("localhost", 12140);
theListener = new StimulusSender();
theListener.open("localhost", 12240);
```

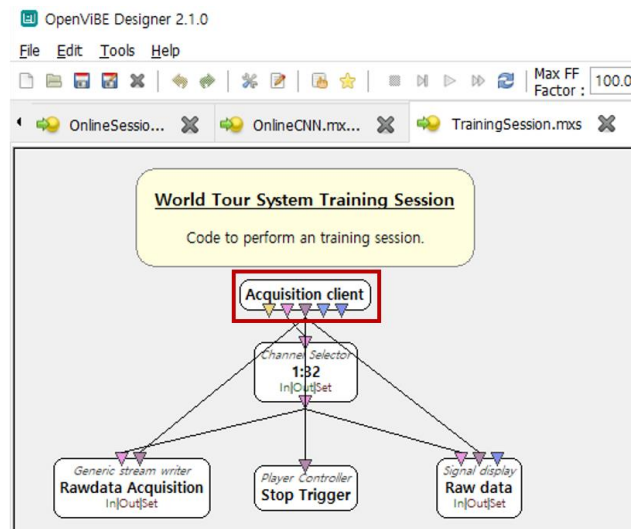
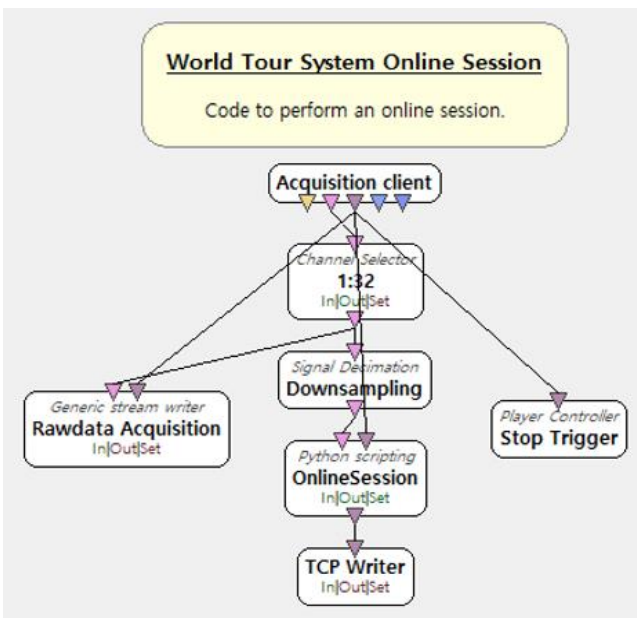


```
array([array(['FP1'], dtype='<U3'), array(['AF3'], dtype='<U3'), array(['O2'], dtype='<U2'), array(['P04'], dtype='<U3'),
array(['F7'], dtype='<U2'), array(['F3'], dtype='<U2'), array(['P4'], dtype='<U2'), array(['P8'], dtype='<U2'),
array(['FC1'], dtype='<U3'), array(['FC5'], dtype='<U3'), array(['CP6'], dtype='<U3'), array(['CP2'], dtype='<U3'),
array(['T7'], dtype='<U2'), array(['C3'], dtype='<U2'), array(['C4'], dtype='<U2'), array(['T8'], dtype='<U2'),
array(['CP1'], dtype='<U3'), array(['CP5'], dtype='<U3'), array(['FC6'], dtype='<U3'), array(['FC2'], dtype='<U3'),
array(['P7'], dtype='<U2'), array(['P3'], dtype='<U2'), array(['F4'], dtype='<U2'), array(['F8'], dtype='<U2'),
array(['Pz'], dtype='<U2'), array(['P03'], dtype='<U3'), array(['AF4'], dtype='<U3'), array(['FP2'], dtype='<U3'),
array(['O1'], dtype='<U2'), array(['Oz'], dtype='<U2'), array(['Fz'], dtype='<U2'), array(['Cz'], dtype='<U2'])])
```

- ③ Driver Properties → Change Channel names → Set channel name

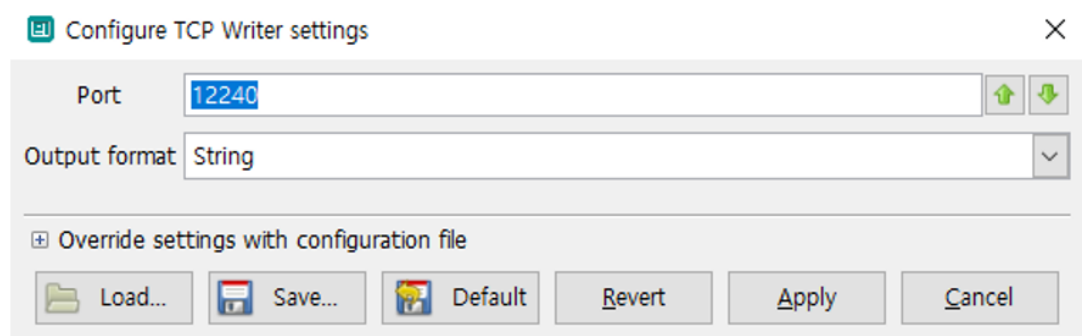
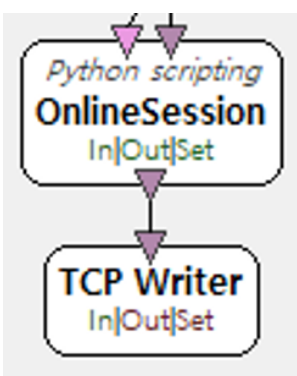
## 2. The brain waves and button signals received by the Openvibe Acquisition Server are transmitted to the Acquisition Client on the Openvibe Designer award, and then the Python Scripting box is used to detect the Target button and send it to the TCP Writer.

- ① Receive brain waves and button signals from Acquisition Server from the Client to perform signal processing in Designer.



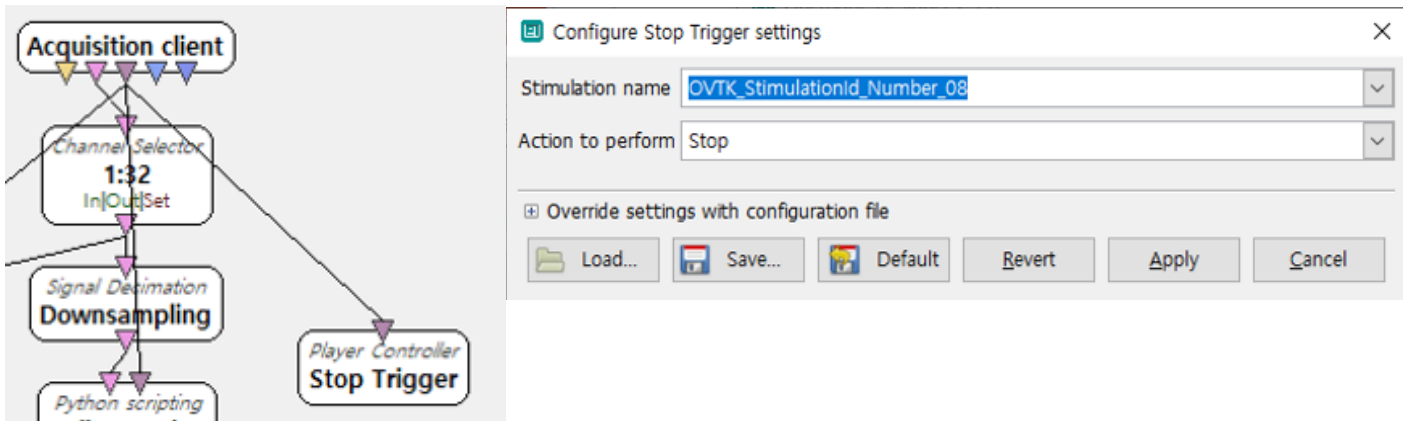
- ② Python Scripting box is used to detect the Target button and send it to the TCP Writer.

## 3. The String value for the target of the Python module is transmitted to the TCP Writer and then passed to Unity via TCP communication, which detects and changes the scene.



- ① The port on the TCP Writer must match the receiving instance's port on Unity.

- ② Before switching, restart Openvibe Designer by sending a button signal to start new Scene.



- ③ The relevant Unity Code section is as follows:

```
output = theListener.receive();
outp = output.Substring(0, 28);
theSender.send(start);
theSender.close();
theListener.close();
System.Threading.Thread.Sleep(1000);
FileStream f = new FileStream(Application.dataPath + "/StreamingAssets/" + InputName.patient_id + ".txt", FileMode.Append, FileAccess.Write);
StreamWriter writer = new StreamWriter(f, System.Text.Encoding.Unicode);
writer.WriteLine("Order: " + random + " / Result: " + outp[27]);
writer.Close();
switch (outp)
{
    case "OVTK_StimulationId_Number_01":
        SceneManager.LoadScene("NorthAmerica");
        break;
    case "OVTK_StimulationId_Number_02":
        SceneManager.LoadScene("Europe");
        break;
    case "OVTK_StimulationId_Number_03":
        SceneManager.LoadScene("Asia");
        break;
    case "OVTK_StimulationId_Number_04":
        SceneManager.LoadScene("SouthAmerica");
        break;
    case "OVTK_StimulationId_Number_05":
        SceneManager.LoadScene("Africa");
        break;
    case "OVTK_StimulationId_Number_06":
        SceneManager.LoadScene("Oceania");
        break;
}
```