\chapter{Implementation}

\label{ch:implementation}

\section{Overview}

Taking the brain signals from a subject and feed it to a neural network consists of certain challenges on the way:

\begin{itemize}

\item Uploading the ModularBCI driver to the OpenViBE Acquisition Server

\item Randomizing the trial order of each class for the runs during the experiments

\item Synchronizing data acquisiton with the PsychoPy experiment

\item Sending the EEG data from OpenViBE designer to another Python script in real-time

\item Preprocessing the brain signal data for the neural network

\end{itemize}

\section{Driver Setup}

\section{PsychoPy-OpenViBE Synchronization}

One of the challenges whilst acquiring data was the time precision of the data that was sampled from the subject. As the queue is given to the subject for motor-movement, OpenViBE Designer has to simultaneously start recording the brain signals on another laptop. Likewise, the acquisition has to stop at the same moment as the queue ends.

Our first approach to this issue was handling it manually by starting-stopping the acquisitions by hand when the queue was given. However, this approach has quickly proven to be unusable as a human error of even 0.5 seconds meant that 12.5 percent of the acquired data was faulty.

The approach that has stood out was to convert the PsychoPy experiment into a Python script and to use the "Socket" library to connect to the Acquisition Server using TCP Tagging. As the queue starts and ends, the Python script from PsychoPy sends a byte array to the Acquisition Server as event markers. These markers can then be extracted from the saved CSV files on the "Event Id" column.

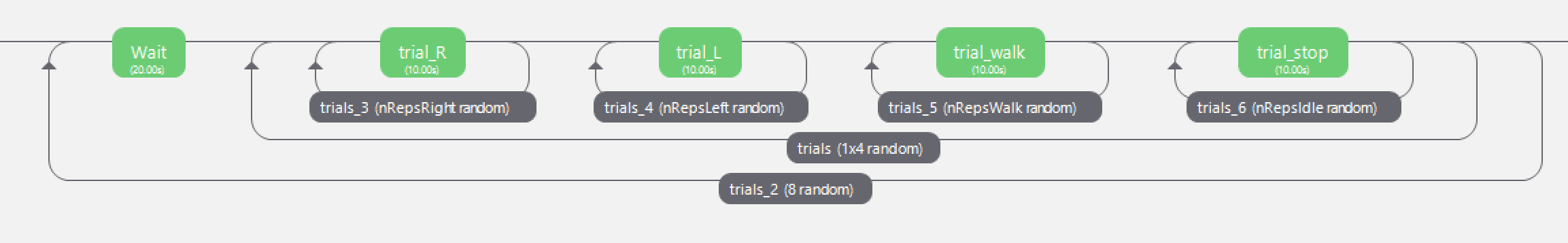
\subsection{TCP Tagging}

TCP tagging is a plugin in Acquisition Server. This feature is used to align events and the data stream timewise. It opens a port from the computer's IP address that can be accessed using the "Socket" library inside Python. This port constantly checks for external stimulations and adds this stimulation as the event marker to the input data stream.

\section{Randomizing Trial Order }

In order to prevent the subject from getting conditioned to the order of the trials, we have implemented a structure that randomizes the trials within a run.

The randomization structure relies upon a looping mechanism. Trials from each class are wrapped around a small loop that are then wrapped around a bigger loop, making up a run. An attached Excel file to the outer loop functions as a conditions list controlling the number of looping repetitions within the inner loops. The conditions list holds four conditions: 1000, 0100, 0010, 0001; meaning only one inner loop get to be run on each phase. These conditions get randomized and distributed into the parameters of the inner loops. When all four conditions randomly take place, the run ends and the flow diagram enters into a "Wait" node.



\section{OpenViBE Designer and Python Script Communication}

In order to make predictions based on the trained model, it is necessary to send the acquired brain signals from OpenViBE Designer to an external Python script. The "LSL Export" box inside OpenViBE Designer is used for sending the brain signal stream to the script. Then, the input stream of type "float32" created through the "LSL Export" box can be externally read using the "pylsl" library inside the Python script. This script saves the brain signals of 4 seconds duration into an array and feeds it to the pretrained model for predictions.

\subsection{LSL Export}

LSL (LabStreamingLayer) is a mechanism used to synchronize the data streams from OpenViBE for external real-time data analysis. It sends out a stream of a certain type, in this case "EEG". It is found as a box algorithm inside OpenViBE.

\section{Preprocessing for the Neural Network }

* Buraya Dataloader ile ilgili bir paragraf lazim.

The acquired brain signals from the Acquisition Server cannot be directly fed into the neural network as it contains a lot of noise and a very high amplitude, which might cause some gradients to explode during training. To eliminate the noise, the data first goes through a fourth-order low-pass Butterworth filter canceling most of the high-frequency oscillations. The filtered data then gets normalized to a reasonable amplitude in the [0-1] interval. 