Main.py

This is the main of the entire project. It begins by paring command line options to determine what the user wants to do.

Next, it uses edf2acii tool to convert all .edf files to \*data.txt in the directory specified by –p or –edfpath option. The default .edf file path is set to where main.py is located.

Next, all \*data.txt files are pre-filtered to only use signals in C4, F4, O2, LOC, ROC, EMG1, and EMG2.

Next, the pre-filtered signals are BP filtered according to their type (EEG, EMG). All signals are Notch filter on 60 HZ to remove power noise. The signals are then converted into epochs.

If –store\_filtered\_signals option is set, all epochs’ signal data are stored into different csv files called \*filter\_signals.csv.

Utility

This folder stores tools that are directly used by main.py.

utility.py

* **def parse\_arguments(argv)**

Parses the command-line options

-p PATH, --edfpath PATH PATH is the complete path to all .edf files.

-d, --debug Print program progress to console.

--store-filtered-signals For each \*data.txt, create \*filtered\_signals.csv

-h, --help Print command line options and exits the program

Signalprocessing

This folder contains tools to filter signal data.

filter.py

* def band\_pass\_filter

Apply band-pass filter to signals

(<http://en.wikipedia.org/wiki/Band-pass_filter>)

Args:

signal\_input: the input data (1d-array).

lower\_bound\_freq: the lower bound frequency below which the signal must be completely attenuated.

higher\_bound\_freq: the higher bound frequency above which the signal must be completely attenuated.

sampling\_freq: the sampling frequency of the recorded data.

filter\_order: a number showing the complexity of the filter structure `remez` function get the number of taps as input. The number of taps is the number of terms in the filter, or the filter order plus one.

Returns:

filtered\_signal: the filtered signal (1d-array)

filter\_coefficients: the coefficients of the optimal filter for the band pass filter

* def notch\_filer

Apply notch (band-stop) filter to signals (<http://en.wikipedia.org/wiki/Band-stop_filter>)

Recursive filter (<http://www.analog.com/static/imported-files/tech_docs/dsp_book_Ch19.pdf>)

Args:

signal\_input: the input data (1d-array).

notch\_freq: the main frequency to be filtered from the signal (for example, 60 Hz transmission line noise)

band\_width: the band width around the notch\_freq center whose containing frequencies also must be attenuated but not in the same level as notch\_freq center

sampling\_freq: the sampling frequency of the recorded data.

Note that:

The notch filter is designed using "Recursive Filters" techniques explained here (Page 326):

<http://www.analog.com/static/imported-files/tech_docs/dsp_book_Ch19.pdf>

y[n] = a\_0 \* x[n] + a\_1 \* x[n-1] + a\_2 \* x[n-2] + b\_1 \* y[n-1] + b\_2 \* y[n-2]

(x\_i are the inputs, y\_i are the outputs)

Returns:

filtered\_signal: the filtered signal (1d-array)

filter\_coefficients: the coefficients of the optimal filter for the notch filter

* def low\_pass\_filter

Apply low pass filtering (<http://en.wikipedia.org/wiki/Low-pass_filter>)

Args:

signal\_input: the input data (1d-array).

cut\_off\_freq: frequency above which the signals are attenuated

sampling\_freq: the sampling frequency of the recorded data.

filter\_order: a number showing the complexity of the filter structure

`remez` function get the number of taps as input. The number of taps is thenumber of terms in the filter, or the filter order plus one.

Returns:

filtered\_signal: the filtered signal (1d-array)

filter\_coefficients: the coefficients of the optimal filter for the low pass filter

* def high\_pass\_filter

Apply high pass filtering

(<http://en.wikipedia.org/wiki/High-pass_filter>)

Args:

signal\_input: the input data (1d-array).

cut\_off\_freq: frequency above which the signals are attenuated

sampling\_freq: the sampling frequency of the recorded data.

filter\_order: a number showing the complexity of the filter structure

`remez` function get the number of taps as input. The number of taps is thenumber of terms in the filter, or the filter order plus one.

Returns:

filtered\_signal: the filtered signal (1d-array)

filter\_coefficients: the coefficients of the optimal filter for the low pass filter

test/test\_filter.py

This sub-folder contains the tests for the filtering techniques

* def test\_band\_pass\_filter

Test that the signals outside the lower and higher frequency bounds are attenuated by at least 3db while the signals within are not.

* def test\_notch\_filter

Test that the signals within the notch filter are attenuated by at least 3db while the signals outside are not.

* def test\_low\_pass\_filter

Test that the signals above the cutoff frequency are attenuated by at least 3db while the signals below are not.

* def test\_high\_pass\_filter

Test that the signals below the cutoff frequency are attenuated by at least 3db while the signals above are not.

Dataprocessing

This folder contains tools to transform .edf files into ascii files and select signals that are useful for the analysis.

Parsing.py

* def convert\_edf\_to\_ascii

Recursively convert all .edf files into ascii files using edf2ascii tool

Args:

directory\_path: Full path to the directory that contains all the .edf data files

log: option to log the progress of the function

* def data\_pre\_filtering

Recursively parse all files at the directory and store data into dictionary

Args:

directory\_path: Full path to the directory that contains all the data files

log: option to log the progress of the function

Returns:

signal\_dictionary: dictionary of file\_data dictionaries, each containing lists of unfiltered EEG and EMG data

* def signals\_to\_epochs

Filter the signal data using BP and Notch filters. Aggregate the signal data into epochs (30 seconds per epoch, 30 \* sampling\_frequency data per epoch)

Args:

signal\_dictionary: dictionary of lists of EEG and EMG data

log: option to log the progress of the function

Returns:

epochs: dictionary of data\_file dictionaries, each containing dictionaries of lists of filtered EEG and EMG data

* def save\_filtered\_signals\_to\_csv

Save data stored in epochs to a csv file

Args:

epochs: dictionary of data\_file dictionaries, each containing dictionaries of lists of filtered EEG and EMG data

directory\_path: Full path to the directory where the csv file is going to be saved

log: option to log the progress of the function

edf2ascii

Public tool that converts .edf file into ascii format (\*data.txt)

For different environment, a different binary of edf2ascii needs to be downloaded from <http://www.teuniz.net/edf2ascii/>

Configuration

This folder contains default and constant variables

Defaults.py

# Data Parsing Constants

F4\_SIGNAL\_CHANNEL = 4

C4\_SIGNAL\_CHANNEL = 6

O2\_SIGNAL\_CHANNEL = 9

LOC\_SIGNAL\_CHANNEL = 13

ROC\_SIGNAL\_CHANNEL = 14

EMG1\_SIGNAL\_CHANNEL = 21

EMG2\_SIGNAL\_CHANNEL = 22

# Signal Names

F4 = 'F4'

C4 = 'C4'

O2 = 'O2'

LOC = 'LOC'

ROC = 'ROC'

EMG1 = 'EMG1'

EMG2 = 'EMG2'

# EDF File Defaults

EDF\_FILE\_PATH = '.'

# Filter Defaults

SAMPLING\_FREQUENCY = 256

EEG\_LOW\_BOUND\_FREQUENCY = 0.3

EEG\_HIGH\_BOUND\_FREQUENCY = 35

EMG\_LOW\_BOUND\_FREQUENCY = 10

EMG\_HIGH\_BOUND\_FREQUENCY = 100

NOTCH\_FREQUENCY = 60

NOTCH\_FREQUENCY\_BANDWIDTH = 2

# Epoch Constants

EPOCH\_SIZE = 30

NUMBER\_OF\_DATA\_PER\_EPOCH = EPOCH\_SIZE \* SAMPLING\_FREQUENCY