**Read\_Me Online\_EMG**

This document will explain the pipeline for Online/live motor unit decomposition.

**Files in this folder:**

- EMG\_Classes.py: Contains EMG, Offline\_EMG and Online\_EMG classes

- Example\_Decomp\_pipeline.py: runs the whole pipeline for live decomposition of motor units.

- Example\_Online\_EMG.py: runs live decomposition based on decomposition data (optional to do offline\_decompostion as well)

- offline\_EMG.py: Contains EMG\_Decomposition that runs the offline EMG decomposition. It uses the offline\_EMG class from EMG\_classes.py and the processing\_tool.py.

- online\_emg\_plotter\_helper.py: Contains live decomposition preprocessing (applying MUfilters to live data) and calculates the discharge rate.

- online\_emg\_plotter.py: Contains the plotter's specific settings (initialization of channels) in live decomposition.

- online\_emg\_chart.py: Contains the specific settings (color, thickness, labels and y ticks) for every channel chart in live decomposition.

- processing\_tools.py: Contains all processing tools for both live and offline decomposition.

**EMG\_Classes.py:**

Contains EMG, Offline\_EMG and Online\_EMG classes. EMG() is the parent class of both Offline\_EMG and Online\_EMG and contains processing settings for both classes. This contains the number of iterations that the algorithm does, before saving the results.

Offline\_EMG(): contains different functions for the offline decomposition of motor units. The EMG\_Decomposition() class off offline\_EMG.py contains the pipeline for offline decomposition and will call the functions of this class. Offline\_EMG() functions use functions off processing\_tools.py for several processing steps.

Online\_EMG(): Extracts found motor unit filters of offline decomposition with preprocess\_with\_biofeedback(). Callback() applies these MUfilters on live data. After a live decomposition measurement, it applies the motor unit filters on the recorded data with the process() function to save the found pulse trains and MUpulses in a \_decomp.json file.

**Example\_Online\_EMG.py:**

Enables all the right channels and configurations for the measurement. The force channel and grid type must be changed according to the measurement setup.

If with\_decomp is set to True, a .poly5 file of a training measurement can be selected and offline decomposition will precede the live decomposition.

After, it initializes and runs a Setup\_Experiment instance in which the right kind of measurement will be set up and the live plotter will be launched.

Called objects:

Setup\_Experiment(): (part of the Test\_Scripts\_Nathan folder)

Asks what kind of measurement you want to do: calibration, MVC, training, or live decomposition

Asks what kind of file type you want to use for the measurement.

Initializes the onlineEMGplotterHelper, GUI\_experiment and right file writer instances.

Launches the app.

Gui\_Experiment(): (part of the Test\_Scripts\_Nathan folder)

Creates the windows and sets the force channel and force profile to be displayed.

Also initiates the execution of a measurement when the main window is closed.

**offline\_EMG.py:**

Contains the EMG\_Decomposition() class. This class contains the whole pipeline for offline decomposition based on training data. It saves the EMG decomposition results for biofeedback (live decomposition).

**OnlineEMGplotterhelper.py: (child class of the FilteredSignalPlotterHelper)**

\_\_init\_\_():

Initializes the right settings for filtering and creates an onlineEMG() object.

**This onlineEMG object will first load the data of a selected decomp.json file and extract the motor unit filters.**

**Functions:**

Initialize():

Initialises channel names, units and types for the data that must be saved.

Then the channel names, units and types are extracted for the data to be plotted. These are the force channel, force profile, and the amount of motor units found.

Initializes an extra file writer to write away the data and uses CHANGED file writers.

initialize\_file\_writer():

Initializes a ForceFileWriter, which is an instance of the right file writer.

The poly5 and xdf file writers are changed to be able to handle the added force profile.

Start(): Waits for user input in the app before starting a live decomposition.

Includes the preview button to preview the force profile.

Start\_live(): initializes the force profile for live decomposition.

Runs the MVC\_Extraction.py which opens the file director and

extracts the offset and mvc value from a self-chosen file.

Initializes the Consumer, ConsumerThread, and Monitor instances and starts the measurement.

Callback():

Gets the data from the Monitor instance and plots it in 2 different windows.

Main window contains the online EMG plotter and the secondary window has the normal signal plotter

Load cell data gets plotted and saved normalized based on extracted MVC values.

**\*\* NEW DATA FROM CHANNELS 1 TILL -3 (R#C# CHANNELS) WILL BE PUT IN online\_emg.callback() FUNCTION AND BINARY DISCHARGE TIMES WILL BE GIVEN BACK. THE DISCHARGE RATE OF THIS NEW DATA BLOCK BE CALCULATED WITH THE calculate\_disrate() FUNCTION.\*\***

put\_sample\_data():

Reshapes the new send data from the device in the right shape and puts it in the SampleData instance.

Puts it in the Queue of the ForceFileWriter.

calculate\_disrate():

calculates the discharge rate for every new sample over a 1-second window.

Some extra added functions that are self-explanatory.

MVC\_Extraction.py: (same as in Test\_Scripts\_Nathan folder)

Loads a .poly5 or .xdf file and calculates the offset and maximum value of the MVC measurement based on found plateaus. The criteria for a plateau are

- a big change precedes the plateau (np.diff between consecutive samples is large)

- the level is at least bigger than

**online\_emg\_plotter.py: (Based of SignalPlotter)**

Changes made in initialize\_channels\_components(), update\_chart() and get\_channel\_info()

initialize\_channels\_components():

Creates the channels for force channel, force profile, and motor units

Any channel that is no real channel of the device (such as the force profile)

will get unit = 'a.u.' and ChannelType.Unknown. Initializes the force channel,

force profile and all motor unit ChannelComponents.

update\_chart():

#if self.\_compute\_autoscale: deleted, so the y range gets updated each iteration.

The standard scale value for motor units is 50 since this is the maximum discharge rate that can be detected by the decomposition algorithm.

The offset is set to the middle value of the force profile.

In this way, it follows the force profile with a range of +/- 1.5\*self.scale.

**\*\*\* IN THE DEMO MEETING, WE AGREED TO USE A STANDARD SCALE WITH OFFSET 0 FOR FORCE PROFILE AND CHANNEL, SO NEEDS TO BE CHANGED \*\*\*.**

get\_channel\_info():

Extracts info of the channels that need to be saved with the custom force file writers.

Any channel that is no real channel of the device (such as the force profile and motor units)

will get unit = 'a.u.' and ChannelType.Unknown.

**online\_emg\_chart.py:**

Changes made in setup\_signals(), update\_y\_ticks(), generate\_rainbow\_colors():

Changes are discussed in the code

**processing\_tools.py:**

Contains all processing tools for both live and offline decomposition.

Added the get\_pulse\_trains(), get\_mu\_filters(), get\_online\_parameters() and getspikesonline() compared to the code of Lisette.

**At the end:**

The pipeline creates 2 file for each live recording

- one file with the original data and the force profile appended to it (name: Online\_EMG+ForceProfile\_{date}\_{time}.{file type})

- one file with the decomposition results. (name: Online\_EMG+ForceProfile\_{date}\_{time}\_decomp.json), which can be opened with OpenHDEmg

**\*\*\* Important considerations:**

* The filtering methods (hpf, lpf and order) in training and live decomposition/online\_emg must be the same, else the motor unit filters will not work on the live data and the results will be unreliable. Therefore, there is also no additional filtering in the offline decomposition. Any bad channels (high RMS or impedance) must be excluded before offline decomposition. This can be done with emg\_reject in the example\_online\_emg.Py, which creates a mask in offline decomposition and live decomposition. The same emg\_reject must be used in live decomposition. This is necessary because else the motor unit filters will be unreliable or the filters will not have the right shape.
* emg\_reject: contains the indices of channels that must be excluded. However, since decomposition is only done on the grid channel data (r#c# channels) and the new Python interface includes the CREF channel, the channel index differs 1 index from the index in the buffer array. For example, R1C8 is index 1 in the callback\_object['buffer'].dataset, but index 0 in the decomposition algorithm. So index 0 must be added in emg\_reject = [] to exclude r1c8.
* Any additional AUX or BIP channels outside of the load cell channel must be deactivated, else the indexing of the emg data will go wrong in the callback function of OnlineEMGPlotterHelper. This must be changed, however, for me there is no time left to do it.
* For .poly5 files a different reader (poly5\_force\_file\_reader in TestScriptsNathan) is used that does not reorder the channels based on the name of the channels, because else the motor units filters also need to be reordered. So the channel ordering is for both training, offline decomposition and live decomposition the same as the order in which the saga device delivers the data to the callback function.
* The whole pipeline has only be tested for .poly5 files. The XDF file type must be tested to see if there are no errors in the pipeline.

**\*\*\***