



December 14th 2022

OpenViBE: an open source BCI
software suite

Concluding Remarks - OpenViBE & more...

Arthur Desbois, Marie-Constance Corsi

ARAMIS team, Paris Brain Institute

PART 3 - Concluding remarks & perspectives

3.1 - OpenViBE & more...

- As seen in Part 2, with OpenViBE you can easily prototype and design BCI protocols and experiments

- Lots of **scenario examples and templates** are already available in the install!

```
<openvibe-3.3.0-64bit>\share\openvibe\scenarios\bci-examples
```

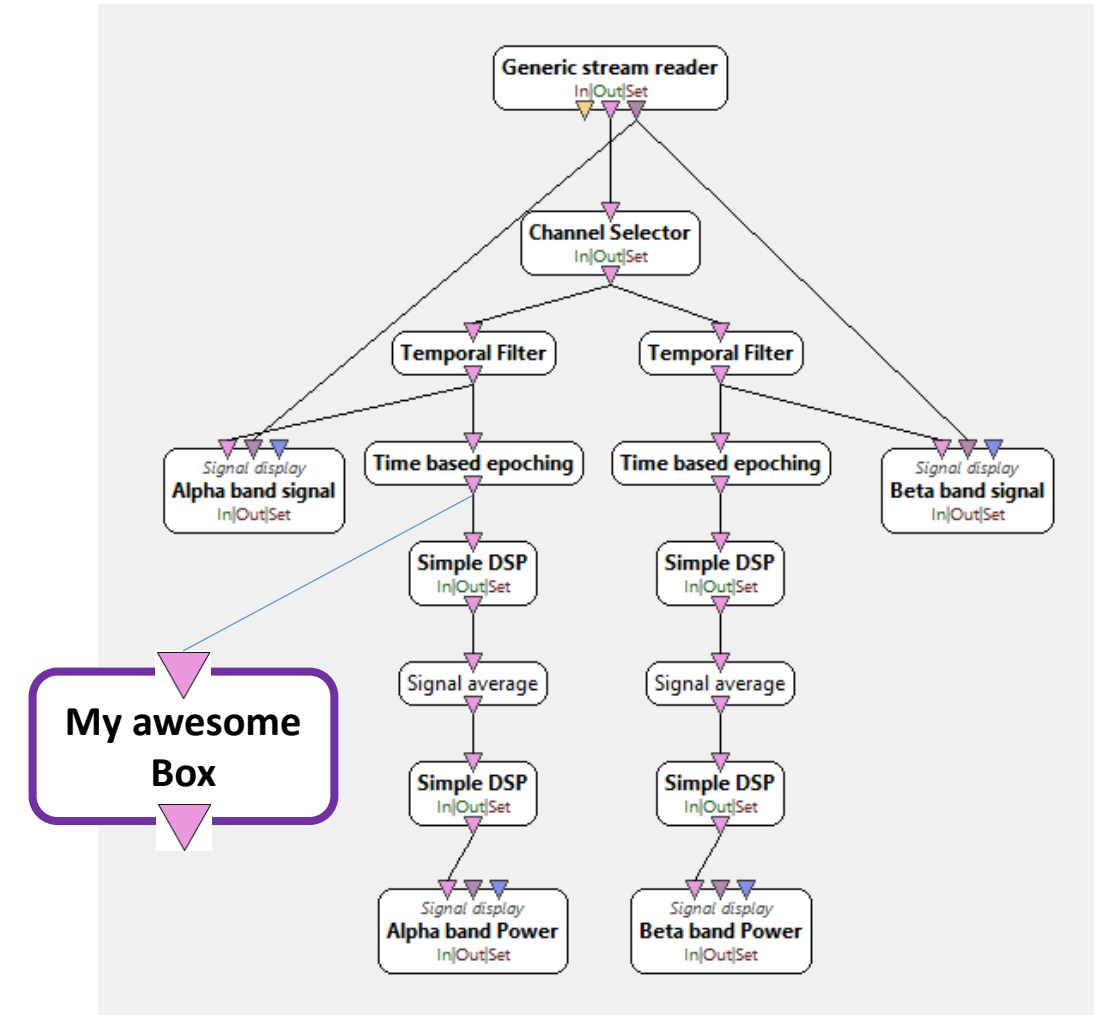
- Want to use a particular box? Tutorial scenarios are there for you:

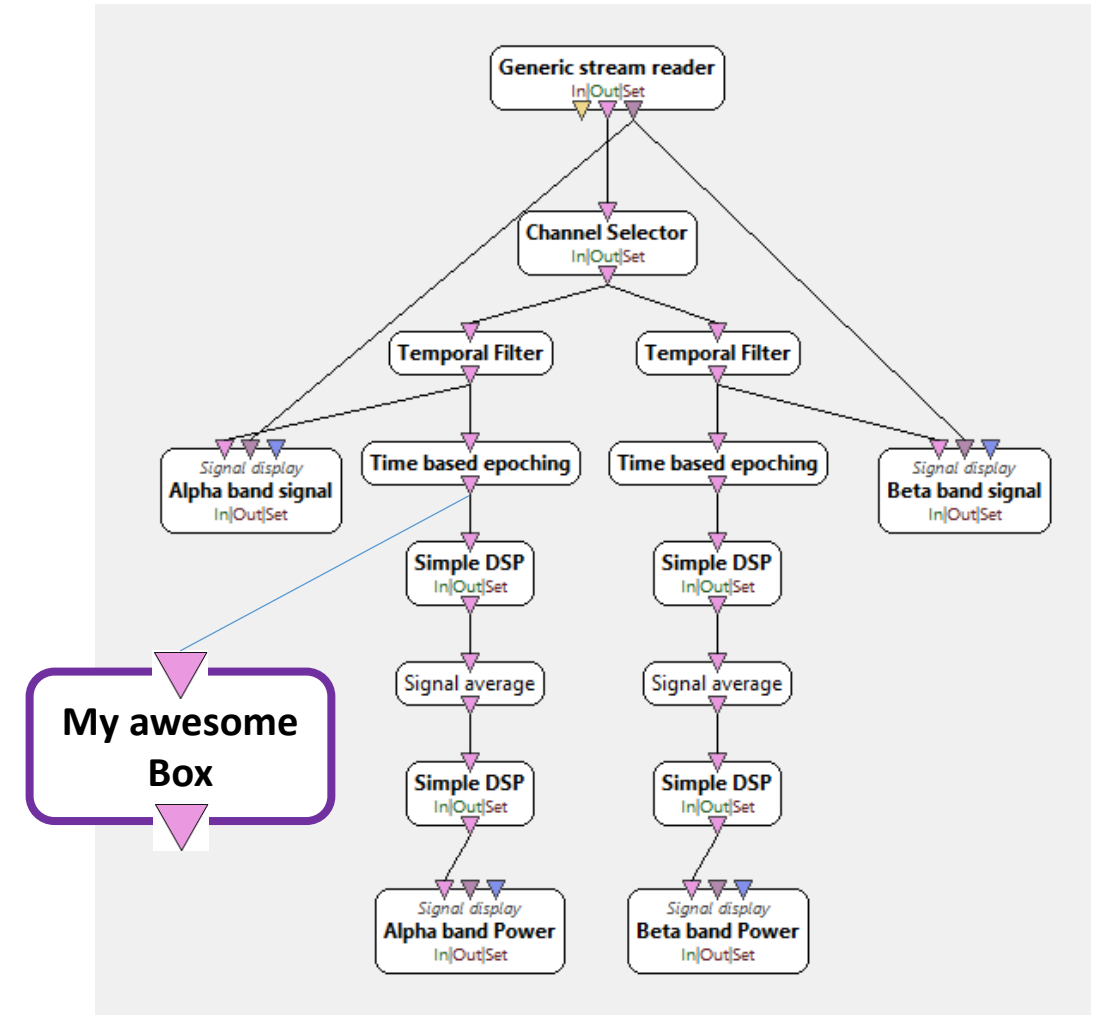
```
<openvibe-3.3.0-64bit>\share\openvibe\scenarios\box-tutorials
```

- Check the general documentation for a great amount of info:
<http://openvibe.inria.fr/documentation-index/>



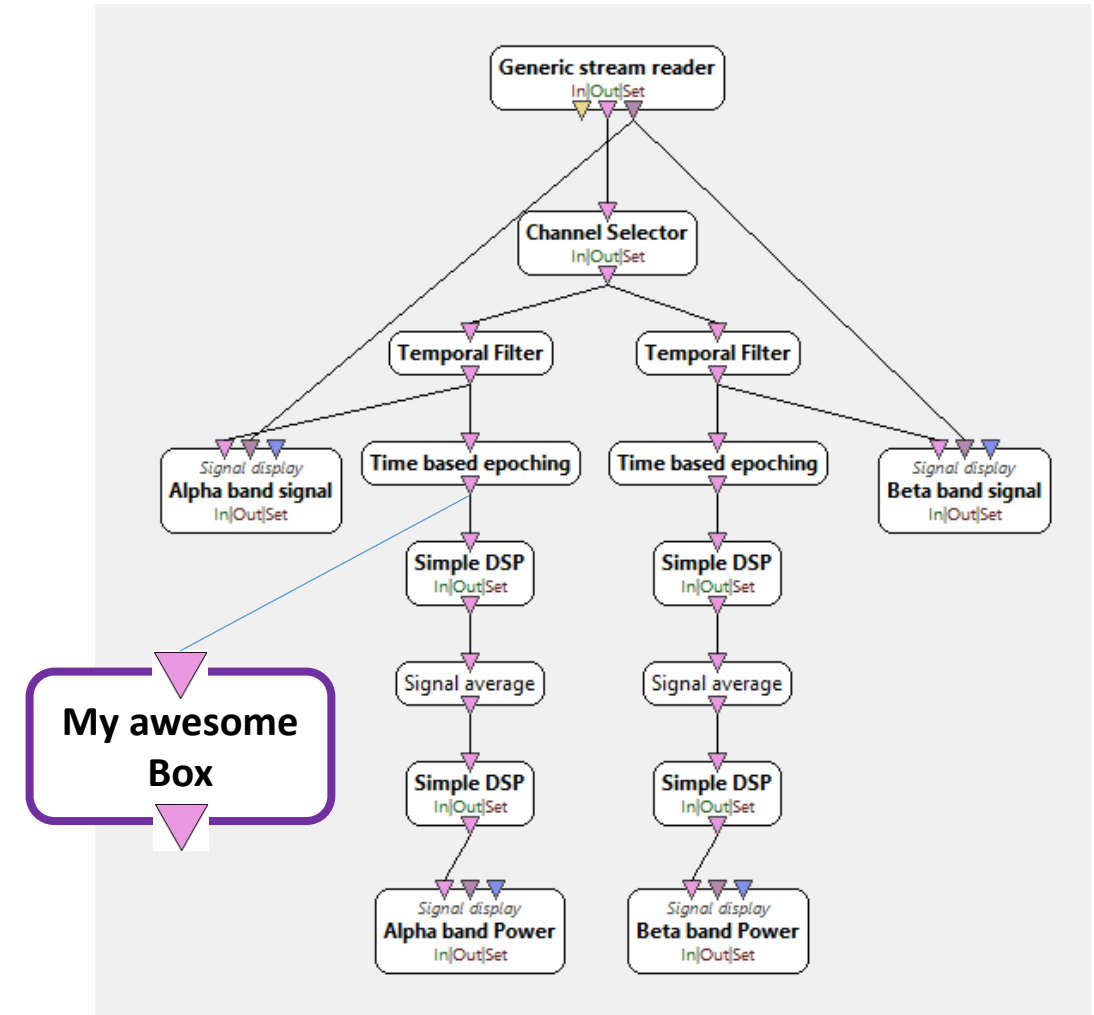
- So, you want to develop a new processing box?





- So, you want to develop a new processing box?
- **First step:**
check if an existing box has what you need...
... or if you can do what you want using a combination of existing boxes.
- **If not - then:**
Do you want a quick & flexible prototype?
→ box calling **Python/MATLAB** scripts

... or a fine-tuned optimized algorithm?
→ **C++ Box & Algorithm** classes



- Using Python/MATLAB scripts in OpenViBE scenarios

Use cases:

- Need for a quick proof-of-concept (e.g. signal processing)
- Don't want/need to code in C++
- Python/MATLAB implementation is already perfect
- Need specific libraries (numpy, scikit-learn...)

→ <http://openvibe.inria.fr/tutorial-using-matlab-with-openvibe/>

→ <http://openvibe.inria.fr/tutorial-using-python-with-openvibe/>

- Great Python tutorial: (courtesy of MENSIA)

→ <http://openvibe.inria.fr/openvibe/wp-content/uploads/2016/06/Quick-prototyping-in-OpenViBE-with-Python.pdf>

- Developing C++ OpenViBE boxes

Use cases:

- Need speed!
- Complete integration with OpenViBE, contribution to the open-source project

→ <http://openvibe.inria.fr/build-instructions/>

- 2016 Tutorial:

→ http://openvibe.inria.fr/openvibe/wp-content/uploads/2016/06/jl_hacking_boxes_2016.pdf

- **Skeleton generator**

Simplest, fastest, go-to solution for beginners...

`openvibe-skeleton-generator.cmd`

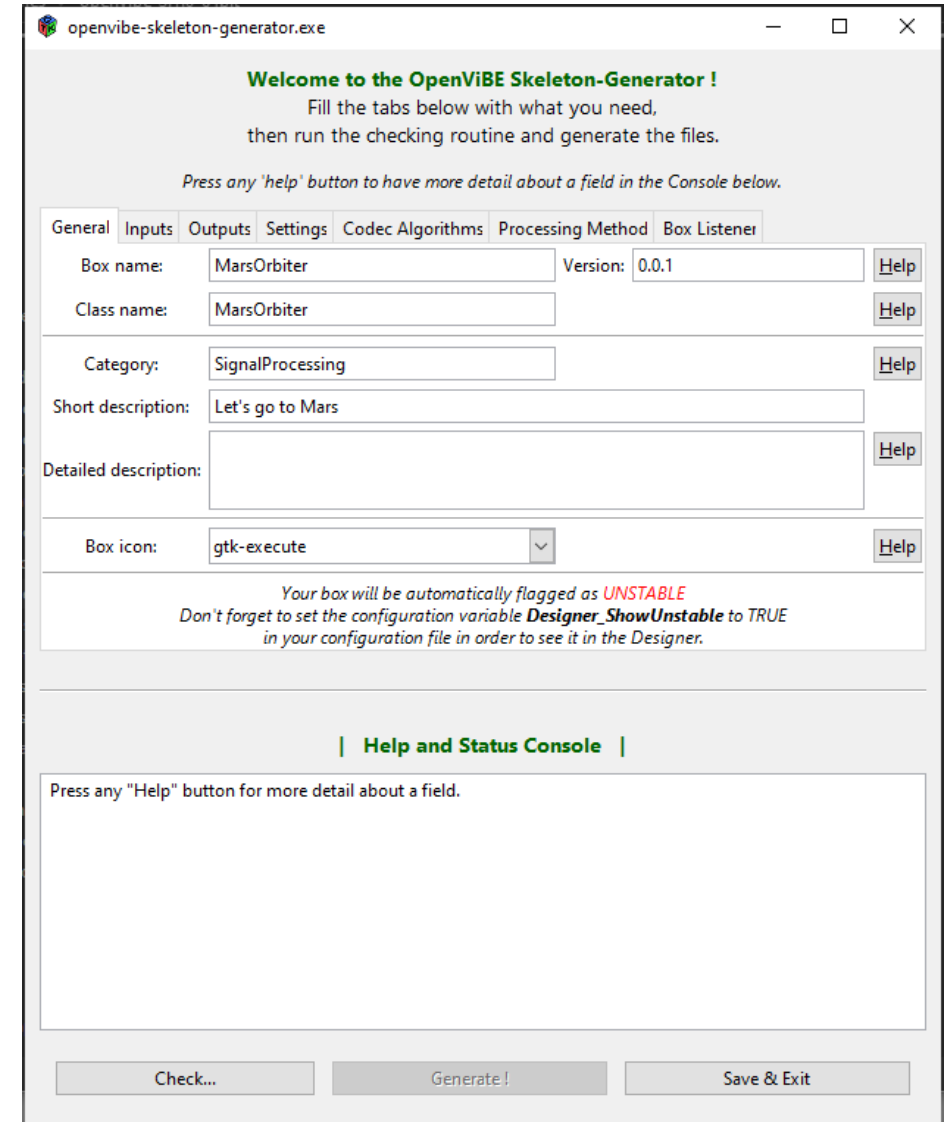
- GUI helping with creating the bare minimum a box needs, with given inputs/outputs, parameters, etc.

➔ All the “OpenViBE glue” is here!

You “just” need to add your specific code.

<http://openvibe.inria.fr/tutorial-1-implementing-a-signal-processing-box/>

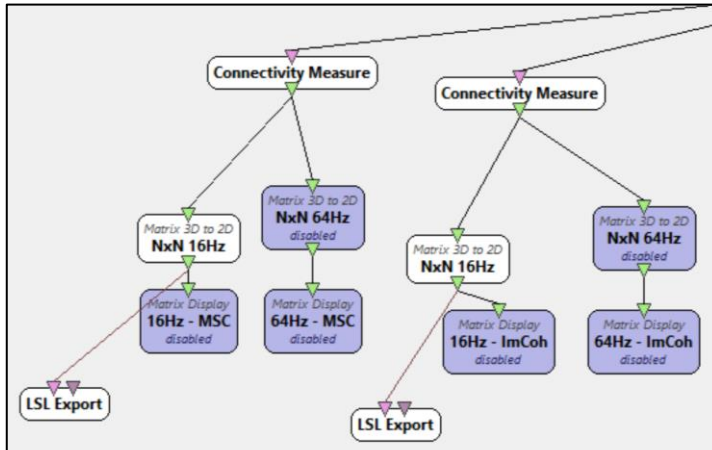
Tip: take inspiration from existing boxes...!



The screenshot shows the 'openvibe-skeleton-generator.exe' window. It has a title bar with standard Windows window controls. The main area is titled 'Welcome to the OpenViBE Skeleton-Generator!' and instructs the user to fill in the tabs below and then run a checking routine. Below this is a tabbed interface with tabs for 'General', 'Inputs', 'Outputs', 'Settings', 'Codec Algorithms', 'Processing Method', and 'Box Listener'. The 'General' tab is active, showing fields for 'Box name' (MarsOrbiter), 'Version' (0.0.1), 'Class name' (MarsOrbiter), 'Category' (SignalProcessing), 'Short description' (Let's go to Mars), 'Detailed description' (empty), and 'Box icon' (gtk-execute). Each field has a 'Help' button next to it. A warning message at the bottom of the form states: 'Your box will be automatically flagged as UNSTABLE. Don't forget to set the configuration variable Designer_ShowUnstable to TRUE in your configuration file in order to see it in the Designer.' Below the form is a section titled 'Help and Status Console' with a text area for messages. At the bottom of the window are three buttons: 'Check...', 'Generate!', and 'Save & Exit'.

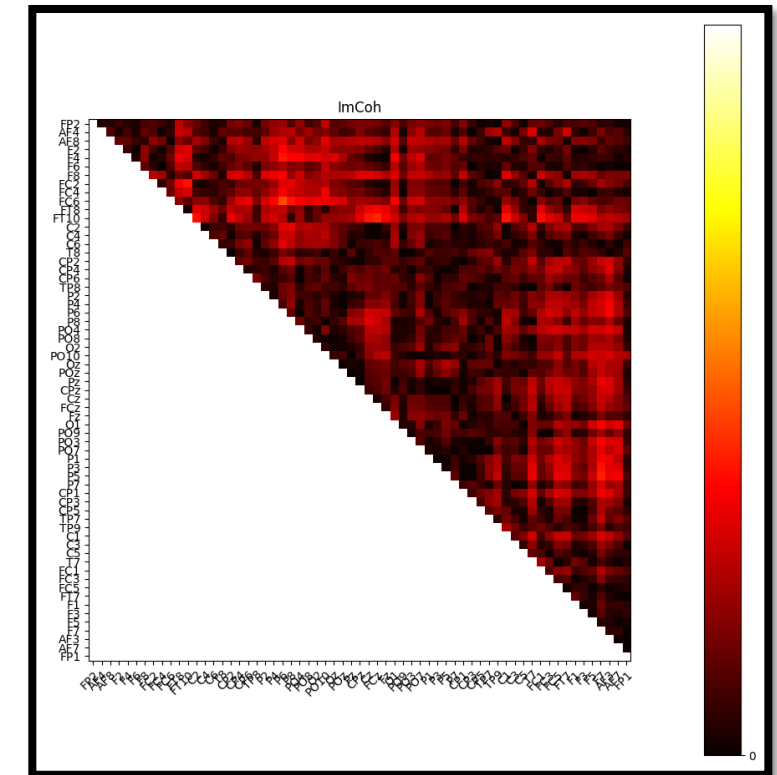
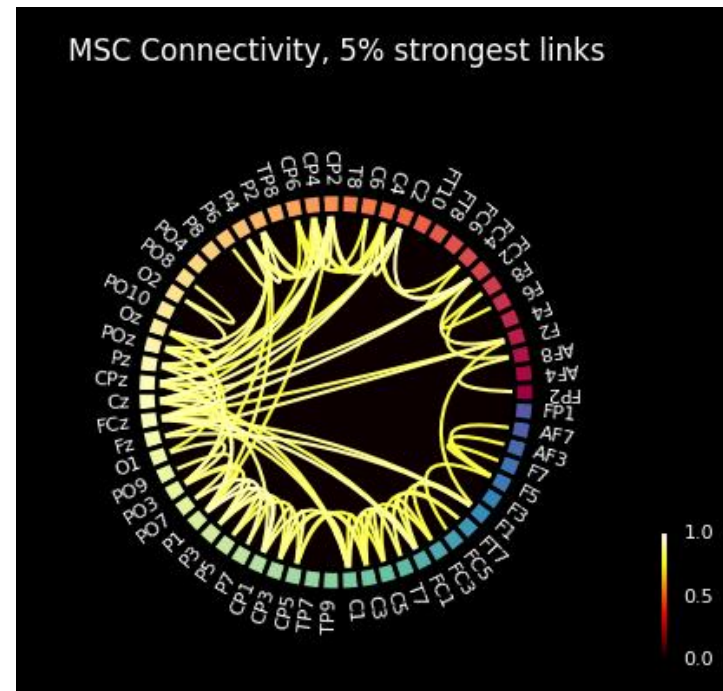
- Examples: interface w/ virtual reality products, video games, external data visualization toolboxes...
- Various available solutions to stream data/events between OpenViBE and external apps.
 - VRPN
 - TCP/IP
 - LSL (Lab Streaming Layer)
 - Python/MATLAB boxes
- Demo using LSL to visualize Connectivity/Adjacency Matrices using an external Python script
- Code and example scenarios:
 - ➔ <https://github.com/AsteroidShrub/openVibe-Lsl-Demo>

Interfacing with external software



- OpenViBE “LSL Export” Box
- Connectivity Measurement Box
- + Python scripting (using pylsl)
 - ➔ external matrices analysis/plotting

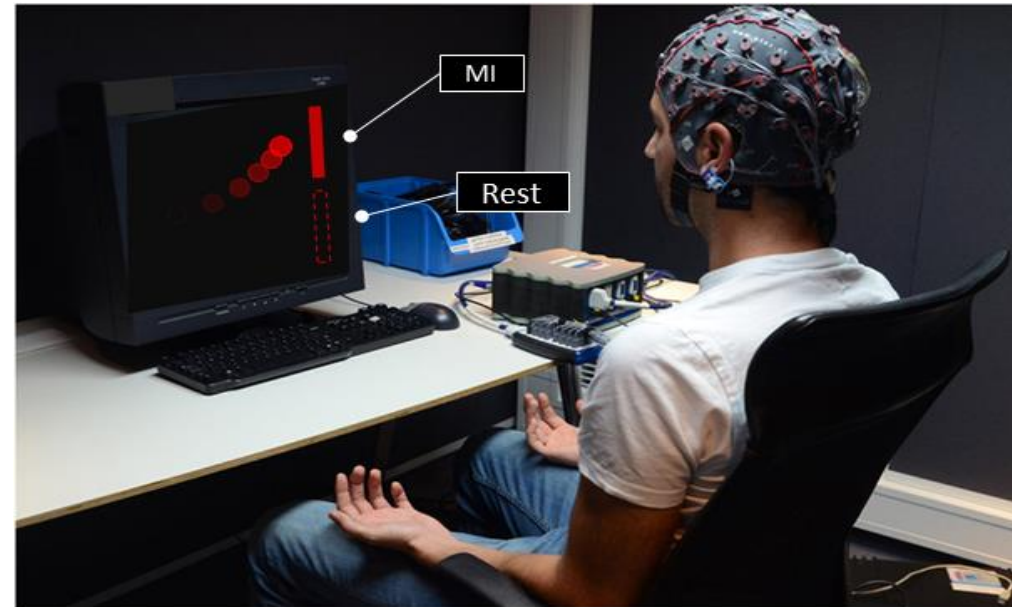
```
1 #!usr/bin/env python
2
3 import numpy as np
4 import numpy.ma as ma
5 import matplotlib
6 matplotlib.use('Qt5Agg')
7 from matplotlib import pyplot as plt
8 from pylsl import StreamInlet, resolve_stream, resolve_byprop
9 from scipy import signal
10 import time
11 import seaborn as sns
12 import mne
13
14 def main():
15
16     plotMatrices = True
17     plotConnectomes = False
18     threshold = 0.33
19     thresholdPercent = 5
20     thresholdType = 'percent' # 'absolute' or 'percent'
21     thresholdedMatrix = 'MSC' # 'ImCoh' or 'MSC'
22
23     nbChannels = 63
24     electrodes = ['FP2', 'AF4', 'AF8', 'F2', 'F4', 'F6', 'F8', 'FC2', 'FC4', 'FC6', 'FT8', 'FT10', 'PO2', 'PO4', 'PO8', 'P2', 'P4', 'P6', 'P8', 'CP2', 'CP4', 'CP6', 'TP2', 'TP4', 'TP6', 'TP8', 'PO10', 'PO12', 'PO14', 'PO16', 'PO18', 'PO20', 'PO22', 'PO24', 'PO26', 'PO28', 'PO30', 'PO32', 'PO34', 'PO36', 'PO38', 'PO40', 'PO42', 'PO44', 'PO46', 'PO48', 'PO50', 'PO52', 'PO54', 'PO56', 'PO58', 'PO60', 'PO62', 'PO64', 'PO66', 'PO68', 'PO70', 'PO72', 'PO74', 'PO76', 'PO78', 'PO80', 'PO82', 'PO84', 'PO86', 'PO88', 'PO90', 'PO92', 'PO94', 'PO96', 'PO98', 'PO100']
25
26     # --- LSL stream
27     print("looking for a stream...")
28     streams = resolve_byprop("type", "signal")
29
30     for strIdx in range(0, len(streams)):
31         print("---STREAM ", strIdx)
32         print("info.type() ", streams[strIdx].type())
33         print("info.name() ", streams[strIdx].name())
34         print("info.nominal_srate() ", streams[strIdx].nominal_srate())
35         print("info.channel_format() ", streams[strIdx].channel_format())
36         if streams[strIdx].name() == "openvibeConnectMSC":
37             inlet_msc = StreamInlet(streams[strIdx])
38             if streams[strIdx].name() == "openvibeConnectImCoh":
```



PART 3 - Concluding remarks & perspectives

3.2 - HappyFeat

- Clinical setting = time constraints
 - Setting up the EEG sensors
 - BCI/MI experiments can be long and strenuous

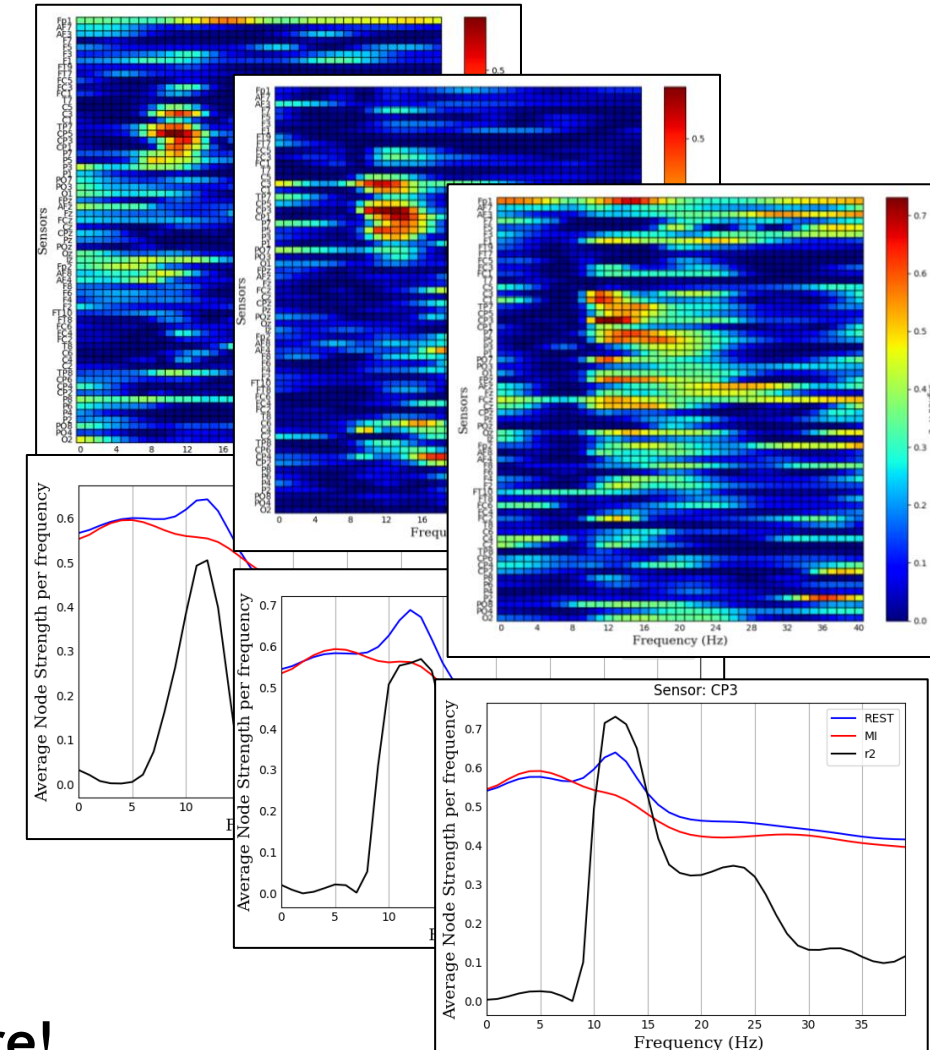


- **Features of interest (FOI)**

- Finding adequate FOIs is a **difficult** and **crucial** step
- After acquisition of MI trials EEG signals, an **analysis phase** is needed to select best FOIs.
 - ➔ Usually involving other scientific softwares (i.e. MATLAB)
- If the analysis phase is too long, a lot can change in the meantime:
 - EEG sensors impedances
 - Subject's brain behaviors
 - Subject's fatigue & motivation drop

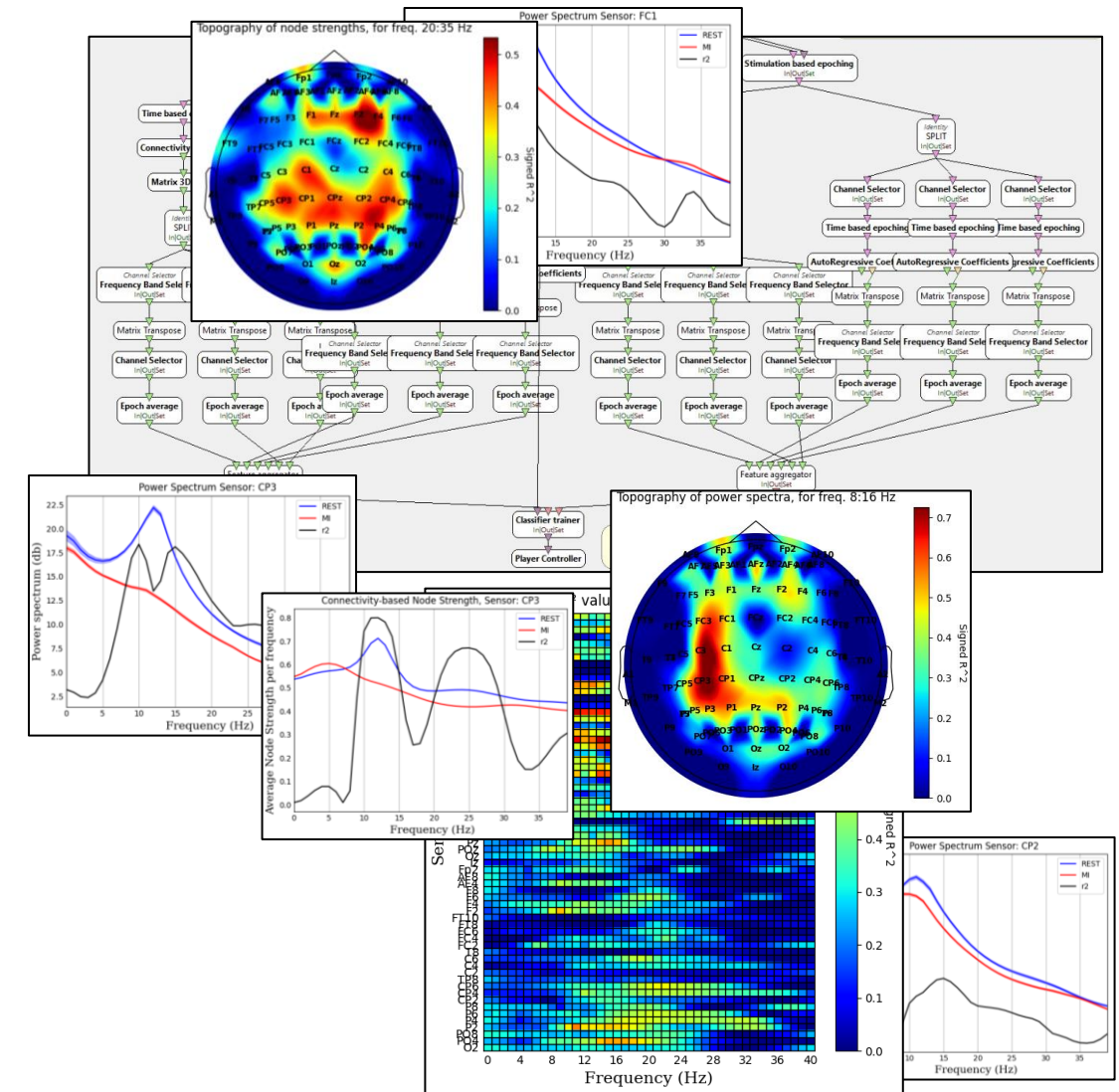
➔ Signal characteristics & Features may differ a lot between Acq/Training phase and Online phase...

➔ **Your trained classifier may not be relevant anymore!**



- **Features of interest (FOI)**

- The “analysis phase” involves a lot of manipulations:
 - Setting up “feature extraction” scenarios in OpenViBE...
 - Selecting FOIs through visualization tools...
 - Setting up & running training scenarios in OpenViBE...
 - ... and maybe going again through those steps multiple times until “correct” features have been selected, or to account for inter-run variability
- ➔ Tedious, error-prone, hard to achieve in a limited time



(“Please send help”, Desbois 2022)

HappyFeat - Main Concepts

Python-based framework for facilitating ML pipelines

Main focus:

making Feature Selection

& Training phases

easy & fast.

Analyze,

select your Features

& train your classifier

in less than 5 minutes!

The screenshot displays the HappyFeat - Feature Selection interface, which is divided into three main functional areas:

- == FEATURE EXTRACTION ==**: This panel contains a list of test files (e.g., Test-[2022.05.05-15.36.18].ov) and a section for extraction parameters. Parameters include Epoch of Interest (EOI) (3), EOI offset (1), Sliding Window (Burg) (0.25), Window Shift (Burg) (0.161), Connectivity Metric (Magnitude Squared Coh.), Length of a connectivity measure (0.25), Overlap btw. connectivity measures (%) (36), Auto-regressive estim. length (s) (0.038), and Frequency resolution (ratio) (1). Below these are experiment parameters (Nb Trials per class: 20, Class / Stimulation 1: MI, Class / Stimulation 2: REST, "Get Set" time (s): 20, Pre-Stimulus time (s): 3, Trial duration (s): 3, Inter-trial interval min (s): 2.5, Inter-trial interval max (s): 3.5, Feedback time (s) (online scenario): 3). At the bottom, there is a button to "Extract Features and Trials".
- == VISUALIZE FEATURES ==**: This panel shows a topography of power spectra for a specific frequency (10:13 Hz). A pop-up window titled "Figure 2" displays a circular brain map with a color scale from 0.0 to 0.7, representing the power spectrum. Below the visualization, there are options to "Load file(s) for analysis" with input fields for Frequency min (0), Frequency max (40), Sensor for PSD visualization (CP3), Topography Freq (Hz) (10:13), and a checkbox for "Scale Colormap (R²map and Topography)" (checked). At the bottom, there are buttons for "Power Spectrum", "Node Strength", "Freq.-chan. R² map", "PSD for the 2 classes", "NodeStr. for the 2 classes", "Brain Topography", and "Brain Topography".
- == CLASSIFIER TRAINING ==**: This panel shows the classifier training process. It includes a section for "Power Spectrum" and "Connectivity" with buttons to "Add feature" and "Remove feature". Below this, there are input fields for "CP3;10" and "CP3;12", and a "Number of k-fold for classification" (10). A list of test files is shown, including "Test-[2022.05.05-16.30.16]-TRIALS.csv". At the bottom, there is a section for "Last Training Results" showing "Test-[2022.05.05-16.30.16]-TRIALS.csv" with "Feature(s) for PowSpectrum: Channel CP3 at 10 Hz" and "Feature(s) for Connectivity: Channel CP3 at 12 Hz". The overall accuracy is 100.0%, and the results for Class 1 and Class 2 are shown. At the bottom, there are buttons for "TRAIN CLASSIFIER" and "FIND BEST COMBINATION".

- **Multiple pipelines available**
 - Available features for classification: Spectral Power, Connectivity-based network metrics
 - ... It's also possible to train the classifier using a fusion of different types of features.
- **Feature extraction**
 - Easy access to experiment & signal processing parameters.
 - Using either **pre-recorded signals** or **live signals, on-the-fly during acquisition phase**.
- **Visual Analysis for feature selection**
 - R^2 maps, PSD comparison across trials, time/freq. ERD/ERS analysis, brain topography...
- **Classifier training**
 - **Run as many training attempts as needed**, using different features, in only a few clicks.
 - Concatenate trials from multiple recorded sessions
 - OpenViBE scenarios are **updated and launched on-the-fly**.
 - Automatically generates/updates the “online classification” scenario.

- **Key feats & mechanisms:**
 - **Clean, risk-free environment**
 - ➔ avoid unnecessary & error-prone manipulations.
 - **Trial-and-error oriented workflow**
 - ➔ all steps can be repeated quickly & as many times as needed
 - **Unified “dashboard” GUI**
 - **OpenViBE** used in the background, as a processing engine.
 - ➔ no scenario edition/manipulation necessary: everything is **automated!**

Full list of dependencies:

- **Python 3.9**
 - shutils>=0.1.0
 - mne>=0.23.0
 - numpy>=1.21.1
 - pandas>=1.3.1
 - PyQt5>=5.15.6
 - statsmodels>=0.13.1
 - scipy>=1.7.1
 - spectrum>=0.8.0
 - matplotlib>=3.4.2
- **OpenViBE v3.4.0**

- Already available online, work-in-progress version:
<https://github.com/Inria-NERV/happyFeat>
- To be continued...
 - More pipeline flexibility
 - More network metrics based on connectivity
 - Associated visualization tools
 - Choose from different classification algorithms
 - Workspace/session manager to save/load session settings
- First official release early 2023 - stay tuned!



BCI Motor Imagery with OpenViBE in X-Men: First Class

Thanks for your attention! Any questions?