BCI workshop at District 3

This document provides instructions and explanations for completing the BCI workshop given by BCI Montréal in May 2015. Specifically, it will guide you through the installation of the necessary software, the configuration of the device we will use, and the two exercises that are at the core of this workshop.

This workshop is intended for people with no or limited experience with Brain-Computer Interfaces (BCIs). The workshop will teach them the basic principles that are necessary to "hack" and develop new applications with BCIs: background on brain activity and brain activity measurement with EEG, structure of a BCI, feature extraction and machine learning. Two hands-on exercises will allow the participants to 1) visualize their EEG signals and some relevant features, and 2) experiment with a very simple BCI design. This should give the participants sufficient knowledge to understand the advantages and limitations of current BCIs, and to devise their own applications.

Programming languages for the workshop exercises

The material for this workshop is provided in two flavors: **Python scripts** and **MATLAB / Octave scripts**.

- **Python***: a popular, multi-purpose powerful, free, open and simple to read scripting language.
- **MATLAB**: very popular in academia, technical programing-oriented, license required, not open.
- **GNU Octave**: high-level interpreted language, primarily intended for numerical computations, quite similar to MATLAB. Moreover, it is free software.

Use the one that is more convenient for you.

* In this workshop, the Python scripts are compatible with Python 2 and Python 3.

Additional software for the workshop

Additional software is required the workshop, specifically, the following tools:

- **Muse SDK**: the software development kit provided to play around with the **Muse*** EEG headband.
- **MuLES**: an EEG server that allows device-agnostic applications. **

* The Muse model utilized for this workshop is the 2014. Unfortunately the newer version (**Muse 2016**) is not supported for the moment in **MuLES**. If your device has 2 micro-USB ports, it's the 2014 model.

** The scripts for the workshop will perfectly work for other EEG headsets supported by *MuLES* for example: EMOTIV EPOC+, Neurosky MindWave, Neuroelectrics ENOBIO and OpenBCI V3. See MuLES documentation for further information.

A Installation of software for the workshop

There are many other programming languages (C, C++, Java, Processing, etc.); a diversity of **BCI toolboxes** (OpenVIBE, BCI2000, BCILAB, etc.); and even **other EEG devices** (OpenBCI, Emotiv EPOC, Neurosky Mindwave, etc.).

Among those, we chose the **Python(or MATLAB/Octave)-Muse-MuLES** combination as it provides a lot of flexibility to hackers, but at the same time is simple enough that novice users can understand what they are doing. Because of this choice, we are stuck with Windows OS; however, the goal of this workshop is to teach you about BCIs in general, so that you are able to apply this knowledge to the environment and tools of your choice. We won't focus much on tools here.

These are the steps to setup your computer

A.1. Installing Python and required packages
A.2. Installing Muse SDK

A.3. Installing MuLES (MuSAE Lab EEG Server)
A.4. Pairing the Muse EEG headset and configuring MuLES
A.5. Download the code for the workshop

If you will use Octave, download the newest **w64** installer (`octave-4.2.1-w64-installer.exe` in March 2017) from https://ftp.gnu.org/gnu/octave/windows/, and execute. Then proceed to step **A.2**

If you will use MATLAB, and it's already installed proceed to step **A.2**

A.1 Installing Python required packages

Python is a high-level scripting language that has been widely adopted in a plethora of applications. It is open, free, simple to read, and has an extensive standard library. Many packages can also be downloaded online to complement its features.

Two packages are especially useful when dealing with scientific computing (as for BCIs): NumPy and matplotlib. **NumPy** allows easy manipulation of arrays and matrices (very similar to MATLAB), which is necessary when dealing with data such as neurophysiological signals. **Matplotlib** is similar to MATLAB's plotting functionalities, and can be used to visualize the signals and features we are working with.

Other packages we will use in this workshop are:

- scikit-learn: a machine learning library.
- pyZMQ: the Python binding for ZMQ, a simple communication library.

To install Python 2 or Python 3, and the required packages, we suggest you download and install the Anaconda distribution. Anaconda is a Python distribution that includes Python 2.7 (in case of Anaconda 2) or Python 3.5 (in case of Anaconda 3), all the packages we will need for the workshop (as well as plenty other useful packages), and Spyder, a great IDE for scientific computing in Python.

(1) Installation of Python with Anaconda (recommended)

1. Download the Anaconda graphical installer (if your Windows version is 32-bit, make sure to download the 32-bit installer).
2. Execute the installer.

This installs Python, Spyder and all the packages we will need for the workshop (including scikit-learn and pyZMQ).

(2) Individual installation of Python and packages

Alternatively, you can download Python 2.7 independently. Make sure to install `pip` (as explained here)) and grab **NumPy**, **matplotlib** and **scikit-learn** by calling `pip install <package_name>` on the command line (or any other way you prefer). Make sure you have a text editor or IDE you can work with as well.

A.2. Muse SDK

This workshop is based on the Muse EEG headband. The Muse provides 4 EEG dry sensors, 2 located on the forehead and 2 behind the ears. It communicates via Bluetooth to a computer or a mobile device. An SDK is available to allow basic control over the Muse's acquisition and record capabilities.

To install the SDK, download and execute the Muse SDK Tools 3.4.1 installer for Windows.

A.3. MuLES (MuSAE Lab EEG Server)

MuLES is an EEG streaming server, i.e. a piece of software that handles the communication with an EEG device and streams the recorded signals on a network. This is very useful in our case since:

1. It provides a high-level interface to EEG signals.

2. It can communicate with any of the following devices without requiring any modification: **Muse**, **Emotiv EPOC**, **Neurosky Mindwave**, **OpenBCI**, **Neuroelectrics Enobio**.
3. It can stream EEG signals over a single computer or a network (or the Internet).

MuLES is developed in LabVIEW, and is currently only available for Windows. Future releases might support other operating systems.

To install MuLES, download the MuLES installer v1.3 and follow the instructions given on the web page.

A.4. Pairing the Muse EEG headset and configuring MuLES

The **Muse** communicates to external devices using the Bluetooth protocol, and thus needs to be paired with your computer. To pair the **Muse** with your computer, follow these steps:

1. Switch on the **Muse** into Pairing Mode by holding the button down for ~5 seconds. The light should start flashing.
2. Under *Control Panel/Hardware and Sound*, click on *Add a device*.
3. A Bluetooth device named *Muse<-something>* should appear. Select the Muse device and click *Next*.
4. Click *Next* when asking for a passcode.
5. Turn off the Muse by holding the button down for ~2 seconds.

Now that your computer is set up to recognize the Muse, we will configure MuLES so it knows which device it should be looking for.

1. Go to the MuLES installation directory (by default `C:\Program Files (x86)\MuSAE_Lab\MuLES`).
2. Open `config.ini` in a text editor.
3. Under the section [DEVICE03], locate the `EXTRA` keyword.
4. Change the value of the `BTNAME` parameter to the name of your Muse. For example, if your Muse's Bluetooth name is `Muse-6AB1`, you would have the line `EXTRA = "FS=220,#CH=4,DATA=ffffi,PRESET=14,BTNAME=Muse-6AB1,OSCPORT=5000"`.
5. If you are working in a 32-bit version of Windows, you will need to change the value of the `PATH` keyword to indicate the correct path to `muse-io.exe` (should be similar to `"/C/Program Files (x86)/Muse/muse-io.exe"`).
6. Save and close the file.

A.5. Download the code for the workshop

The code for the workshop consists of Python (and MATLAB / Octave) scripts that you can find here.

You can download everything as a `.zip` file using the button download zip on the right. You then need to unzip the folder on your computer.

Alternatively, if you have `git` installed on your computer, you can clone the repository by calling `git clone https://github.com/NeuroTechX/bci-workshop.git` in the command line.

Exercise 02 uses the Support Vector Machine (SVM) algorithm to perform classification. Pre-build binaries of the LIBSVM are provided for 64-bit versions of MATLAB and Octave.

Exercise 1: A simple neurofeedback interface

In this first exercise, we will learn how to visualize the raw EEG signals that come from the Muse inside a Python application. We will also extract and visualize basic features of the raw EEG signals. This will showcase a first basic use of a Brain-Computer Interface: a so-called neurofeedback interface.

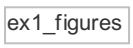
> Neurofeedback (also called neurotherapy or neurobiofeedback) uses real-time representation of brain-activity (as sound and visual effects) to teach users to control their brain activity.

E1.1 Streaming data from the Muse

1. Open MuLES (a shortcut should have been automatically added to your desktop, otherwise you can run it from the installation directory `C:\Program Files (x86)\MuSAE_Lab\MuLES`).
2. In the dropdown menu, choose `MUSE Consumer FW`.
3. Make sure the *Enable TCP Server* button is highlighted in green. You can also save the EEG data in a file by enabling the *Enable Logging* button.
4. Switch on the Muse by holding the button down for ~1 second. The light should start oscillating.
5. Start streaming data from the Muse by clicking on the green *Play* button. A command line window should appear giving details on the Muse communication status.

E1.2 Running Exercise 1 script

1. Open the script `exercise_01.py` in **Spyder** or `exercise_01.m` in **MATLAB** or **Octave**.
2. Read the code - it is thoroughly commented - and modify the experiment parameters as you wish in the **Set the experiment parameters section**.
3. Run the script. In **Spyder**, select a Python console on the bottom right of the screen, then click on the *Run File* button on top of the editor.
4. Two figures should appear: One displaying the raw signals of the Muse's 4 EEG sensors, and another one showing the basic band power features we are computing on one of the EEG signals.
5. To stop the execution of the script, in **Python** press `Ctrl+C` in the Python console. In **MATLAB** and **Octave** close any of the figure windows. Don't forget MuLES is still running as well!



E1.3 Playing around

Here are some things we suggest you do to understand what the script does.

Visualizing your raw EEG signals

Run the script and look at the first figure (raw EEG visualization). What makes your signal change?

1. Try blinking, clenching your jaw, moving your head, etc.
2. Imagine repeatedly throwing a ball for a few seconds, then imagine talking to a good friend for another few seconds, while minimizing any movement or blinks.

In the first case, you can see that the first movements (blinking, etc.) strongly disturb the EEG signal. We call these **artifacts**. Some artifacts are so huge that they can completely obscure the actual EEG signal coming from your brain. We typically divide artifacts according to their source: *physiological artifacts* (caused by the electrical activity of the heart, muscles, movement of the eyes, etc.), and *motion artifacts* (caused by a relative displacement of the sensor and the skin).

In the second case, you can see that different mental activities (e.g. imagining eating or talking) are not easily recognizable in the EEG signals.
First, this is because mental activity is distributed across the brain: for example, sensorimotor processing occurs on top of the brain, in the central cortex, while speech-related functions occurs at the sides of the brain, in the temporal cortex. Therefore, the 4 sensors on the Muse are not necessarily on the right "spot" to capture those EEG signals.
Second, the EEG signals are very, very, very noisy. Indeed, the electrical signals that we pick up on the scalp are smeared by the skull, muscles and skin. As you saw, eye balling the signals is often not enough to analyze brain activity. (That is why we need to extract descriptive characteristics from the signals: what we call features!)

Visualizing your EEG features

Since the raw EEG signals are not easy to read, we will extract **features** that will hopefully be more insightful. Features are simply a different representation, or an individual measurable property, of the EEG signal. Good features give clearer information about a phenomenon being observed.

The most often used features to describe EEG are frequency band powers.

In **Python**

1. Open the script `bci_workshop_tools.py`.
2. Locate the function `compute_feature_vector()`.

In **MATLAB** or **Octave**

1. Open the script `compute_feature_vector.m`, located in the `bci_workshop_tools` folder.

This function uses the Fast Fourier Transform, an algorithm that extracts the frequency information of a time signal. It transforms the EEG time series (i.e., the raw EEG signal that you visualized above) into a list of amplitudes each corresponding to a specific frequency.

In EEG analysis, we typically look at ranges of frequencies, that we call *frequency bands*:

- Delta (< 4 Hz)
- Theta (4-7 Hz)
- Alpha (8-15 Hz)
- Beta (16-31 Hz)
- Gamma (> 31 Hz)

These are the features that you visualized in E1.2 in Figure 2.

We expect each band to reflect specific mental activities. For example, we know that closing the eyes and relaxing provokes an increase in Alpha band activity and a decrease in Beta band activity, especially at the back of the head. We will try to reproduce this result now.

1. Open the script `exercise_01_one_channel.py` or `exercise_01_one_channel.m`.
2. Change the value of the `eeg_buffer_secs` parameter to around 40.
3. Run the script and look at the second figure.
4. Keep your eyes open for 20 seconds (again, try to minimize your movements).
5. Close your eyes and relax for another 20 seconds (minimize your movements).

Do you see a difference between the first and the last 20-s windows for the Alpha and Beta features?

Advanced: computing supplementary features

Many other features can be used to describe the EEG activity: band powers, Auto-Regressive model coefficients, Wavelet coefficients, coherence, mutual information, etc. As a starting point, adapt the code in the `compute_feature_vector()` function to compute additional, finer bands, i.e. low Alpha, medium Alpha, high Alpha, low Beta, and high Beta:

- Low Alpha (8-10 Hz)
- Medium Alpha (9-11 Hz)
- High Alpha (10-12 Hz)
- Low Beta (12-21 Hz)
- High Beta (21-30 Hz)

These bands provide more specific information on the EEG activity, and can be more insightful than standard band powers. Additionally, adapt the code to compute ratios of band powers. For example, Theta/Beta and Alpha/Beta ratios are often used to study EEG.

Repeat the above eyes open/closed procedure with your new features. Can you see a clearer difference between the two mental states?

Other points you can consider to design better features:

- Extract the features for each of the Muse's 4 sensors. Each sensor measures a different part of the brain, and so features from different sensors can again provide more specific information.
- Similarly, ratios of features between different sensors (e.g. Alpha in frontal region/Beta in temporal region) can provide additional useful information.

Exercise 2: A basic BCI

In this second exercise, we will learn how to use an automatic algorithm to recognize somebody's mental states from their EEG. We will use a *classifier*: a classifier is an algorithm that, provided some data, learns to recognize patterns, and can then classify similar unseen information.

For example, let's say we have many images of either cats or dogs that we want to classify. A classifier would first require *training data*: in this case we could give the classifier 1000 pictures of cats that we identify as cats, and 1000 pictures of dogs that we identify as dogs. The classifier will then learn specific patterns to discriminate a dog from a cat. Once it is trained, the classifier can now output *decisions*: if we give it a new, unseen picture, it will try its best to correctly determine whether it's a cat or a dog in the picture.

In a Brain-Computer Interface, we use classifiers to identify which type of mental task somebody is doing. For example, in the previous exercise, you saw that opening and closing your eyes modifies the features of the EEG signal. To use a classifier, we would need to proceed like this:

1. Collect EEG data of you performing the two mental activities (eyes open and eyes closed).
2. Input this data to the classifier, while specifying which part corresponds to each mental activity.
3. *Train* the classifier.
4. Use the trained classifier by giving it new EEG data, and asking for a decision on which mental activity this represents.

Brain-Computer Interfaces rely heavily on **machine learning**, the field devoted to building classifiers (and other cool stuff). You might have already understood why: in a typical EEG application we have several features (e.g. many band powers) and the mental activity we want to classify is not easily recognizable.

Let's try it now.

E2.1 Running the Python basic BCI script

1. Open the script `exercise_02.py` or `exercise_02.m`
2. Read the code - it is thoroughly commented - and modify the experiment parameters as you wish in the **Set the experiment parameters section**. You will see it's very similar to the code of **Exercise 1**, but with a few new sections.
3. When you feel confident about what the code does, run the script.
4. When you hear a **first beep**; keep your eyes open and concentrate (while minimizing your movements).
5. When you hear a **second beep**; close your eyes and relax (while minimizing your movements).
6. When you hear a **third beep**, the classifier is trained and starts outputting decisions in the Python console: `0` when your eyes are open, and `1` when you close them. Additionally, a figure will display the decisions over a period of 30 seconds.
7. To stop the execution of the script, in **Python** press `Ctrl+C` in the Python console. In **MATLAB** and **Octave** close the classification prediction figure, to terminate the script. Don't forget MuLES is still running as well!

ex1_figures

E2.2 Playing around

Here are some things we suggest you do to understand what the script does.

Visualizing the classifier decisions

Try the above procedure to train and use the classifier. If it does not work well, make sure the EEG signals are stable (you can

reuse the code of Exercise 1 to visualize the raw signals) and try again. Some people will typically have a stronger Alpha response than others, and it also takes practice to be able to modulate it at will.

Using other mental tasks

Try the same procedure again, but train your classifier with different mental activities. For example, perform some random mental mutliplication during the first 20 seconds, and try to come up with as many words as possible starting with the letter *T* during the next 20 seconds. Once the classifier is trained, repeat the two mental tasks. Is the classifier able to recognize the task you are performing?

Additionally, try the first training procedure again (eyes open vs. eyes closed). However, don't close your eyes during the second task, but instead relax with your eyes open. Can the classifier recognize your mental state?

Sending decisions to an external application

Once your BCI framework is functional, you can start thinking about sending your EEG features or classifier decisions to an external application.
Many different libraries can be used for that, beginning with standard TCP/IP communication implementations (e.g. Python's socket module).
We suggest pyZMQ, which allows simple communication between a Python application and any programming language supporting the ZMQ protocol.

For example, you could send the classifier's decisions to a Processing script to create simple animations based on your mental activity.

Conclusion

In this workshop, we saw **1)** how to run a simple neurofeedback interface, and **2)** use a basic brain-computer interface. To do so, we covered the basic principles behind the use of electroencephalography signals in modern BCI applications: properties of the raw EEG time series, extraction of band power features, physiological and motion artifacts, and machine learning-based classification of mental activities.

We used the following tools in this workshop: the **MuLES EEG server**, the **Python**, **MATLAB** and **Octave** scripting languages, and the **Muse EEG headset**. All the necessary scripts for this workshop are available online and their re-use is strongly encouraged.

Now is **your turn** to come up with inventive ways of using neurophysiological data! You can follow the pointers in the *References* section for inspiration.

References

Tutorials and neurohacks

- A blog with very cool and detailed posts about EEG/BCI hacking: http://eeghacker.blogspot.ca/
- The neuralDrift, a neurogame based in MATLAB that exploits the same concept as was seen in this workshop: https://github.com/hubertjb/neuraldrift
- Series of introductory lectures on Brain-Computer Interfacing: http://sccn.ucsd.edu/wiki/Introduction_To_Modern_Brain-Computer_Interface_Design
- A visualizer in Unity
- Using spotify and the Muse
- Other NeurotechX resource

Authors

Hubert Banville & Raymundo Cassani

Thanks to the MuSAE Lab, District 3 and NeuroTechMTL. Thanks also to Ana, Sydney, Rohit and William for initial feedback on the workshop.

Also, please cite the following paper if you use MuLES for academic purposes:

Cassani, R., Banville, H., & Falk, T. H. (2015). MuLES: An Open Source EEG Acquisition and Streaming Server for Quick and Simple Prototyping and Recording. In Proceedings of the 20th International Conference on Intelligent User Interfaces Companion (pp. 9-12). ACM.