Lab 5

Overview

The laboratory aims to establish a real-time data streaming connection between an OpenBCI headset and Python using the BrainFlow library. This setup will facilitate the continuous transmission of neurophysiological data from the headset to Python, enabling online analysis.
The results of this lab will serve as the foundation for the subsequent lab, where the harnessed data will be utilized to design and create a Brain-Computer Interface.

Deliverables

This lab is worth 1% of your overall grade. The grade will be assessed based on in-lab components. There is no report for this lab. The preparation of this lab is not graded, but should be completed *before* the lab session.

You **must** attend the session for the entire duration of the lab, unless you finish early and your TA confirms that you have completed the required deliverables and may leave.

Your grade will be assessed based on the following requirements:

- Demonstrating completion of each section to your TA
- Answering TA evaluation questions during the session

Streaming data from OpenBCI to Python

Depending on the application, you will realize that rather than using preexisting toolboxes, or GUIs, it's better to tailor code from scratch. In this case, we will be working on streaming data directly from the Cyton board to Python, without the necessity of the OpenBCI GUI.

Packages

For this laboratory you will need to import the following packages:

- time
- brainflow
- numpy

Specifically from BrainFlow, it is easier if you only import the following modules:
- BoardShim
- BrainFlowInputParams
- BoardIds

And from the brainflow.data_filter module, import the following submodules:
- DataFilter
- WindowOperations
- DetrendOperations

Main function

1. Create a function called "**calculate_alpha_beta_ratio**"; it should take one optional argument as an input.
    a. Name the argument "**port**" and set its default value to a string containing the name of the port that will be employed (eg. **port = 'COM4'**)
    b. All the following instructions should be coded inside of this function, unless stated otherwise.
2. Initialize the board information
    a. Begin by creating a variable where the type of board (boardID) will be assigned (eg. *[variable name] = BoardIds.CYTON_BOARD*)
    b. Create another variable where the board description (boardDescr) will be stored. The function **get_board_descr()** from the **BoardShim** submodule will be necessary. The only argument required for this function is the variable where you stored the boardID. (eg. *[board description variable] = BoardShim.get_board_descr([board ID variable]))*
    c. Store the sampling rate of the board (sampleRate) in a new variable.
        i. You can extract this information from the dictionary you created in step 2.b, just type **['sampling_rate']** after the variable where you stored your board description.
        ii. Convert this variable into an integer for further use.
3. Set the board parameters
    a. Create an instance of **BrainFlowInputParams()** and store it in a variable of your choosing. No arguments are necessary.
    b. Set the **serial_port** attribute of the instance you just created to be the same as the argument of the function you created in 1.a (eg. *[name_of_parameters_instance].serial_port = port*)
4. Create the board object.
    a. Create an instance of a board object by calling the **BoardShim()** function, the necessary arguments are the boardID (created in 2.a) and boardParameters (created in 3.a), in that specific order. Store this board object in a variable of your choosing.
    (eg. *[board name] = BoardShim([boardID object], [boardParameters object])*)

5. Prepare the board session and start the stream.
    a. Call the **prepare_session()** method on the board object. (eg. *[board object name].prepare_session()*). This prepares the board for data streaming.
    b. Call the **start_stream()** method on the board object to initiate the streaming process (eg. *[board object name].start_stream()*)
6. Calculate the nearest power of two for spectral power estimation.
    a. Start by calculating the nearest power of two to the sampling rate (Should be stored in the variable created in 2.c) and store it in a variable of your choosing.
        i. There's several ways to do this calculation. The recommended method is the **get_nearest_power_of_two()** function of the **DataFilter** module; the only necessary argument is the sampling rate. (eg. *[nearest power of 2] = DataFilter.get_nearest_power_of_two([sampling rate])*)
7. Acquire data.
    a. Using the sleep() function of the **time** module, pause the execution of the code for 2 seconds. This pause allows time for data to accumulate before processing.
    b. Retrieve data from the OpenBCI board using **board.get_board_data()** and store it in a variable.
8. Extract the EEG channel information.
    a. Using the **['eeg_channels']** key, extract information of which items are EEG channels from the boardDescr dictionary created in 2.b and store it in a variable. (eg. *[eeg channels variable] = [board description dictionary]['eeg_channels']*)
9. Initialize the alpha-beta ratio array.
    a. Using the np.zeros() function, create an array with size equal to the number of eeg channels.
10. Since the filtering functions employed by BrainFlow can only operate on individual time series at a time, analysis on individual channels will be necessary. Loop through the EEG channels using the enumerate() function on the variable you created on 8.a. This will allow you to access individual channel's data as well as index them by number. (eg. *for count, channel in enumerate(eeg_channels):*)
    a. Inside of this loop, initialize a DataFilter object with the detrend() method. It will require two input arguments, the first should be the data of an individual channel, and the second must be the initialization value for the detrending. **DetrendOperations.LINEAR.value** must be used. (eg. *DataFilter.detrend(data[channel], DetrendOperations.LINEAR.value)*)
    b. Calculate the power spectrum using the **get_psd_welch()** function. Parameters include data from a single channel, the length of the signal to be analyzed (stored in the variable created in 6.a.i), overlap (half the length of the signal), sampling rate and the window type (in this case, a Hanning window). Store this value in a variable. (eg. *[power spectrum] = DataFilter.get_psd_welch(data[channel], [signal length], [signal length] // 2, [sampling rate], WindowOperations.HANNING.value)*)

c. Now that the power spectrum has been calculated, use the **get_band_power()** method from the DataFilter object to extract the power of the alpha and beta bands. The required parameters are the power spectrum (calculated in the previous step), the lower frequency of the band, and the higher frequency.
    i. For the alpha band, use frequencies 7 & 13Hz
    (eg. *[alpha power] = DataFilter.get_band_power([power spectrum], 7.0, 13.0)*)
    ii. For the beta band, use frequencies 14 & 30Hz
    (eg. *[beta power] = DataFilter.get_band_power([power spectrum], 14.0, 30.0)*)
d. Calculate the ratio between the two band powers and store it in the alpha-beta array created in step 9. (eg. *[alpha-beta ratio][index] = [beta power] / [alpha power]*)

11. Check the status of the session using the **is_prepared()** method from the board object (created in step 4.a), and stop streaming using the **release_session()** method.
    (eg. *if [board object].is_prepared(): [board object].release_session()*)
12. Return the mean value of the alpha-beta array as the output of the function.
    (eg. *return np.mean([alpha-beta ratio])*)

Running as a standalone script

Since this code will be employed as a standalone during this laboratory, but as a module on the next, it will be necessary to differentiate how the script runs in each case. As a stand alone, it should constantly update the value of the alpha-beta ratio and display it on the console until it is interrupted; as a module, it should only return the alpha-beta ratio once every time it is called, without any console output.

To do so, the next adjustments to the code should be made:

1. Create a new section of the code using the following statement:
   if __name__ == "__main__":
2. Under a try statement, create an infinite loop that calls the main function of this script and prints the returned value on the console.
   Eg. *try:*
   while True:
   [ratio] = [main function name]()
   print("alpha/beta ratio:", [ratio])
3. Under a except clause, use the KeyboardInterrupt exception to make the script stop running.
   Eg. *except KeyboardInterrupt:*
   quit()

Testing

1. Turn on the OpenBCI headset.
2. Insert the dongle to a USB port of your choosing.

3. Update the default setting of the argument “**port**” to the port you are currently connected to.
    a. You can check to which port you are connected to by opening Device Manager (Start → Control Panel → Hardware and Sound → Device Manager)

       Look in the Device Manager list, open the category "Ports", and find the matching COM Port.
4. Save your script.
5. Call it through the console (*python [script name].py*)
    a. Troubleshoot if necessary.
    b. If everything is successful, your code should continuously print the average ratio of beta/alpha every 2 seconds.

References

Parfenov, A. (2018). *User Api¶*. User API - BrainFlow documentation. https://brainflow.readthedocs.io/en/stable/UserAPI.html