Hybrid BCI documentation

Jonas Braun, jonas.braun@tum.de

Experimental Paradigm: *run_session_livehybrid_180329_2.py*

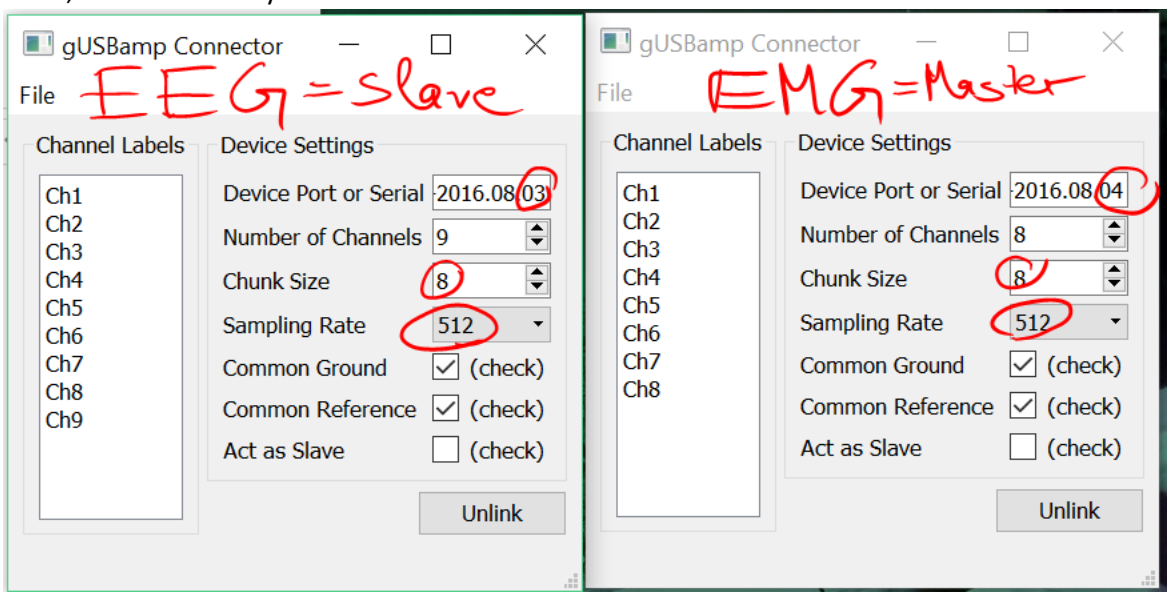
The class hybridBCI implements an experimental paradigm that can be used for EMG, EEG, or hybrid BCI experiments, all in either offline mode or online mode. The operation mode is specified with the command line inputs.

Before starting an experiment

- Make sure Python version 3.6 is installed. On the NST recording PC, start the anaconda prompt with the environment py36 (run the following command: activate py36)
- Make sure all the software is installed and all changes to gumpy as described below are included in your gumpy distribution. If not, change the source, e.g. on the recording PC under: C:\Miniconda2\envs\py36\Lib\site-packages\gumpy-0.5.0-py3.6.egg
- On the NST recording PC navigate to : C:\Users\nst\Desktop\Jonas_HybridBCI\02_realtime

How to run an experiment:

- Connect both g.USBamps with the synchronisation cable (EMG sync out and EEG sync in)
- Connect both g.USBamps to the computer with the USB cables
- **Start the g.USBamp connector** app (gUSBamp.exe) that was delivered with the lab streaming layer (LSL) distribution **twice**
- Select "File" → "Load Configuration" and **load** "g_USB_config_EMG.cfg" and "g_USB_config_EEG_hybrid.cfg" in both of the windows. Both sampling frequencies have to be the same, otherwise the synchronisation will not work.



- First "**Link**" the one with the EEG config (2016.08.03), then "**Link**" the one with the EMG config (2016.08.04). If done the other way around it does not work and will produce a "No Recording Error". If it does not work, the "Unlink" both amplifiers (If the button does not toggle immediately unplug the USB cable of the respective amplifier) and link them the other way around than before.
- Start the **anaconda prompt** and **cd** to the folder where run_session_livehybrid_180329_2.py is located
- Run the command as described below
- If robotic arm and/or hand are connected then they will be used otherwise the output is just to the command line

Command line options

Example: *python run_session_livehybrid_180329_2.py -t 12 -l 9 -f 512 -m HYBRID -p 0 -a 22 -g male*

- -m: mode of experiment: either *EEG* or *EMG* or *HYBRID*
- -f: sampling frequency: Has to be the same as selected in the g.USBamp connector
- -t: total number of trials: has to be a multiple of 3
- -l: number of offline trials before training: Has to be a multiple of 3 and smaller than -t
- -p: force level decoding on or off: 0 for off and 1 for on. The first half of the offline trials will be labelled as high force, the second half as low force, hence the subject should be instructed to do so. Same for the online trials.
- -a: age of the subject
- -g: gender of the subject
- -w: feedback: not required

First the offline trials will be performed, then some training happens, then the online trials are performed.

To use only offline trials, select -t equal to -l. If using mode EMG or HYBRID, -l must at least be 9, because of the threefold cross-validation in the model calculation

Script details

Everything is implemented in the **class hybridBCI**.

The script is implemented as a state machine, that runs in the method *run_trial()*, which has been added as a process to the task manager of Panda. It is called every 10 seconds. See comments in the code for details of the individual states and the transitions. Within the states the methods run_notlive_EMG(), run_live_EMG(), show_motor_imagery(), classify_notlive() and classify_live() are called.

For the actual recording an instance of the classes **RecordData_liveEEG_JB** and **RecordData_liveEMG** are created in the beginning.

For data analysis two instances of the classes **liveEMG** and **liveEEG_CNN** are created in the beginning. Their methods fit() and classify_live() are called throughout the script. With changing two lines of code, the EEG class for machine learning created b Mirjam Hemberger can also be included.

Instances of the classes **RobotHand** and **RobotArm** are used to control the robotic hand and arm.

Record Data: *record_data_liveEEG_JB.py & record_data_liveEMG_180301*

These are the scripts used for connecting to the lab streaming layer and recording the data. They contain the classes **RecordData_liveEEG_JB** and **RecordData_liveEMG**. The major difference between them is, that the EMG class is able to record force values from an Arduino connected to “COM3” (change port if required!) and the EEG class pushes a stream of live spectrograms to the lab streaming layer. By setting the variable self.docontrol = True a thread will be added to do live EEG classification directly in the script. This option is yet to be developed. For now, this is dealt with in the class liveEEG_CNN.

Important methods

- **add_trial():** called in every trial, saves the current timestamp as the beginning of a trial
- **start_recording():** starts the recording
- **pause_recording()** and **restart_recording()** stop and restart the recording thread
- **pause_recording_and_dummp(), stop_recording_and_dump():** used to save data to a .m file and pause/stop recording
- **get_last_trial():** Important in the online mode. Returns an **NST_EEG_LIVE** or **NST_EMG_LIVE** object containing the data of the last trial

EMG posture and force detection: *live_EMG_180301.py*

Everything is implemented in the **class liveEMG**. It is based on the Jupyter notebook for EMG offline decoding.

- The **constructor** receives the directory and filename of a .m file which includes the data of one recording that should be used for training of the model. It generates an NST_EMG_LIVE objects and does all the pre-processing.
- The function fit() does the actual training of the model.
- The function **classify_live()** receives an NST_EMG_LIVE object (e.g. one created by get_last_trial() in RecordData_liveEMG), which it pre-processes and classifies.

EEG CNN motor imagery: *live_EEG_CNN_180403.py*

Everything is implemented in the **class liveEEG_CNN**. The structure is the same as in the class liveEMG. The function **fit()** receives an input parameter "**load**". If True a model with the name being defined in the code will be loaded. Otherwise the model will be trained based on the data selected in the constructor. This takes a very long time. The function **classify_live()** receives an NST_EEG_LIVE object.

Robotic hand: *robothand_arduino_180321.py*

Everything is implemented in the **class RobotHand**. The constructor is called with the serial port number to which the robotic hand is connected (e.g. "COM4" on the recording PC) and the baud rate. The serial port number can either be found in the device settings or by opening the Arduino application and checking to which port an Arduino is connected. The function **do_posture()** receives an integer (0 = fist, 1 = pinch_2, 2 = pinch_3), adds 1 and sends the command to the robotic hand via the serial port. The robotic hand will do the posture for "duration" seconds and will then go back to open hand and wait there for "duration" seconds. **shutdown()** is self-explanatory.

Arduino script

The robotic hand is precompiled with an Arduino script. The script can be found in the folder robothand_doposture. It is programmed to receive characters from 0 to 3 via serial port. Try the special command when sending a 7 (entering a 6 into do_posture()). It is based on fingerlib which is provided in step 6 of the following instructions: https://openbionicslabs.com/obtutorials/ada-v1-assembly

Robotic Arm: *robotarm_KUKA_180405.py*

Everything is implemented in the **class RobotArm**. The script communicates with the KUKA arm by sending chars via UDP. The function **do_posture()** receives integers (0 = move left, 1 = move right, 2 = both hands -> move down) and sends a command to the robotic arm to go to the specified position. The function **return_home()** lets the robotic arm return to the central position.

Operating the KUKA robotic arm

Starting the robotic arm

- turn on robotic arm with red handle on the power box of the robotic arm
- keep control instrument close to where you sit, to be able to press the emergency stop
- connect own PC to hand and to network with an Ethernet cable to send UDP messages
- switch on control computer, password: ***dhrikarl***
- start robot on instrument: ***Ackn All → bottom left + position + with arrows reach FRIC_5ms / yellow enter***
- turn on motors with big (I) button on top
- S and I are green → press green button on the back once → R turns red → press green again → R turns green

- On PC: start terminator to open shell (ctr shift e + ctr shift o to split shell)
    - 1st shell ***roscore***
    - 2nd shell **roscd LWR_Zied**
    - 2nd: ***sudo su***
    - 2nd :enter password
    - 2nd : ***rosrun LWR_Zied robotControlEEG***

NOW THE ROBOTIC ARM IS READY TO RECEIVE COMMANDS VIA UDP

Shutting Down the Robotic arm

- Send 4 to finish the experiment: display R turns black and written successfully closed
- Press "program" on top → 7 to cancel
- Possible to restart with bottom left + position
- Use (0) on top to stop motors
- Shut down robot with big red handle

Changes to gumpy

Dataset class for live EEG: *nst_eeg_live.py*

This file contains the **class NST_EEG_LIVE**. It is a modification of the class NST_EEG. The important difference is the function **load_from_mat()**. It is used for online processing and leads data from matrices handed over as arguments instead of loading data from a .m file. Important note: Contrary to load() It does not subtract the trial offset from the indices that indicate the beginning of the trial. Hence the indices handed over should really indicate the beginning of the whole trial and not just the beginning of active motor imagery. It is programmed this way because of the nature of the live data.

Dataset class for live EMG: *nst_emg_live_180301.py*

This file contains the **class NST_EMG_LIVE**. It is a modification of the class NST_EMG. The important difference is the function **load_from_mat()**. It is used for online processing and leads data from matrices handed over as arguments instead of loading data from a .m file.

Dataset

Both new scripts defining the two new dataset classes need to be added to the folder gumpy/data. In order to recognise them the __init__.py in the folder needs two additional lines of code:

```
from .nst_emg_live_180301 import NST_EMG_LIVE
from .nst_eeg_live import NST_EEG_LIVE
```

features.py

The following function has to be added:

```
def sequential_feature_selector_realtime()
```

It is the same as sequential feature_selector() besides its return arguments.

```
return feature_idx, cv_scores, algorithm, sfs
```

It also returns the sfs object, which is lateron used for classification.

Utils.py

The following function needs to be included:

```
def extract_trials_corrJB()
```

It is similar to extract_trials() but works more stable. Do look into the code for more details.