TOOLBOX ARCHITECTURE

I. Toolbox

NeuroPrime was built from the ground up on Python open source language, synthesizing and using the best parts, we extensively tested, from specific BCI and EEG modules, for signal acquisition, signal processing/classification and signal presentation (diagram in Figure 1). Signal Acquisition: pycorder (Brain Products, Gilching, Germany), pylsl/lab streaming layer [1], and mushu [2]. Signal processing/classification: wyrm [3] and mne [4]. Signal presentation: pyff [5] and psychopy [6]. Additionally, some other important packages, pandas for managing data, matplotlib for graphs, numpy for arrays, scipy for specific algorithms, pygatt for bluethooth connectivity with GSR and HR sensors, and also pyqtgraph for real-time graphical interfaces [7]–[9].

This framework was built, following the necessity for further expandability, utility and reusability from the neuroscience community. In fact, Python has great modules in machine learning that could help optimize and automate the paradigm of priming subjects in future experiments. Also, the code is ready to use with any EEG amplifier that can connect with the lab streaming layer (LSL) [1].

The name chosen for the package was NeuroPrime, a combination of neuromodulation and priming. For more information on NeuroPrime and access to the last version go online [10]. The software will be available as soon as this paper is published. Consequently, in the upcoming months, if one has an EEG, then one can try out the framework, use it for research, and contribute to its development.

In the next section we go deeper into the Pythonic structure of each of the 3 parts that constitute the software: signal acquisition; signal presentation; signal processing (& classification). They work together to bring the framework online. Additionally, we also refer to the offline processing.

A. *Signal Acquisition*

When using Human Computer Interfaces (HCI) in experiments, one always has to use special hardware (e.g. EEG, GSR, HRV or other amplifiers) to acquire the signals. This hardware usually comes with their own software and different vendors have different formats for saving and online streaming. Additionally, these types of experiments require synchronization markers. Markers are label signals that mark certain time point in the data. These markers are also distinct between vendors. In summary, signal acquisition is comprised of data servers (outlets from amplifiers) and data clients (inlets to storage and further processing).

Therefore, in order for the signal acquisition to be open source it has to meet certain requirements: software that runs on all major operating systems (OS), supports a wide range of EEG hardware, that produces and outputs data in a standardized format independent from the amplifier used, receives markers via network interface, that is free and open source software.

For that, we found in the Python library great software like pylsl and mushu. Also, specifically to our case, the amplifier from Brain Products used in the validation experiments, Actichamp (Brain Products, Germany), has a Python driver for signal acquisition named pycorder. Nonetheless, one can use our toolbox with any amplifier supported by LSL.

a) Pylsl

LSL is the magic behind the streaming of data, allowing for a wireless open source design. LSL is an overlay network for real-time exchange of time series between applications, most often used in research environments. LSL has clients for many other languages and platforms that are compatible with each other. Pylsl is the Python interface to LSL. This way, one can submit (outlet) and request (inlet) data from the network using Python. In addition, many of the most known amplifiers are supported already, and if is not supported one can create the driver [1].

b) Pycorder

Pycorder is outdated but is a solid package with built in remote data access (RDA) server. It enables the configuration of the EEG: electrode impedance check, signal data visualization, reference selection, selection of working modes (default and shielding mode) and saving. It's an important package to work with the amplifier. However, in closed-loop real time applications it can be substituted by LSL server app [11], [12] that has shorter delays of about 2 ms, while pycorder RDA server is about 50-200 ms. The RDA delay is still within accepted range for EEG NFT [13], [14], and the RDA server can also be used in conjunction with LSL app to create a data outlet stream that will then be pulled from the network by pylsl inlet request (Figure 2).

c) Mushu

Mushu is a package that creates an amplifier class with the necessary methods to configure, start, stop, save and quit the signal acquisition in the client side [2]. Mushu is still maintained, but certain limitations were patched during development, mainly concerning the saving (relative timestamps instead of only absolute timestamps) and bad interaction with pylsl driver. This patch, named "mushu_patch", is located in the "signal_acquisition" folder.

It is a great and simple package with retrieval and online streaming of EEG data. However, in the future, instead of the patch developed for mushu, an even simpler amplifier class should be created to work directly with pylsl without the need of mushu. The versatility of mushu is to be able to function as a Python library or a stand-alone server (with some amplifier divers already built in for streaming data and markers online). Using the software as a Python library was convenient because our signal processing is in Python and we could use it to create a single application (Figure 2).

d) Data structure

We use two types of saving structures: one that follows the MNE Python package standard Neuromag FIF file format [4] and the mushu numpy format [2]. The fif format is a more complex file that can in fact save more information and can be directly used by MNE, markers and data are all saved in the same file, while the mushu saves in 3 files data (.dat), markers (.markers) and meta (.meta). For the data, each received packet from the amplifier Mushu translates the data into a matrix where each column is mapped to a channel (i.e. channel 1 to column 1, etc.) and the rows maintain the ordering of the measured values per channel in time. Having the EEG data stored in NumPy arrays is convenient and efficient on those arrays, that is mandatory for the later signal processing steps.

The interchange between formats is also easy to do. The same can be said for the import of data to matlab. Nonetheless, in the future, a possible better idea is to port entirely to MNE format, in this way mushu and wyrm will be a deprecated complexity.

e) NeuroPrime signal acquisition

As referred before we developed a "mushu_patch" for better saving and integration with pylsl. The developed algorithms and specific functions are located in the appropriate folder named "signal_acquisition".

B. Signal processing & classification

The signal processing of any online BCI system (Figure 3) is responsible for transforming brain outputs into actionable signals by detecting and extracting certain patterns from the brain signals. In order to detect those patterns, the raw brain signals have to be preprocessed and specific features (representing the patterns) have to be extracted and classified [15].

The libraries we decided to use for this part were wyrm and MNE.

a) Wyrm

We selected the wyrm pakage due to its ease of use, speed and integration with the signal acquisition package mushu and the signal presentation package pyff. Wyrm tries to cover both toolbox-aspects: wyrm can be used as a toolbox for offline analysis and visualization of neurophysiological data, and in real-time settings, like an online BCI experiment [3]. The list of algorithms includes: channel selection IIR filters, sub-sampling, spectrograms, spectra, baseline removal for signal processing, Common Spatial Patterns, Source Power Co-modulation, class wise average, jumping means, signed r2-values for feature extraction, Linear Discriminant Analysis with and without shrinkage for machine learning, and others. It is worth mentioning that the data structure is readily compatible with scikit-learn [16], therefore a wide range of machine learning algorithms are at ones disposal (cross validation, support vector machines, k-nearest neighbors, independent- and principal component analysis, and many more).

b) MNE

Nonetheless, wyrm has only the necessary functions for online/offline preprocessing, processing and classification, and is not constantly updated. Therefore, for advanced processing and active update and maintenance by the community we found MNE-Python [4]. The MNE-Python package provides a complete pipeline for MEG and EEG data analysis: covers preprocessing, forward modeling, inverse methods, and visualization; advanced analysis for time-frequency, statistics, and connectivity; enables fast and memory-efficient processing of large data sets. While MNE-Python is designed to integrate with packages within the Python community, it also seamlessly interfaces with the other components of the MNE suite, like for example the MNE-matlab, because it uses the same Neuromag FIF file format, with consistent analysis steps and compatible intermediate files. Therefore, going forward, we expect to fully transition to MNE-Python for signal processing and classification.

c) NeuroPrime signal processing

For signal processing we developed an online/offline pipeline for closed loop NF and offline data analysis of NF data, named "nftalgorithm", a children class of "eegpipeline" that imports functions mainly from "eegfunctions". The "eegpipeline" standardizes and defines the necessary methods for the pipeline: initiate vars, input data, preprocessing, processing, postprocessing and output data. These developed algorithms and specific functions are located in the appropriate folder named "signal_processing".

C. Signal Presentation

Feedback and stimulus presentation are another fundamental piece of any BCI system, which relates to the interaction and interface with end user. While in the stimulus presentation the interaction is passive (e.g. the subject does usually not control anything with his thoughts), in the feedback presentation the interaction is active, where the participant intentionally tries to modulate the signal that is being presented (Figure 4).

The presentation applications need to receive data from the signal processing and send markers to the signal acquisition. Furthermore, they should give researchers conducting experiments some means for interaction with the running application. Another important aspect is the usability of the package by the community of researchers. Therefore, a framework should make as easy as possible the development of feedback and stimulus applications, by providing the right infrastructure.

The packages meeting these requirements are pyff and psychopy.

a) Pyff

Pyff is a very solid and simple package for researchers familiar with Python. Pyff provides a platform for running feedback and stimulus applications and interact with them, and it provides a framework for developing new applications [5]. Pyff also provides some standard BCI paradigms (stroop test, oddball, etc). The limitations with pyff are the fact that it is outdated, it is not maintained and also was not ported to Python 3.

b) Psychopy

Due to the aforementioned issues, we also implemented psychopy functions. Psychopy provides a richer and research friendly environment (both for experienced or non-experienced programmers) for developing feedback and stimulus applications. It is also actively updated and maintained [6]. Therefore, in the future, psychopy should become the main package for signal presentation and also construction of session paradigms.

c) NeuroPrime signal presentation

For signal presentation we developed a feedback and stimulus application class named "feedbackclass" that supplies all the necessary methods for creating a pyff task. Then, from this class we implemented a loop class, named "loopfeedbackclass" that enables a continuous and sequential construction of multiple tasks within a session (for examples go to the experiments sections). These algorithms and specific functions are located in the appropriate folder named "signal_presentation". Additionally, the stimuli used for the signal presentation are inside the root folder "stimulus".

D. *Offline data analysis*

The offline data analysis pipelines for EEG advanced signal processing/classification and automation of the pipeline are performed in Python MNE package [4]. Then, after extracting features we create a dataset by using a factorial design ("for" loops) with the Pandas package and json files to easily parse the dataset to excel and create a database. Finally, this database is used for multivariate statistical analysis. These data analysis functions are located in the folder "utils/data_analysis".

E. *Integration of all packages*

Simplicity and reusability are the foundation of NeuroPrime package, as is intended to be an open source project to be used by the neuroscience community. It is also intended to be a BCI hub that evolves by synthesizing the best packages the Python community has to offer, in terms of signal processing, signal presentation and signal acquisition. Therefore, it should supply an easy and simple structure to update and connect new packages within the same design. This way, a lot of effort was directed to its structure design and simplification of the methods. This work can be fully appreciated in the scripts, while in this section we give it an overview.

For that, we decided to follow Python hierarchical class construction (the root directory for the source code is "src"). An initial class, named "initbciclass", initiates all the methods and modules from signal acquisition, signal processing and signal presentation. This class is the hub for connecting and working with all the packages, is where all the fundamental functions reside. From this class we can develop child classes for each task needed on a BCI session.

In Figure 5 we provide an example of the simplicity of constructing a "testclass" for a rest task, this which is located in the folder "brain_interfaces". To fully run the "testclass", one first needs to simulate the signal acquisition data and signal presentation markers. For simulating the signal acquisition, one has two methods the script "file_stream_player" or the "outlet_actichamp" inside folder "utils/simulate". The "file_stream_player" is the preferred method, which makes use of the data sample in the folder "sampledata", while the "outlet_actichamp" generates a sinusoidal signal. For the signal presentation, the "outlet_pyffmarker" inside folder "utils/simulate" should be used, that generates markers similar to the pyff signal presentation. Secondly, one needs to run the "testclass". Each variable inside the script serves a specific purpose described in the figure (for a full documentation go to the package online). Additionally, as one can see this hierarchical class inheritance serves the purpose to simplify debugging and as a means of quality assurance for each task class, where each task is tested individually, but then can be ensembled in a BCI session, as one will see in the next section .

For easy deployment of experiments a "start_gui" batch initiates the necessary Python environment and a basic GUI (Figure 6) to start all the amplifiers, signal acquisition, signal presentation and signal processing (batchs scripts located in the "batchs" folder). Additionally, the root folder named "modules" is the directory for modules that need to be inside of the package for simplicity, like pyff.

F. *Toolbox validation*

Concerning the validation two experimental designs were implemented. In this guide we show the GUI of a single-session of Experiment 1 in Figure 7 and Experiment 2 in Figure 8. As one can see, they diverge in the temporal design, but also in other attributes. For the justification of the choice of attributes in the Experiment 1 design, please refer to the following publication [17]. Experiment 2 results are still in submission, however as an example of a complete BCI, the Experiment 2 pipeline can be found in the folder "brain_interfaces". There one can find, child classes similar to the above "testclass" for the rest baseline task (REST), with the file named "restclass"; the priming task (PRIME), "primeclass"; the NFT task, "nftclass"; and the bci class "bci" that initiates all the other tasks and creates the necessary experimental session. As such, each class works as standalone tasks, great for debugging and as a means of quality assurance, but they can also be ensembled in "bci" to create a closed-loop real-time NFT session.

With relevance, online performance of the experiments were similar, even using different methods for data acquisition, because we blocked the update of data acquisition to segments of ≈ 250 ms. In Experiment 1, we used the LSL server app [11], [12] that has shorter delays of about 2 ms, and in Experiment 2 we used pycorder RDA server with delays about 50-200 ms. Pycorder was used in Experiment 2, due to the powerful GUI for electrode preparation and the fact that the LSL server app has limitations and artifact problems. Actually, a new LSL browser app [18] is being developed to substitute the previous server app.

One of the requirements was a software that runs on all major operating systems. As such, NeuroPrime was developed and tested on Mac OS and Windows OS and depending on the hardware requirements it can run in all major OS that run all the Python packages. Currently, the package has some limitations or more appropriately a to-do list: parse the package completely from Python 2.7 to Python 3; documentation needs to be reviewed and simplified; variables nomenclature needs to be reviewed for standardization (e.g standardization of uppercase and lowercase); deprecated code should be removed; implementation of machine learning algorithms; among others. In retrospective, this package is stable in the current version, but should be continuously simplified, tested and updated to meet the criteria of new experiments.

References

[1] SCCN, "sccn/labstreaminglayer: LabStreamingLayer super repository comprising submodules for LSL and associated apps.," 2014. https://github.com/sccn/labstreaminglayer (accessed Apr. 17, 2020).

[2] B. Venthur and B. Blankertz, "Mushu, a free- and open source BCI signal acquisition, written in Python," in *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, 2012, pp. 1786–1788, doi: 10.1109/EMBC.2012.6346296.

[3] B. Venthur and B. Blankertz, "Wyrm, A Pythonic Toolbox for Brain-Computer Interfacing," Dec. 2014, Accessed: Apr. 20, 2020. [Online]. Available: http://arxiv.org/abs/1412.6378.

[4] A. Gramfort *et al.*, “MEG and EEG data analysis with MNE-Python,” *Front. Neurosci.*, no. 7 DEC, 2013, doi: 10.3389/fnins.2013.00267.

[5] B. Venthur *et al.*, “Pyff – A Pythonic Framework for Feedback Applications and Stimulus Presentation in Neuroscience,” *Front. Neurosci.*, vol. 4, p. 179, Dec. 2010, doi: 10.3389/fnins.2010.00179.

[6] J. W. Peirce, “PsychoPy-Psychophysics software in Python,” *J. Neurosci. Methods*, vol. 162, no. 1–2, pp. 8–13, May 2007, doi: 10.1016/j.jneumeth.2006.11.017.

[7] E. Jones, T. Oliphant, and P. Peterson, “SciPy: Open source scientific tools for Python,” 2001, Accessed: Apr. 22, 2020. [Online]. Available: https://www.scienceopen.com/document?vid=ab12905a-8a5b-43d8-a2bb-defc771410b9.

[8] T. E. Oliphant, “Guide to NumPy,” 2006. Accessed: Apr. 22, 2020. [Online]. Available: http://www.trelgol.com.

[9] W. Mckinney, “Data Structures for Statistical Computing in Python,” 2010. Accessed: Apr. 22, 2020. [Online]. Available: http://conference.scipy.org/proceedings/scipy2010/pdfs/mckinney.pdf.

[10] N. M. C. da Costa, “nmc-costa/neuroprime: A framework for real-time HCI/BCI. Specifically developed for advanced human-computer assisted self-regulation of Neurofeedback.,” *github*, 2020. https://github.com/nmc-costa/neuroprime (accessed Apr. 19, 2020).

[11] Brain Products GmbH, “labstreaminglayer/App-BrainProducts/ActiChamp,” 2015. https://github.com/labstreaminglayer/App-BrainProducts/tree/master/ActiChamp (accessed Apr. 15, 2020).

[12] Brain Products GmbH, “Brain Products Press Release: Ultra fast closed loop applications combining actiCHamp amplifier with Lab Streaming Layer (LSL),” *Brain Products*, 2017. https://pressrelease.brainproducts.com/closed-loop-actichamp-lsl/ (accessed Apr. 16, 2020).

[13] J. Gruzelier, “EEG-neurofeedback for optimising performance. III: A review of methodological and theoretical considerations.,” *Neurosci. Biobehav. Rev.*, Mar. 2014, doi: 10.1016/j.neubiorev.2014.03.015.

[14] S. Enriquez-Geppert and R. Huster, “EEG-neurofeedback as a tool to modulate cognition and behavior: a review tutorial,” *Front. Hum.*, 2017, Accessed: Mar. 30, 2017. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5319996/.

[15] M. Cohen, *Analyzing neural time series data: theory and practice*. 2014.

[16] F. Pedregosa *et al.*, “Scikit-learn: Machine learning in Python,” 2011. Accessed: Apr. 20, 2020. [Online]. Available: http://scikit-learn.sourceforge.net.

[17] N. M. C. da Costa, E. G. Bicho, and N. S. Dias, “Priming with mindfulness affects our capacity to self-regulate brain activity?,” in *2020 IEEE 8th International Conference on Serious Games and Applications for Health (SeGAH)*, Aug. 2020, pp. 1–8, doi: 10.1109/SeGAH49190.2020.9201841.

[18] Brain Products GmbH, “brain-products/LSL-actiCHamp: LSL connector for the actiCHamp (plus) device from Brain Products.,” *Brain Products*, 2020. https://github.com/brain-products/LSL-actiCHamp (accessed Apr. 19, 2020).

II. Images

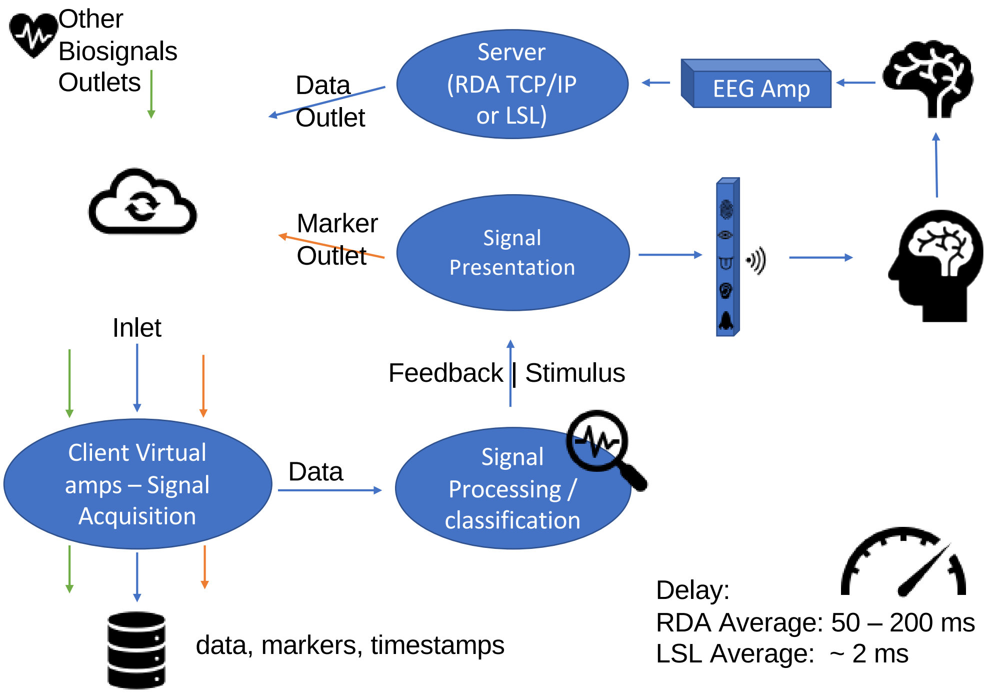


Figure 1 – NeuroPrime. Online closed loop BCI Python framework. It's comprised of 3 main parts: signal acquisition, signal processing and signal presentation.

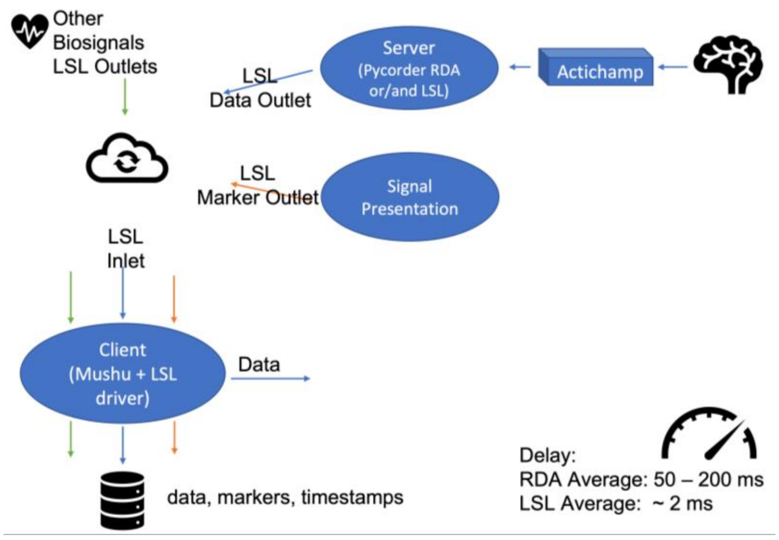


Figure 2 - Signal acquisition diagram and main packages used. Input data and markers from servers and output data to client storage and processing.

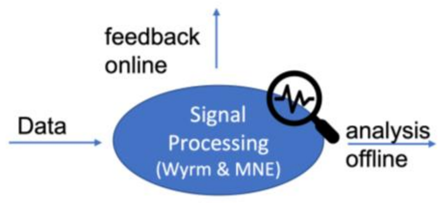


Figure 3 - Signal processing diagram and main packages used. Input data and output specific features for online or offline analysis.

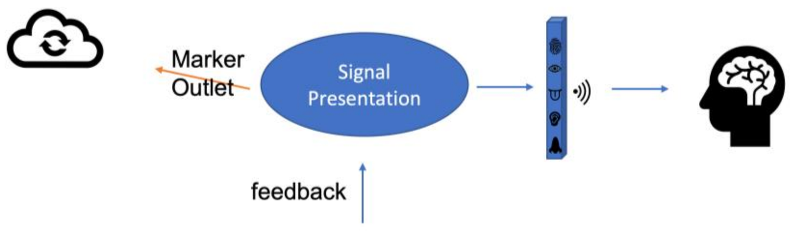
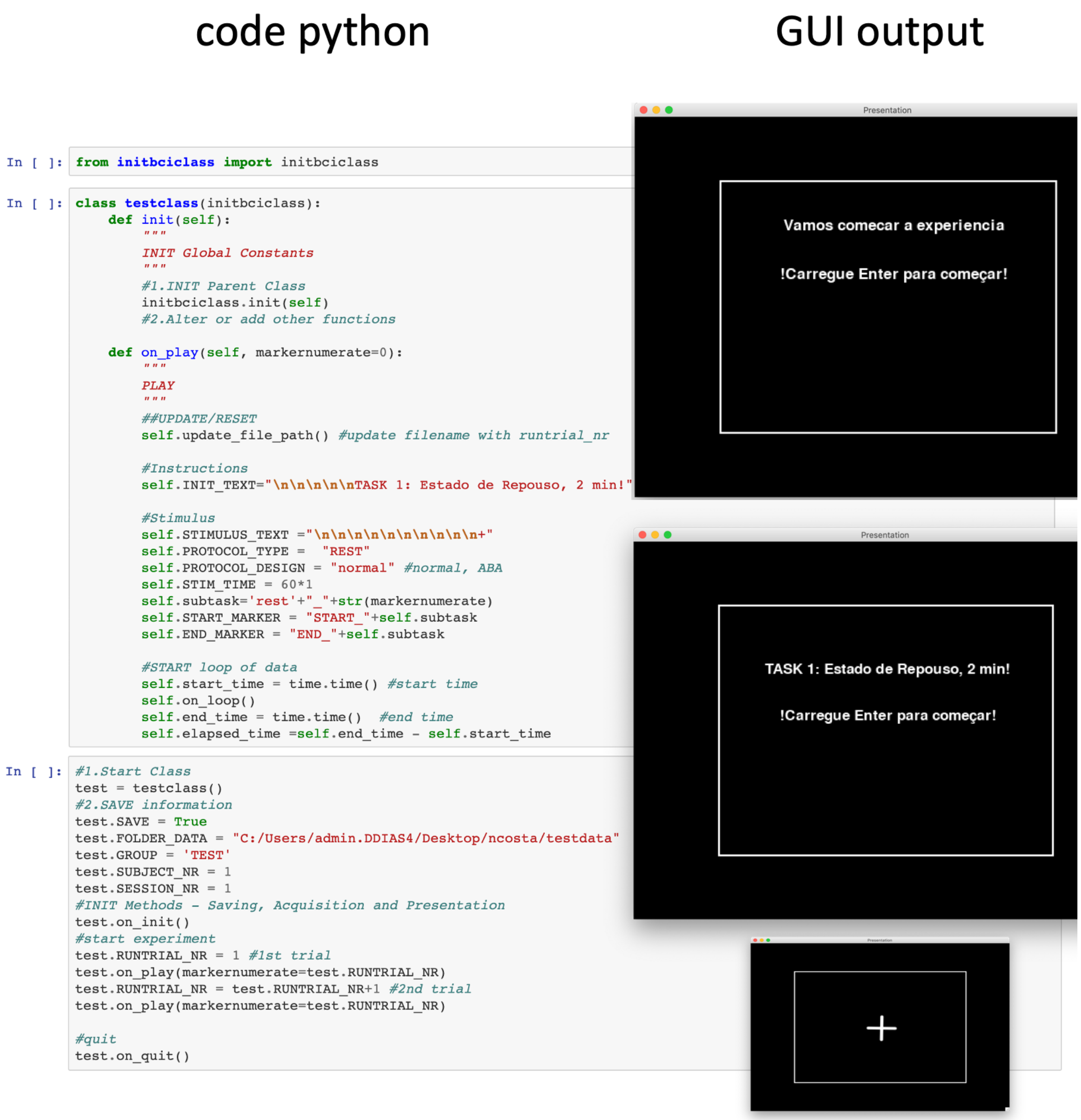


Figure 4 - Signal presentation diagram and main packages used. Input feedback data and output to interface and synchronization markers.


code python
GUI output
Presentation
Vamos comecar a experiencia
!Carregue Enter para começar!
Presentation
TASK 1: Estado de Repouso, 2 min!
!Carregue Enter para começar!
Presentation
+

```
In [ ]: from initbciclass import initbciclass

In [ ]: class testclass(initbciclass):
            def init(self):
                """
                INIT Global Constants
                """
                #1.INIT Parent Class
                initbciclass.init(self)
                #2.Alter or add other functions

            def on_play(self, markernumerate=0):
                """
                PLAY
                """
                ##UPDATE/RESET
                self.update_file_path() #update filename with runtrial_nr

                #Instructions
                self.INIT_TEXT="\n\n\n\n\nTASK 1: Estado de Repouso, 2 min!"

                #Stimulus
                self.STIMULUS_TEXT ="\n\n\n\n\n\n\n\n\n\n+"
                self.PROTOCOL_TYPE =  "REST"
                self.PROTOCOL_DESIGN = "normal" #normal, ABA
                self.STIM_TIME = 60*1
                self.subtask='rest'+"_"+str(markernumerate)
                self.START_MARKER = "START_"+self.subtask
                self.END_MARKER = "END_"+self.subtask

                #START loop of data
                self.start_time = time.time() #start time
                self.on_loop()
                self.end_time = time.time()  #end time
                self.elapsed_time =self.end_time - self.start_time

In [ ]: #1.Start Class
        test = testclass()
        #2.SAVE information
        test.SAVE = True
        test.FOLDER_DATA = "C:/Users/admin.DDIAS4/Desktop/ncosta/testdata"
        test.GROUP = 'TEST'
        test.SUBJECT_NR = 1
        test.SESSION_NR = 1
        #INIT Methods - Saving, Acquisition and Presentation
        test.on_init()
        #start experiment
        test.RUNTRIAL_NR = 1 #1st trial
        test.on_play(markernumerate=test.RUNTRIAL_NR)
        test.RUNTRIAL_NR = test.RUNTRIAL_NR+1 #2nd trial
        test.on_play(markernumerate=test.RUNTRIAL_NR)

        #quit
        test.on_quit()
```

Figure 5 - Example of a rest task using the "initbciclass". The notation used in coding is Python language on the left, while on the right is the GUI output of the script.

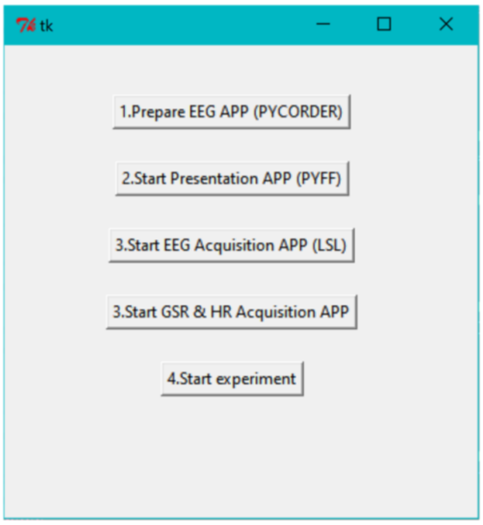

tk
1.Prepare EEG APP (PYCORDER)
2.Start Presentation APP (PYFF)
3.Start EEG Acquisition APP (LSL)
3.Start GSR & HR Acquisition APP
4.Start experiment

Figure 6 - Start GUI. This simple graphic user interface helps the technician setup the experiment. 1) Preparing the amplifier. 2) Initiate the presentation app. 3) Initiate the acquisition app. 4) Initiate other amplifiers like the GSR and HRV. 5) Start the BCI session.

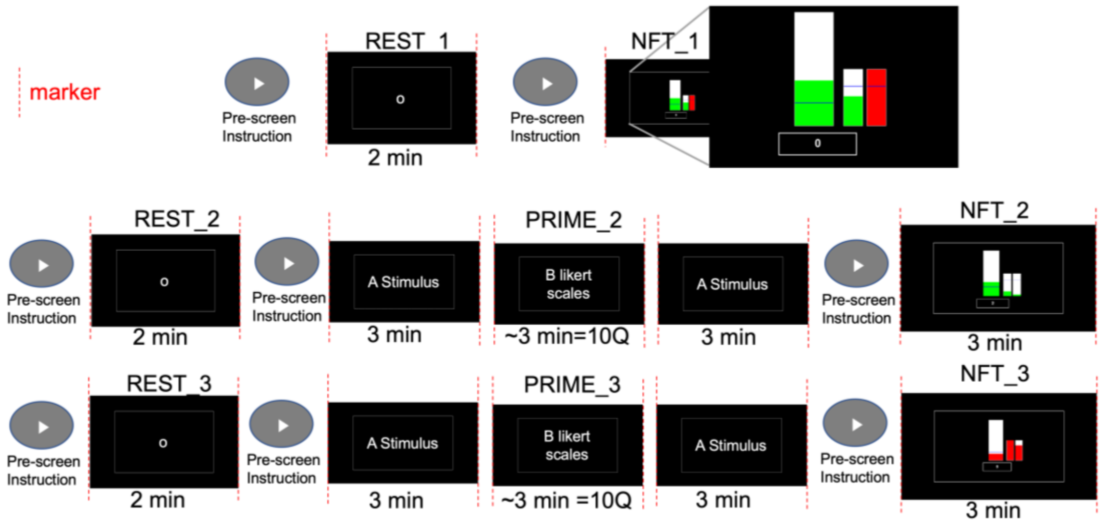


Figure 7 – Experiment 1GUI representation of the full session of EG. Time flows from left to right, top to bottom. In a single session, first the subject fills traits self-reports. Then, the training starts. There are three blocks in total, and each begins with the REST task with eyes open and ends with the NFT. In block 2 and 3, the PRIME is applied for the EG but not for the CG. PRIME follows an (A)(B)(A) design, where stimulus (A) is a mindfulness guided practice, while (B) are Likert scale questions in random order (total of 10 questions). NFT bars represent the neurofeedback presented to the subject, and the horizontal line within the bars represents the thresholds.

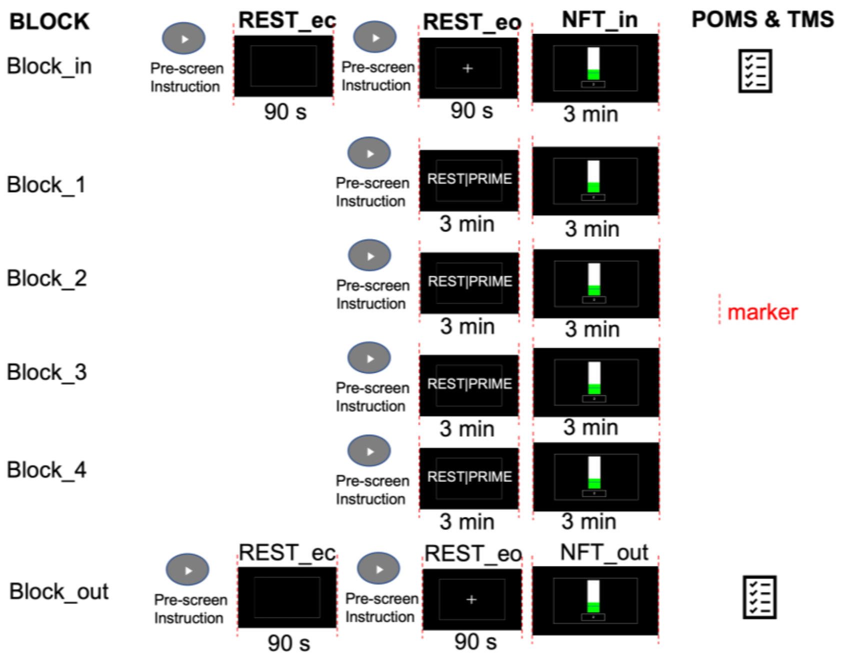


Figure 8 – Experiment 2 GUI representation of the full session. Time flows from left to right, top to bottom. In a single session, first the subject fills the traits self-reports, then, the training starts. There are 6 blocks in total. Block in and Block out each begins with rest state with eyes closed then eyes open, followed by alpha NFT and. From block 1 to 4, in the EG first is the PRIME task then NFT. In the control group PRIME is substituted by REST. Also, from block 1 to 4, eyes closed and eyes open are randomized between blocks.