Article

NeuXus: A Biosignal Processing and Classification Pipeline for Real-Time Brain-Computer Interaction

Athanasios Vourvopoulos [1,*,†,‡] 0000-0001-9676-8599**, Simon Legeay** [2,‡] **and Patricia Figueiredo** [1,†]

1   Institute for Systems and Robotics - Lisboa, Instituto Superior Técnico, Universidade de Lisboa; {athanasios.vourvopoulos, patricia.figueiredo}@tecnico.ulisboa.pt

2   CentraleSupélec; simon.legeay@supelec.fr

*   Correspondence: athanasios.vourvopoulos@tecnico.ulisboa.pt;

†   Current address: Av. Rovisco Pais, 1, 1049-001 Lisboa, Portugal

‡   These authors contributed equally to this work.

Abstract: In the last few years, Brain-Computer Interfaces (BCIs) have progressed as an emerging research area in the fields of human-computer interaction and interactive systems. This is primarily due to the introduction of low-cost electroencephalographic (EEG) systems that render BCI technology accessible for non-medical research but also due to the advancements of signal processing and machine learning methods. Consequently, BCIs could provide a wide new range of possibilities in the way users interact with a computer system (e.g., neuroadaptive interfaces). However, major challenges must still be addressed for BCI systems to mature into an established communication medium for effective human–computer interaction. One of the major challenges involves the easy integration of real-time processing pipelines with portable EEG systems for an "out of the lab" use. To date, despite the amount of options current open-source tools provide, most toolboxes focus mainly in extending the processing and classification methods but lack on the ability to provide an easy-to-design yet extensible architecture for ubiquitous use. Here, we present NeuXus, a modular toolbox in Python for real-time biosignal processing and pipeline design. NeuXus is open-source and platform independent, providing high-level implementation of processing pipelines for easy BCI design and deployment.

1. Introduction

Brain-computer Interfaces (BCIs) can be described as communication systems able to establish an alternative pathway between the user's brain activity and a computer, providing an additional non-muscular channel for communication and control to the external world [1]. This can be achieved either explicitly by allowing users to issue direct commands into devices without physical involvement of any kind (e.g., speller typing, wheelchair control), or implicitly by monitoring a user's state (e.g., workload level, attention state) to proactively adapt a user-interface or a virtual environment.

To date, the use of non-invasive Electroencephalography (EEG) is the most common brain signal acquisition technique in BCIs. EEG utilizes electrodes placed over the heads scalp for capturing the combined electrical activity of "firing" neurons. These neurons have characteristic intrinsic electrical properties, producing electrical and magnetic fields which propagate through volume conduction over the scalp [2]. Through this evoked electrical potential, different EEG patterns of brain activity are activated depending of the mental task. This oscillatory brain activity is currently used for the interfacing of humans with computers in a EEG-based BCI system. Some of the benefits of using EEG in BCIs include the high temporal resolution (1ms), EEG systems they are relatively lightweight and portable, and finally the cost

compared to other brain imaging technologies (e.g., Magnetoencephalography), allows for more flexible data collection in real-world environments. On the downside, EEG signals are easily contaminated with artifacts and noise of non-brain sources, and have low spatial resolution, making source localization hard to achieve and rendering their utilization cumbersome [3].

A unidirectional brain-to-computer communication is elicited first by a stimulus (visual, auditory or somatosensory), which is generating endogenous or exogenous potentials. Endogenous potentials are considered those that their occurrence is not related to the physical attributes of a stimulus (e.g. frequency or intensity), but to a person's reaction to it. For example, motor-imagery BCI (MI-BCI), involves the imagination of movement and is considered a BCI paradigm that evokes endogenous potentials since users are generating motor-related brain patterns unrelated to the stimulus attributes [4]. In contrast, BCIs using Steady State Visual Evoked Potentials (SSVEP) are considered an exogenous potential stimulus since it is caused by external visual stimulation of flashing lights which occur at the primary visual cortex of the brain [5]. In addition, P300 BCIs are also receive external visual simulations using brain responses that are generated approximately 300 ms after stimulus onset (hence the name P300). Nonetheless, P300 are considered to be also endogenous potentials since after the stimulus presentation, there is a stimulus validation process (a cognitive function) by the brain [6]. Both SSVEP and P300 paradigms are mostly used for patients in the locked-in state for communicating with the outside world [7], while MI-BCI is capturing activity over the motor and somatosensory cortices and is utilized for motor restoration in neurorehabilitation [8,9].

Undoubtedly, BCIs have not only been proven to be important tools in the medical domain as either assistive or restorative interfaces but also have introduced a unique form for the human–computer interaction (HCI) paradigm [10]. In the last few years, BCIs have progressed as an emerging research area in the fields of HCI and interactive systems, primarily due to the introduction of low-cost EEG systems that render BCI technology accessible for out of the lab research moving closer to the integration with wearable technology [11]. Consequently, BCIs provide a wide new range of possibilities in the way users interact with a computer system (e.g., neuroadaptive interfaces). However, major challenges must still be tackled for BCI systems to mature into an established communication medium for effective human–computer interaction.

One of the major challenges involves the easy integration of real-time processing pipelines with portable EEG systems for an out-of-the-lab use. To date, despite the amount of options current open-source tools provide, most toolboxes focus mainly in extending the processing and classification methods but lack on the ability to provide an easy-to-design yet extensible architecture for ubiquitous use. Here, we present NeuXus (**neu**ral + ne**xus**, latin for connection, pronounced IPA: /'nek.sus/), a modular toolbox in Python for real-time biosignal processing and pipeline design.

2. Background

In EEG-based BCIs, specialized algorithms are responsible for translating EEG signals into control commands to devices in order to provide feedback. To date, a plethora of software tools for EEG processing are available for both basic and clinical/translational research. This is highlighted in a significant amount of bibliometric reviews of the state-of-the-art of BCI software tools that go back up to 20 years of BCI research [12] [13,14] [15].

Current software for EEG processing is highly diverse and it involves tools ranging from general purpose processing of electrophysiological signals up to more specific for BCI or neurofeedback use. In many cases, the application level (general vs specific) of the EEG/BCI tool dictates also the programming environment that is utilized for implementing the processing. For instance, the use of MATLAB is more common in most of the general-purpose biosignal or EEG signal processing tools for more than 20 years.

On the other hand, first with C++ and next with Python, BCI-specific tools have emerged for consumer products and emerging applications.

In MATLAB, a comprehensive set of EEG tools have been developed as toolboxes but also as standalone versions. Specifically, EEGLAB (Swartz Center for Computational Neuroscience, UCSD) [16] is one of the most widely used tools for EEG analysis, incorporating independent component analysis (ICA), time/frequency analysis, artifact rejection, event-related statistics, and several visualization features. Moreover, EEGLAB supPorts a large amount of plugins which extend its functionality. Specifically, for real-time EEG processing, the BCILAB plugin is supported for the design, prototyping, testing, experimentation and evaluation of BCIs [17]. Similarly, Fieldtrip is also a MATLAB toolbox for EEG, MEG and NIRS analysis. Like EEGLAB, it offers pre-processing and advanced analysis methods, such as time-frequency analysis, source reconstruction using dipoles, and non-parametric statistical testing [18]. Furthermore, for real-time data acquisition the Fieldtrip buffer module is used. The FieldTrip buffer is a network transparent TCP server that allows the acquisition client to stream data to it per sample or in small blocks, while at the same time previous data can be analyzed. One of the oldest and still active EEG processing toolbox for MATLAB is Brainstorm (University of Southern California, Los Angeles, CA) [19]. Brainstorm is incorporating all the necessary processing tools for continuous or event-related data for MEG, EEG, fNIRS, ECoG, depth electrodes and multi-unit electrophysiology. For realtime data acquisition, Brainstorm is also using the FieldTrip buffer. Despite most of MATLAB toolboxes are available for free, MATLAB requires a paid license. Alternatively, the GNU Octave Project is a free alternative to MATLAB. Nonetheless, Octave does not support the entirety of the tools available (such as EEGLAB), with many of the functions not properly working, including the graphical user interface and menus. Some of the standalone processing tools include, BrainVision Analyzer (Brain Products GmbH), BrainVoyager (Brain Innovation B.V.) and LORETA (KEY Institute for Brain-Min Research, Zurich) for estimating cortical connectivity [20].

For specific BCI applications, specialized algorithms and signal processing libraries have been implemented in C++ for increased speed of execution and efficiency. Some of these tools include: BCI2000 [21], one of the earliest BCI tools, TOBI (Tools for Brain-Computer Interaction)[22], BioSig [23], and OpenVibe (Inria, Rennes, France) [24]. Notably, OpenVibe is the only one using visual programming through a graphical interface, rendering it one of the most user friendly BCI tools, especially for entry level users.

In the last few years, many open-source Python software for exploring, visualizing, and analyzing EEG have emerged. For example, MNE-Python provides libraries for supporting the analysis of MEG, EEG, sEEG, ECoG, NIRS data [25], being the mostly used Python tool for post-hoc analysis. Moreover, Wyrm is a BCI toolbox suitable for running on-line BCI experiments as well as, off-line analysis of EEG data together with Mushu [26] for signal acquisition and Pyff, a BCI feedback and stimulus framework [27]. Other tools in Python include Gumpy [28] and BCIPy[29]. In most of these tools, the data acquisition support is extended through the use of the Lab Streaming Layer[1] (LSL) protocol and the Extensible Data Format[2] (XDF)(Swartz Center for Computational Neuroscience, UCSD).

3. Architecture and Functions

NeuXus is a modular software written in Python 3[3] and it requires version 3.7 (or greater) in order to be compatible with current dependencies and third-party libraries. NeuXus processes are represented as

---

[1] https://github.com/sccn/labstreaminglayer
[2] https://github.com/sccn/xdf
[3] https://www.python.org/

Nodes in a graph structure, connected by edges called Ports. NeuXus Nodes can pass data to one another, interact with the operating system or a network stack.

3.1. Nodes and Ports

A Node represents a single function running sequentially in a NeuXus pipeline and managed by the pipeline.py script. Each Node has two ports, an input and an output (Figure 1). Every Node is instantiated with unique name and a set of parameters which are passed to the constructor of the Node during the initialization of the pipeline. During the initialization, Nodes are created, then each Node checks whether the input Port data type is coherent and creates the output port. Next, it initializes the required attributes that are useful for the Node functionality. Finally, it creates a Log of each instance for debugging purposes. After initialization, NeuXus runs the pipeline in a loop. On every loop, input Ports receive several chunk of data as a DataFrame[4], which is a two-dimensional data structure. Next, each DataFrame is processed based on the Node functions and finally it is forwarded to the output Port for the next node. All Nodes update their input and output Port(s) in the same order as they were instantiated in the pipeline.
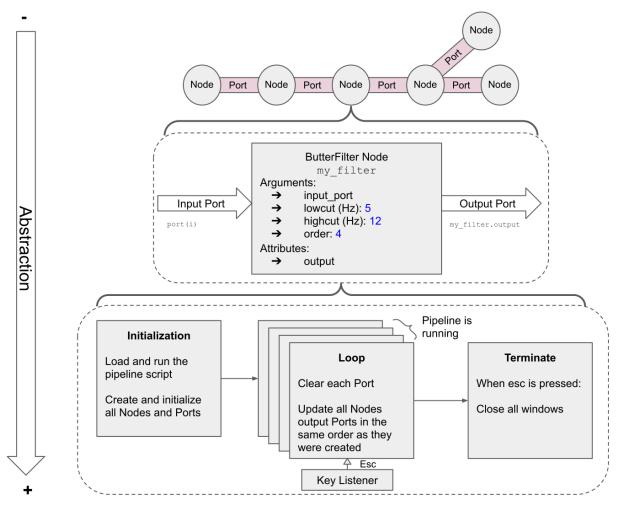
Figure 1. NeuXus Nodes and Ports architecture in three abstracted levels

---

[4]   https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.html

3.2. Functions

Each Node represents a single function in a NeuXus processing pipeline. These processes are grouped under different modules based on their functionality (table 1). Current available modules or core modules are described in the following sections.

3.2.1. IO

The IO module is responsible for implementing input and output Nodes for real-time EEG or other biosignal data handling.

The first supported Node is using LSL, a well established protocol for real-time data measurement and widely supported by the research community. Specifically, two LSL Nodes are available, one for streaming (LSLSend) and one for receiving data (LSLReceive) for both time-series and/or markers.

example:

```
#receive LSL stream
lsl_signal_node = io.LslReceive('name', 'EEGAmp001',
                                data_type='signal',
                                sync='network')


#send "lsl_signal" node output via LSL
lsl_send_node = io.LslSend(input_port=lsl_signal_node.output,
                           name='my_lsl_signal',
                           type='signal')
```

Next, an RDA (Remote Data Access) Node is implemented (RdaReceive), supporting the data transferring protocol from Brain Products (Brain Products GmbH, Gilching, Germany) which allows to remotely access BrainVision Recorder for transferring data to other programs located on the local computer or in the network. The "RdaReceive" Node receives both signals and markers, while it can also be used as a bridge between BrainVision RecView and a NeuXus processing pipeline for the acquisition of real-time corrected EEG data from a simultaneous EEG-fMRI setup.

example:

```
#receive RDA data from a local computer with ip: 192.168.1.200
 rda_node = io.RdaReceive(rdaport=52136,
                          offset=.125,
                          host='192.168.1.200')
```

Last but not least, data can be also be streamed via the UDP protocol by using the "UdpSend" node.

3.2.2. Filter

The filter module is responsible for the real-time signal filtering. Nodes are implemented using signal processing functions from the SciPy[5] ecosystem. Specifically, a butterworth filter ("ButterFilter") for bandpass filtering, specifying the lowest and highest frequency cut in Hz and the order of the filter. A notch filter ("NotchFilter"), or band-stop filter which rejects a narrow frequency band (usually from power-line noise at 50 or 60 Hz) and leaves the rest of the spectrum little changed. Finally, a down-sampling Node ("DownSample") for changing the sample rate of the signal.

---

[5]   https://www.scipy.org/

example:

```
#create a bandpass filter between 1−40Hz
bandpass_node = filter.ButterFilter(input_port=port.output,
                                    lowcut=1,
                                    highcut=40,
                                    order=4)
```

3.2.3. Processing

The Processing module includes Nodes for further signal processing and transformations. The "PsdWelch" Node is for Spectral Analysis using Welch's Method, the "Fft" Node for Fast Fourier Transform and "HilbertTransform" Node for computing the analytic signal using the Hilbert transform.

example:

```
#Spectral Analysis of the bandpass_node
psd_node = processing.PsdWelch(bandpass_node.output)
```

3.2.4. Generate

The Generate module hosts Node for artificial or synthetic data generation. The "Generator" Node is instantiated with one of three modes: 'random' for random time-series generation, 'oscillator' for sinusoidal signal generation, and 'simulation' for simulated EEG data. For every "Generator" node, the number of channels and the sampling frequency needs also to be specified.

3.2.5. Read

Read module hosts Nodes that load and replay data. The "Reader" Node is able to parse and replay data and markers in real-time. Specifically, it can replay EEGLAB set files (.set), General Data Format (.gdf), Extensible Data Format (.xdf), and Brain vision format (.vhdr).

3.2.6. Select

The Select module is responsible for data selection and spatial filtering. Specifically, the "ChannelSelector" Node allows to select outgoing signal data to a subset of incoming data based on a list of channels. Channels may be identified by their index (e.g. 1,5,8) or their name (e.g. Cz, Fp1, P3). The "SpatialFilter" Node maps M inputs to N outputs by multiplying the each input vector with a matrix of coefficients. The "ReferenceChannel" Node subtracts the value of the reference channel from all other channels while the "CommonAverageReference" Node is subtracting from each sample the average value of the samples of all electrodes.

3.2.7. Function

The Function module hosts the "ApplyFunction" Node which can take as an input a python function and can pass a row or a numpy array (of shape number of input channels) for further processing inside a custom function.

example:

```
#function Node for subtracting 4 on every chunk of every channel
    def subfour(x):
        return x − 4
```

```
applyfunction_node = ApplyFunction(port.output, subfour)
```

3.2.8. Epoching

The Epoching module hosts Nodes that are responsible for slicing (or epoching) the signal into chunks of a desired length following either a stimulation event or a specific time interval. Specifically, the "MarkerBasedEpoching" Node cuts a continuous signal each time a Marker appears without a specific name unlike the "StimulationBasedEpoching" Node which cuts the signal into chunks of a desired length of specific stimulation events (or stimulation codes). Last but not least, the "TimeBasedEpoching" Node cuts the signal in a specific time intervals and for a specific length.

3.2.9. Epoch function

The Epoch Function module hosts Nodes that only process epoched data and not continuous. These Nodes can only be placed after an epoching node. The "UnivariateStat" Node computes the 'mean', 'median', 'min', 'max', 'range', 'std', 'quantile', 'iqr' on each incoming epoched signal. The "windowing" Node applies a windowing function between: 'blackman', 'hanning', 'hamming', 'triangular' on each incoming epoched signal.

Table 1. Core modules and nodes

Module	Nodes
classify.py	Classify [LDA 'class' or 'probability']
display.py	Plot, Graz, PlotSpectrum
epoch_function.py	UnivariateStat, Windowing['blackman', 'hanning', 'hamming', 'triang']
epoching.py	TimeBasedEpoching, MarkerBasedEpoching, StimulationBasedEpoching
feature.py	FeatureAggregator
filter.py	ButterFilter, NotchFilter, DownSample
function.py	ApplyFunction
generate.py	Generator ['random', 'oscillator', 'simulation']
io.py	LslSend, LslReceive, RdaReceive ['signal', 'marker'], UdpSend
log.py	Hdf5, Mat
processing.py	PsdWelch, FFT, Hilbert
read.py	Reader ['.xdf', '.gdf', '.set', '.vhdr']:
select.py	ChannelSelector, SpatialFilter, ReferenceChannel, CommonAverageReference
stimulator.py	Stimulator
store.py	ToCsv

3.2.10. Display

The Display module hosts Nodes responsible for signal visualization and feedback for stimulus markers. The "Plot" Node is visualizing signal streams and time series. The "PlotSpectrum" Node is for visualizing a spectrum signal in a matplotlib[6] window. Finally, the "Graz" Node is used for rendering feedback related to the Graz BCI paradigm used for Motor Imagery BCI experiments[30]. The "Graz" Node is using the Tk GUI toolkit[7] for rendering the stimulus as received through a stimulator node.

---

[6] https://matplotlib.org/
[7] https://docs.python.org/3/library/tkinter.html

3.2.11. Stimulator

The Stimulator module is hosting Nodes responsible for generating stimulations based on experimental design parameters. The "Stimulator" Node is generating a marker stream parsed from an external configuration file in xml format.

3.2.12. Store and Log

The Store module is for hosting Nodes responsible for data storage but not for large amounts of data like EEG datasets. Currently, under the Store module, logging with comma separated values is supported through the "ToCsv" Node for logging low resolution time-series, events or markers. For large datasets, the Log module hosts the "Hdf5" and "Mat" Nodes corresponding to the Hierarchical Data Format (HDF) and the Matlab matfile format respectively.

3.2.13. Feature

The Feature module is hosting Nodes responsible for creating feature vectors. The "FeatureAggregator" Node is taking each chunk of input and catenated into one feature vector together with class labels that can be used for classification.

3.2.14. Classification

The Classification module hosts Nodes responsible for model training and data classification. Currently the "Classify" Node is loading a trained model from joblib[8] pipelining and assigns to the input vector either a 'class' or 'probability' value.

3.3. Pipeline example

NeuXus loads and executes a .py file (called the pipeline script) at the beginning of its execution, this file describes the pipeline. It initializes all Nodes and Ports between them.

```python
#This pipeline sends by LSL a simple DSP feedback calculated in real-time

from neuxus.Nodes import *

def square(x):
    # x is a matrix of a row
    return x ** 2

#Node 1: receive LSL stream
lsl_reception = io.LslReceive(
    prop='type',
    value='EEG',
    data_type='signal'
)
#Node 2: bandpass filtering between 8-12hz
butter_filter = filter.ButterFilter(
    input_port=lsl_reception.output,
```

---

[8] https://pypi.org/project/joblib/

```
    lowcut=8,
    highcut=12
)
#Node 3: square the signal from the square(x) function
apply_function = function.ApplyFunction(
    input_port=butter_filter.output,
    function=square
)
#Node 4: epoch 1 sec of data every 0.5 sec
epoch = epoching.TimeBasedEpoching(
    input_port=apply_function.output,
    duration=1,
    interval=0.5
)
#Node 5: average signal
average = epoch_function.UnvariateStat(
    input_port=epoch.output,
    stat='mean'
)
#Node 6: send via LSL
lsl_send = io.LslSend(
    input_port=average.output,
    name='mySignalEpoched'
)
```

3.4. Extending NeuXus

NeuXus is a modular software allowing developers and researcher to implement third-party or custom nodes. NeuXus loads first the pipeline script. It then repeats indefinitely a loop that clears all Ports and updates each node. The *update* function of each Node is called at each iteration and get data from the input port. NeuXus pipeline is interrupted when 'Esc' is pressed. A Node has then 3 methods *init*, *update* and *terminate* (Figure 2).

A Node can have a port as input and/or output. Ports can share different type of data stored in Pandas DataFrame (table). A Port is an iteration object, it means that to get its values you need to iterate over it.

Type of Ports are:

- 'signal': 1 iteration = 1 chunk of the continuous signal
- 'epoch': 1 iteration = 1 epoch
- 'marker': 1 iteration = 1 marker
- 'spectrum': 1 iteration = 1 spectrum computed at one timestamp
- 'vector': 1 iteration = 1 vector

4. Discussion

NeuXus is an open-source and platform independent biosignal processing tool that attempts to balance usability with functionality. First, in terms of usability, NeuXus provides an abstract level of pipeline design through Nodes and Ports for real-time biosignal processing through its core functions. Secondly, its functionality can be extended with custom code by creating new Nodes in Python. Last but
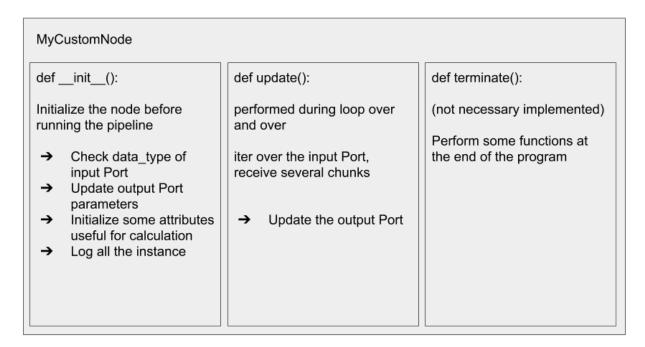
Figure 2. Custom Node architecture

not least, the processing pipelines of NeuXus can also exchange real-time data with external BCI tools through the supported networking protocols (e.g. LSL). Although interacting with NeuXus might not be as high-level as with visual programming tools (e.g. OpenVibe), nonetheless, and accounting for the trade-off between usability, functionality and extendability, NeuXus provides an easy-to-use interfacing with the Python language through the pipeline design.

In terms of performance, the speed of real-time processing is limited by the processing speed of each host that runs the pipeline. The processing frequency is self-regulated, hence, the software calculates all updates as quickly as possible and then waits for the next data entry. If the RAM is overloaded (because of other software or because there are too many Nodes in the pipeline), the loop takes longer to complete. This may involve a significant (and exponentially increasing) delay. If the stimuli is sent from outside the pipeline (e.g. via LSL), it is recommended to divide your pipeline, one with an independent stimulator, the other with the processing Nodes. This ensures that the pipeline does not slow down the sending of stimulation's via LSL. Further, by choosing Python, one loses execution speed compared to C++, nonetheless, each pipeline that can be designed with NeuXus can be considerably optimized on a case-by-case basis.

Last but not least, NeuXus is freely available through the Python Package Index (PyPI): https://pypi.org/project/neuxus/ and the package can be installed using the pip package installer: *pip install neuxus*. Finally, both the repository: https://github.com/LaSEEB/NeuXus and documentation: https://laseeb.github.io/NeuXus are available online.

Author Contributions: Conceptualization, A.V.; software, S.L. and A.V; validation, S.L. and A.V; writing–original draft preparation, A.V.; writing–review and editing, P.F.; supervision, P.F.; All authors have read and agreed to the published version of the manuscript.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wolpaw, J.R.; Birbaumer, N.; McFarland, D.J.; Pfurtscheller, G.; Vaughan, T.M. Brain–computer interfaces for communication and control. *Clinical neurophysiology* **2002**, *113*, 767–791.

2. da Silva, F.L. EEG: Origin and Measurement. In *EEG - fMRI: Physiological Basis, Technique, and Applications*; Mulert, C.; Lemieux, L., Eds.; Springer Berlin Heidelberg: Berlin, Heidelberg, 2010; pp. 19–38. doi:10.1007/978-3-540-87919-0_2.

3. Lotte, F.; Congedo, M.; Lécuyer, A.; Lamarche, F.; Arnaldi, B. A review of classification algorithms for EEG-based brain-computer interfaces. *J Neural Eng* **2007**, *4*, R1–R13. doi:10.1088/1741-2560/4/2/R01.

4. Pfurtscheller, G.; Neuper, C. Motor imagery and direct brain-computer communication. *Proceedings of the IEEE* **2001**, *89*, 1123–1134. doi:10.1109/5.939829.

5. Müller-Putz, G.R.; Scherer, R.; Brauneis, C.; Pfurtscheller, G. Steady-state visual evoked potential (SSVEP)-based communication: impact of harmonic frequency components. *Journal of Neural Engineering* **2005**, *2*, 123–130. doi:10.1088/1741-2560/2/4/008.

6. Sellers, E.W.; Arbel, Y.; Donchin, E. 12 BCIs that use P300 event-related potentials. *Brain-computer interfaces: principles and practice* **2012**, p. 215.

7. Kübler, A.; Furdea, A.; Halder, S.; Hammer, E.M.; Nijboer, F.; Kotchoubey, B. A Brain–Computer Interface Controlled Auditory Event-Related Potential (P300) Spelling System for Locked-In Patients. *Annals of the New York Academy of Sciences* **2009**, *1157*, 90–100. _eprint: https://nyaspubs.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1749-6632.2008.04122.x, doi:https://doi.org/10.1111/j.1749-6632.2008.04122.x.

8. Biasiucci, A.; Leeb, R.; Iturrate, I.; Perdikis, S.; Al-Khodairy, A.; Corbet, T.; Schnider, A.; Schmidlin, T.; Zhang, H.; Bassolino, M.; Viceic, D.; Vuadens, P.; Guggisberg, A.G.; Millán, J.d.R. Brain-actuated functional electrical stimulation elicits lasting arm motor recovery after stroke. *Nature Communications* **2018**, *9*, 2421. Number: 1 Publisher: Nature Publishing Group, doi:10.1038/s41467-018-04673-z.

9. Vourvopoulos, A.; Jorge, C.; Abreu, R.; Figueiredo, P.; Fernandes, J.C.; Bermúdez i Badia, S. Efficacy and Brain Imaging Correlates of an Immersive Motor Imagery BCI-Driven VR System for Upper Limb Motor Rehabilitation: A Clinical Case Report. *Frontiers in Human Neuroscience* **2019**, *13*, 244. doi:10.3389/fnhum.2019.00244.

10. Graimann, B.; Allison, B.Z.; Pfurtscheller, G. *Brain-computer interfaces: Revolutionizing human-computer interaction*; Springer Science & Business Media, 2010.

11. Vourvopoulos, A.; Niforatos, E.; Giannakos, M. EEGlass: An EEG-Eyeware Prototype for Ubiquitous Brain-Computer Interaction. Adjunct Proceedings of the 2019 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2019 ACM International Symposium on Wearable Computers; Association for Computing Machinery: New York, NY, USA, 2019; UbiComp/ISWC '19 Adjunct, p. 647–652. doi:10.1145/3341162.3348383.

12. Stegman, P.; Crawford, C.S.; Andujar, M.; Nijholt, A.; Gilbert, J.E. Brain–Computer Interface Software: A Review and Discussion. *IEEE Transactions on Human-Machine Systems* **2020**, *50*, 101–115. doi:10.1109/THMS.2020.2968411.

13. Brunner, P.; Bianchi, L.; Guger, C.; Cincotti, F.; Schalk, G. Current trends in hardware and software for brain–computer interfaces (BCIs). *Journal of Neural Engineering* **2011**, *8*, 025001. doi:10.1088/1741-2560/8/2/025001.

14. Brunner, C.; Andreoni, G.; Bianchi, L.; Blankertz, B.; Breitwieser, C.; Kanoh, S.; Kothe, C.A.; Lécuyer, A.; Makeig, S.; Mellinger, J.; Perego, P.; Renard, Y.; Schalk, G.; Susila, I.P.; Venthur, B.; Müller-Putz, G.R., BCI Software Platforms. In *Towards Practical Brain-Computer Interfaces: Bridging the Gap from Research to Real-World Applications*; Springer Berlin Heidelberg: Berlin, Heidelberg, 2013; pp. 303–331. doi:10.1007/978-3-642-29746-5_16.

15. Delorme, A.; Kothe, C.; Vankov, A.; Bigdely-Shamlo, N.; Oostenveld, R.; Zander, T.O.; Makeig, S., MATLAB-Based Tools for BCI Research. In *Brain-Computer Interfaces: Applying our Minds to Human-Computer Interaction*; Springer London: London, 2010; pp. 241–259. doi:10.1007/978-1-84996-272-8_14.

16. Delorme, A.; Makeig, S. EEGLAB: an open source toolbox for analysis of single-trial EEG dynamics including independent component analysis. *Journal of Neuroscience Methods* **2004**, *134*, 9 – 21. doi:https://doi.org/10.1016/j.jneumeth.2003.10.009.

17. Kothe, C.A.; Makeig, S. BCILAB: a platform for brain–computer interface development. *Journal of neural engineering* **2013**, *10*, 056014.

18. Oostenveld, R.; Fries, P.; Maris, E.; Schoffelen, J.M. FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data. *Computational Intelligence and Neuroscience* **2011**, *2011*, 9.

19. Tadel, F.; Baillet, S.; Mosher, J.C.; Pantazis, D.; Leahy, R.M. Brainstorm: A User-friendly Application for MEG/EEG Analysis. *Intell. Neuroscience* **2011**, *2011*, 8:1–8:13. doi:10.1155/2011/879716.

20. Pascual-Marqui, R.D. Standardized low-resolution brain electromagnetic tomography (sLORETA): technical details. *Methods and Findings in Experimental and Clinical Pharmacology* **2002**, *24 Suppl D*, 5–12.

21. Schalk, G.; McFarland, D.J.; Hinterberger, T.; Birbaumer, N.; Wolpaw, J.R. BCI2000: a general-purpose brain-computer interface (BCI) system. *IEEE Transactions on biomedical engineering* **2004**, *51*, 1034–1043.

22. Müller-Putz, G.R.; Breitwieser, C.; Tangermann, M.; Schreuder, M.; Tavella, M.; Leeb, R.; Cincotti, F.; Leotta, F.; Neuper, C. Tobi hybrid BCI: principle of a new assistive method. *International Journal of Bioelectromagnetism* **2011**, *13*, 144–145.

23. Schlögl, A.; Brunner, C. BioSig: a free and open source software library for BCI research. *Computer* **2008**, *41*, 44–50.

24. Renard, Y.; Lotte, F.; Gibert, G.; Congedo, M.; Maby, E.; Delannoy, V.; Bertrand, O.; Lécuyer, A. Openvibe: An open-source software platform to design, test, and use brain–computer interfaces in real and virtual environments. *Presence: teleoperators and virtual environments* **2010**, *19*, 35–53.

25. Gramfort, A.; Luessi, M.; Larson, E.; Engemann, D.; Strohmeier, D.; Brodbeck, C.; Goj, R.; Jas, M.; Brooks, T.; Parkkonen, L.; Hämäläinen, M. MEG and EEG data analysis with MNE-Python. *Frontiers in Neuroscience* **2013**, *7*, 267. doi:10.3389/fnins.2013.00267.

26. Venthur, B.; Blankertz, B. Mushu, a free- and open source BCI signal acquisition, written in Python. Engineering in Medicine and Biology Society (EMBC), 2012 Annual International Conference of the IEEE. IEEE, 2012, pp. 1786–1788. doi:10.1109/EMBC.2012.6346296.

27. Venthur, B.; Scholler, S.; Williamson, J.; Dähne, S.; Treder, M.S.; Kramarek, M.T.; Müller, K.R.; Blankertz, B. Pyff—A Pythonic Framework for Feedback Applications and Stimulus Presentation in Neuroscience. *Frontiers in Neuroinformatics* **2010**, *4*. doi:10.3389/fninf.2010.00100.

28. Tayeb, Z.; Waniek, N.; Fedjaev, J.; Ghaboosi, N.; Rychly, L.; Widderich, C.; Richter, C.; Braun, J.; Saveriano, M.; Cheng, G.; Conradt, J. Gumpy: a Python toolbox suitable for hybrid brain–computer interfaces. *Journal of Neural Engineering* **2018**, *15*, 065003. Publisher: IOP Publishing, doi:10.1088/1741-2552/aae186.

29. Memmott, T.; Koçanaoğulları, A.; Lawhead, M.; Klee, D.; Dudy, S.; Fried-Oken, M.; Oken, B. BciPy: Brain-Computer Interface Software in Python, 2020, [arXiv:cs.HC/2002.06642].

30. Pfurtscheller, G.; Neuper, C. Motor imagery and direct brain-computer communication. *Proceedings of the IEEE* **2001**, *89*, 1123–1134.