Proceedings of the
8th Graz Brain-Computer Interface Conference 2019 DOI: 10.3217/978-3-85125-682-6-17

TIMEFLUX: AN OPEN-SOURCE FRAMEWORK FOR THE ACQUISITION AND NEAR REAL-TIME PROCESSING OF SIGNAL STREAMS

P. Clisson[1], R. Bertrand-Lalo[2], M. Congedo[3], G. Victor-Thomas[2], J. Chatel-Goldman[2]

[1]Independent researcher, Paris, France
[2]Open Mind Innovation, Paris, France
[3]GIPSA-lab, University Grenoble Alpes, CNRS, Grenoble-INP, Grenoble, France

E-mail: pierre@clisson.net

ABSTRACT: *Timeflux* is an open-source framework for the acquisition and near real-time processing of signal streams. It can be used to build any kind of brain-computer interface and neurofeedback application, allowing high performance for both quick prototyping and final product development. *Timeflux* has been designed to be lightweight, easy to use, scalable, and extensible. It equally suits researchers, companies and neurotechnology enthusiasts. *Timeflux* is developed in Python and released under the permissive and flexible MIT license. It is available at https://timeflux.io.

INTRODUCTION

Research on brain-computer interfaces (BCIs) has been carried out for about half a century, but has been experiencing momentum only recently [1]. It is now attracting researchers—not only in neurosciences, but also in diverse disciplines such as engineering and informatics—as well as private companies and individuals interested in exploring the frontiers of information technology such as hackers [2]. For a long time, advances in BCIs have been hindered by a lack of standardization of data acquisition protocols, data format and data processing. The poor availability of large public databases is another factor that has slowed progress [3]. In Europe, the necessity of introducing standards and sharing data has been pointed out by several consortia of European projects. For instance, the integrative project BNCI Horizon 2020 has created and promoted the first comprehensive repository of public BCI databases [4]. Coupling with this, the Mother of All BCI Benchmarks (MOABB) [5] allows to test the accuracy of BCI decoders on all available databases in an objective and easily reproducible way (*e.g.*, see [6] in these proceedings).

Another factor slowing the progress in BCI research is the lack of a *standard framework for online data acquisition and processing*, that is, the software at the very heart of actual BCI systems. For such a framework several characteristics are desirable: it should be easy-to-use, lightweight, efficient, scalable and extensible. Moreover, it should encourage exploration, allowing quick prototyping and testing as well as facilitating the integration with relevant existing code libraries and external tools, without sacrificing adequacy with the development of reliable products for the public. In this way, it may suit all actors of BCI research and development, both within academic institutions and private companies, allowing global collaboration.

Several frameworks for acquisition and real-time processing of signal streams have been developed (Tab. 1), however none of them satisfy all these criteria. Existing frameworks are either domain-specific—that is, they apply only to specific neuroimaging modalities— or are difficult to extend, or else are made available under a restrictive license that does not favor at the same time free and commercial development. For these reasons we hereby present a new framework, named *Timeflux*.

Timeflux has been developed in Python, which has become the programming language of choice in the data science and machine learning communities, thus already offering a wide panel of code libraries. Python is an extensible and cross-platform language, characteristics that have proved essential for its massive adoption. *Timeflux* is open-source and is released under the MIT license, which is very flexible. It is not specific to neuroimaging or bioelectrical signals, thus it can be used in the more general context of IoT, as well as in geoscience, control engineering, algorithmic trading, and more.

DESIGN PRINCIPLES

Based on the considerations above, the following set of rules has guided the development of *Timeflux*:

Simplicity of use: *The framework should be easy to learn, to use, and to extend.* This implies a clear documentation, a simple descriptive syntax for pipelines, and a minimal effort requirement for writing new plugins.

Lightness: *In order to ensure stability and frictionless maintenance, the core should maintain a small footprint.* Only the essential features belong to the core, everything else is moved to plugins. This leads to a codebase that is easy to apprehend.

Agnosticism: *The user should not be locked into a specific paradigm or set of tools.* From acquisition through processing to recording, every step is configurable and replaceable.

 DOI: 10.3217/978-3-85125-682-6-17

Extensibility: *The architecture should be based on plugins.* This allows custom node development, integration with specialized libraries and hardware, and encourages external contributions without depending on base code merges. In *Timeflux*, plugins are ordinary Python modules, with only one required method to implement.

Reusability: *Resources should not be wasted by rewriting common algorithms and structures. Timeflux* relies on industry standards such as SciPy [7] and Numpy [8] for scientific computing, Pandas [9] for tabular data, Xarray [10] for multidimensional data, Scikit-learn [11] for machine learning, and NetworkX [12] for graph processing. These tools are well maintained, and many scientists are already familiar with them.

Scalability: *Parallelism and concurrency should be automatically enforced when applicable. Timeflux* distributes computing across CPU cores, threads, and even hosts.

Efficiency: *High-level interfacing should not sacrifice performance.* Sensible choices and careful memory management allow near real-time processing.

THEORY OF OPERATION

Timeflux is a framework to create pipelines, called *applications*. It does not matter if they are mere acquisition units, signal processing prototypes or complex BCI and biofeedback solutions. Applications are defined by a set of processing steps, called *nodes*, which are linked together using a simple YAML syntax.

In order to be valid, an application must satisfy the requirements of a directed acyclic graph (DAG) [13, 14], that is, a set of nodes connected by edges, where information flows in a given direction, with no internal loop (Fig. 1). Multiple DAGs are authorized within the same application, optionally communicating with each other using one of the available network protocols. DAGs run simultaneously at their own adjustable rate. Within each DAG, nodes are executed sequentially according to the topological sorting of the graph [15].

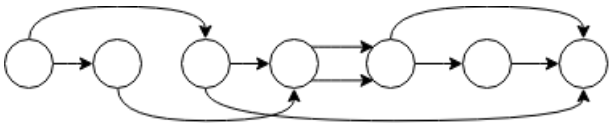

Figure 1: An example directed acyclic graph (DAG), arranged in topological order. Circles represent *nodes*, arrows are *edges*. The connection points between nodes and edges are called *ports*. Information flows from left to right, at a frequency defined by the graph rate.

Nodes may expect one or more inputs and may provide any number of outputs. These I/O, called *ports*, have two main properties. The *data* property is either a datetime-indexed Pandas DataFrame, or an Xarray structure with at least two dimensions: *time* and *space*. The *meta* property is a dictionary containing arbitrary keys, which can be used for example to declare a stream rate or to describe the context associated with an event.

AVAILABLE NODES

Timeflux comes with a growing collection of nodes, either available in core or as plugins.

LSL: The Lab Streaming Layer [16] is a transport and synchronization library compatible with a large range of EEG equipment. *Timeflux* is able to handle both input and output streams.

Publish/Subscribe: The publish/subscribe pattern allows asynchronous messaging between DAGs and/or between external components. Subscribers express interest in topics, and receive data matching these topics. There can be more than one publisher per topic. This protocol is implemented using the ZeroMQ library [17].

OSC: The Open Sound Control protocol [18] is commonly used in media applications. It is useful to create rich interactive environments.

Epoching and windowing: Several nodes to extract epochs or to accumulate data from streams are included.

HDF5: The Hierarchical Data Format [19] is a stable and powerful data storage and query solution. Nodes for recording and playback are incorporated.

Queries and expressions: A small set of nodes are available to extract data matching specific criteria and to execute arbitrary arithmetic operations.

Web User Interface: Visualizing data and sending events is possible directly within a browser. Fig. 2 is a screenshot of a typical session. The current implementation uses the HTML5 canvas object, which runs in the main thread. A new version taking advantage of the WebGL technology is underway. Preliminary tests show that over a million points can be plotted per second.

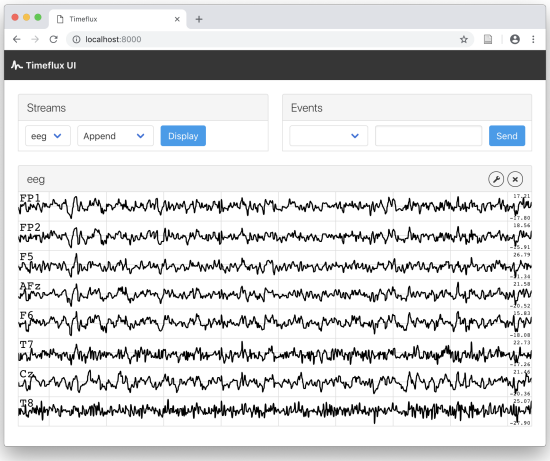

Figure 2: Data stream visualization in a web browser (electroencephalographic data in this example).

Digital Signal Processing: Common signal processing algorithms are already available (for instance, filtering, FFT, covariance matrix estimation, etc.).

Dejittering: A signal periodicity can deviate from its nominal rate, or data may be acquired in chunks. Several mitigating methods are provided.

 DOI: 10.3217/978-3-85125-682-6-17

Machine Learning: Models can be fitted on epoched data, and then used to classify or transform new data. This is made possible by the scikit-learn library.

Branches: The same processing pipelines are often reused from project to project. *Timeflux* will provide a method to define sub-graphs as simple nodes. At the time of writing, this is still a work in progress.

Device drivers: Plugins for direct integration with popular open-source hardware, such as OpenBCI [20] and BITalino [21], as well as proprietary hardware, have been developed and more will be released in the upcoming months.

PERFORMANCES

Good practices: Python is a high-level programming language. While this offers several advantages, it also implies some trade-offs in terms of efficiency. This issue is mitigated in several ways. Special attention is paid to avoid memory copy unless absolutely required. When dealing with numpy-based structures, vectorized functions (written in C) are used whenever possible. At the very least, the Pandas `apply()` function benefits from low-level optimization. These simple measures help maintaining good performances for most applications. In extreme cases, it is possible to speed execution time by rewriting a node's critical functions in Cython [22] or by using Numba [23].

Architecture: *Timeflux* takes advantage of modern CPU architecture and distributes the execution of DAGs across cores. Nodes that require an infinite loop (*i.e.*, device drivers) run parts of their code into a separate thread, so the whole graph is not penalized. Demanding applications can further be optimized by running synchronized *Timeflux* instances on multiple computers.

Real-time: Hard real-time is when missed deadlines are unacceptable and result in system failure [24]. In most cases, hard real-time is not required, and soft real-time (also known as near real-time) is sufficient. *Timeflux* allows a few tens of milliseconds latency (the latter being modulated by the graph rate), as long as the incoming data is timestamped at the point of origin. Time offset correction is achieved either by using a dedicated communication protocol such as LSL, or by repeatedly synchronizing *Timeflux* instances using an algorithm similar to the Network Time Protocol [25], thus ensuring sub-millisecond precision.

EXAMPLE

As an example we illustrate a simple alpha-neurofeedback application, which is schematically represented in Fig. 3. The application consists of three graphs. In the main one, we assume that the EEG data is acquired through a LSL inlet. Data is accumulated into a rolling window, on which the classical Welch's method is applied with default parameters [26]. The frequency bands are then extracted from the periodogram. Finally,

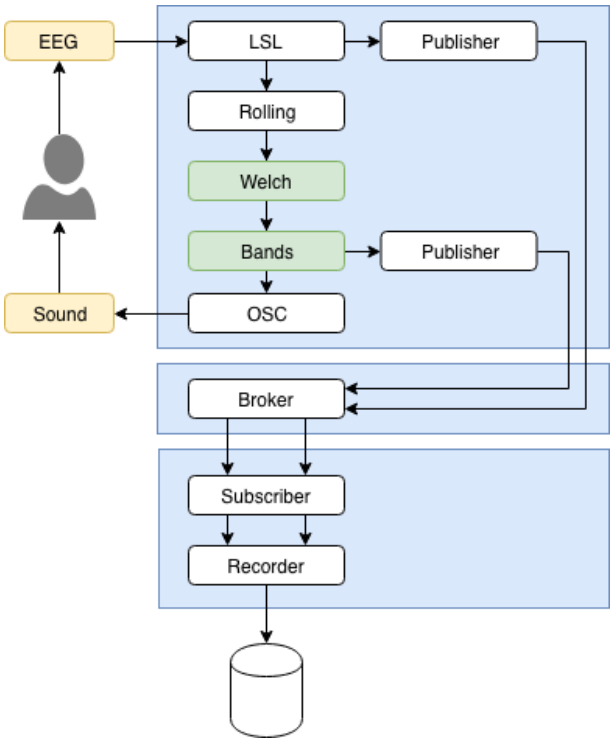

Figure 3: Schematic representation of a basic neurofeedback application. The three blue boxes constitute the *Timeflux* application. The white boxes are core nodes. The green boxes are plugin nodes. Yellow boxes indicate external components.

the relative alpha power is sent to an OSC outlet. An external application receives this data and plays a sound when the feedback signal crosses a defined threshold. The other two graphs are not strictly required, but illustrate some important principles. The graph containing the `Broker` node acts as a proxy. It receives data from publishers (in our example, the raw EEG stream and the computed frequency bands) and redistributes it to subscribers. In the last graph, these two data streams are aggregated and saved to a HDF5 file.
The whole application is expressed in YAML as follows:

```yaml
graphs:

  # The publish/subscribe broker graph
  - id: PubSubBroker
    nodes:
    # Allow communication between graphs
    - id: Broker
      module: timeflux.nodes.zmq
      class: Broker

  # The main processing graph
  - id: Processing
    nodes:
    # Receive EEG signal from the network
    - id: LSL
      module: timeflux.nodes.lsl
      class: Receive
      params:
        name: signal
    # Continuously buffer the signal
    - id: Rolling
      module: timeflux.nodes.window
      class: Window
      params:
        length: 1.5
        step: 0.5
    # Compute the power spectral density
    - id: Welch
```

 DOI: 10.3217/978-3-85125-682-6-17

```
    module: timeflux_dsp.nodes.spectral
    class: Welch
  # Average the power over band frequencies
  - id: Bands
    module: timeflux_dsp.nodes.spectral
    class: Bands
  # Send to an external application
  - id: OSC
    module: timeflux.nodes.osc
    class: Client
    params:
      address: /alpha
  # Publish the raw EEG signal
  - id: PublisherRaw
    module: timeflux.nodes.zmq
    class: Pub
    params:
      topic: raw
  # Publish the frequency bands
  - id: PublisherBands
    module: timeflux.nodes.zmq
    class: Pub
    params:
      topic: bands
  # Connect nodes
  edges:
    - source: LSL
      target: Rolling
    - source: Rolling
      target: Welch
    - source: Welch
      target: Bands
    - source: Bands:alpha
      target: OSC
    - source: LSL
      target: PublisherRaw
    - source: Bands
      target: PublisherBands
  # Run this graph 25 times per second
  rate: 25

# The recorder graph
- id: SaveToFile
  nodes:
  # Receive data streams from the broker
  - id: Subscriber
    module: timeflux.nodes.zmq
    class: Sub
    params:
      topics:
      - raw
      - bands
  # Record to file
  - id: Recorder
    module: timeflux.nodes.hdf5
    class: Save
  # Connect nodes
  edges:
    - source: Subscriber:raw
      target: Recorder:eeg_raw
    - source: Subscriber:bands
      target: Recorder:eeg_bands
  # Update file every second
  rate: 1
```

COMPARISON WITH SIMILAR SOFTWARE

Several frameworks are currently available for the online analysis of bioelectrical signals (Tab. 1). Choosing one is a difficult task since it often requires a balance among a variety of criteria, such as desired characteristics, performances, extensibility, license, and ease of use. The characteristics are generally oriented towards specific applications (*e.g.*, BCI using electroencephalography) thus only a few frameworks are versatile enough to be used with heterogeneous time series and in more general contexts. The performance is hard to evaluate without a comprehensive benchmark using comparable objective metrics and a thorough knowledge of the software inner functioning. Regarding extensibility, all open-source solutions are extensible by nature, but the effort required to do so varies according to the architecture and coding experience. Concerning the license, while commercial licensing may incur financial costs and vendor lock-ins, GPL-derived licenses can in some cases also be treacherous since they require derivative works to be distributed under the same copyleft license. The MIT and BSD licenses—and to some extent, the Apache licence—are much more permissive instead. *In fine*, much of these criteria are intrinsically subjective or difficult to evaluate objectively.

Timeflux is a modern framework for real-time processing of signal streams. According to the aforementioned principles we have followed for development, it brings a well-balanced set of features, delivered under a very flexible license.

DISCUSSION

Although *Timeflux* is mature enough to be used in production, there is still room for improvement. Our roadmap foresees significant updates in the following areas:

Performances: Extensive benchmarks will be systematized to identify bottlenecks.

Scalability: For large projects involving many instances spreading over multiple hosts, mechanisms allowing automatic discovery, load-balancing, fail-over, and seamless synchronization (possibly using the Precision Time Protocol [50]), will be implemented.

General usage: We aim for a comprehensive documentation and better development tools. We also intend to provide turnkey BCI paradigms to accelerate prototyping.

User Interface: The web interface is currently being rewritten to significantly increase the number of points that can be plotted per second, and to introduce specialized widgets. In the future, it will be possible to design pipelines directly from a browser. A JavaScript API for stimulation presentation will also be available.

Integration: Nodes are being developed to integrate with more data acquisition devices, stimulation presentation software, and alternative storage solutions.

CONCLUSION

We have presented *Timeflux*, an open, highly flexible and actively developed solution that is meant to accelerate the creation of applications and standardize them. It is naturally suitable for BCI and biofeedback applications.

ACKNOWLEDGEMENT

This project has been supported by Open Mind Innovation.

 DOI: 10.3217/978-3-85125-682-6-17

Table 1: Comparable frameworks

Name	Release year	Language	License	Main domain
BCI++ [27]	2008	C/C++	GPL	Closed loop neuroscience
BCI2000 [28]	2001	C++	GPL	Closed loop neuroscience
BCILAB [29, 30]	2010	Matlab	GPLv2	Closed loop neuroscience
BioEra [31]	2004	Java	Commercial	Closed loop neuroscience
BioSig [32, 33]	2003	C++	GPLv3	Closed loop neuroscience
BrainBay [34]	2014	Python	GPL	Closed loop neuroscience
CloudBrain [35]	2007	Python	AGPLv3	Acquisition
Falcon [36]	2017	C++	GPLv3	Closed loop neuroscience
Fieldtrip [37]	2003	Matlab	GPLv2	Closed loop neuroscience
Gumpy [38, 39]	2018	Python	MIT	Closed loop neuroscience
Midas [40]	2014	Python	MIT	Barebone
Neuromore [41]	2015	C++	Commercial	Closed loop neuroscience
Neuropype [42]	2014	Python	Commercial	Closed loop neuroscience
Nipype [43]	2010	Python	Apache	Neuroimaging
Open Ephys GUI [44, 45]	2011	C++	GPLv3	Extracellular electrophysiology
OpenVibe [46, 47]	2009	C++	AGPLv3	Closed loop neuroscience
PyAcq [48]	2015	Python	BSD 3-clause	Acquisition
Timeflux [49]	2019	Python	MIT	Generic time series

REFERENCES

[1] Gartner. *Gartner identifies five emerging technology trends that will blur the lines between human and machine*. 2018. URL: https://gtnr.it/2N2X0QE (visited on 02/01/2019).

[2] NeuroTechX. *The international community for neurotech enthusiasts*. URL: https://neurotechx.com (visited on 02/01/2019).

[3] I. Obeid and J. Picone. "Bringing Big Data to neural Interfaces". In: *Proc. Fifth Int. BCI Meeting*. Pacific Grove (CA), USA, 2013, p. 180.

[4] B. H. 2020. *Databases*. URL: http://bnci-horizon-2020.eu/database/data-sets (visited on 02/08/2019).

[5] V. Jayaram and A. Barachant. "MOABB: trustworthy algorithm benchmarking for BCIs". In: *Journal of Neural Engineering* 15.6 (2018), p. 066011.

[6] M. Congedo, P. Rodrigues, and C. Jutten. "The Riemannian Minimum Distance to Means field Classifier". In: *Proc. 8th Int. Graz BCI Conference*. Ed. by TUG. Graz, Austria, 2019.

[7] E. Jones, T. Oliphant, P. Peterson, et al. *SciPy: Open source scientific tools for Python*. 2001–. URL: https://www.scipy.org (visited on 02/01/2019).

[8] T. Oliphant. *Guide to NumPy*. 2nd. USA: CreateSpace Independent Publishing Platform, 2015.

[9] W. McKinney. "Data Structures for Statistical Computing in Python". In: *Proceedings of the 9th Python in Science Conference*. Ed. by S. van der Walt and J. Millman. 2010, pp. 51–56.

[10] S. Hoyer and J. J. Hamman. "xarray: N-D labeled Arrays and Datasets in Python". In: *Journal of Open Research Software* 5 (2017).

[11] F. Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *J. Mach. Learn. Res.* 12 (Nov. 2011), pp. 2825–2830.

[12] A. A. Hagberg, D. A. Schult, and P. J. Swart. "Exploring Network Structure, Dynamics, and Function using NetworkX". In: *Proceedings of the 7th Python in Science Conference*. Ed. by G. Varoquaux, T. Vaught, and J. Millman. Pasadena, CA USA, 2008, pp. 11 –15.

[13] N. Christofides. *Graph Theory: An Algorithmic Approach (Computer Science and Applied Mathematics)*. Orlando, FL, USA: Academic Press, Inc., 1975.

[14] K. Thulasiraman and M. N. S. Swamy. *Graphs: theory and algorithms*. New York: Wiley, 1992. Chap. 5.7 Acyclic Directed Graphs.

[15] U. Manber. *Introduction to Algorithms: A Creative Approach*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989. Chap. 7.4 Topological Sorting.

[16] C. Kothe. *Lab Streaming Layer*. 2013. URL: https://github.com/sccn/labstreaminglayer (visited on 02/01/2019).

[17] *Distributed Messaging*. URL: http://zeromq.org/ (visited on 02/01/2019).

[18] M. Wright. *Open Sound Control: an Enabling Encoding for Media Applications*. URL: http://opensoundcontrol.org (visited on 02/01/2019).

[19] T. H. Group. *Hierarchical Data Format*. URL: https://www.hdfgroup.org/ (visited on 02/01/2019).

[20] *Open Source Biosensing Tools (EEG, EMG, EKG, and more)*. URL: https://openbci.com/ (visited on 02/01/2019).

[21] *BITalino*. URL: https://bitalino.com/ (visited on 02/01/2019).

[22] S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn, and K. Smith. "Cython: The Best of Both Worlds". In: *Computing in Science Engineering* 13.2 (2011), pp. 31 –39.

[23] S. K. Lam, A. Pitrou, and S. Seibert. "Numba: A LLVM-based Python JIT Compiler". In: *Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC*. LLVM '15. ACM, 2015, 7:1–7:6.

[24] M. Ben-Ari. *Principles of concurrent and distributed programming*. Prentice Hall international series in computer science. Prentice Hall, 1990. Chap. 16, p. 164.

[25] J. Martin, J. Burbank, W. Kasch, and P. D. L. Mills. *Network Time Protocol Version 4: Protocol and Algorithms Specification*. RFC 5905. June 2010.

[26] P. Welch. "The use of fast Fourier transform for the estimation of power spectra: A method based on time averaging over short, modified periodograms". In: *IEEE Transactions on Audio and Electroacoustics* 15.2 (1967), 70–73.

[27] L. Maggi, S. Parini, P. Perego, and G. Andreoni. "BCI++: an object-oriented BCI Prototyping Framework". In: *Proceedings of the 4th International Brain-Computer Interface Workshop and Training Course* (Jan. 2008).

[28] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw. "BCI2000: a general-purpose brain-computer interface (BCI) system". In: *IEEE Transactions on Biomedical Engineering* 51.6 (2004), 1034–1043.

[29] C. A. Kothe and S. Makeig. "BCILAB: a platform for brain–computer interface development". In: *Journal of Neural Engineering* 10.5 (2013), p. 056014.

[30] T. S. C. for Computational Neuroscience (SCCN. *Open Source Matlab Toolbox for Brain-Computer Interface research.* URL: https://sccn.ucsd.edu/wiki/BCILAB (visited on 02/01/2019).

[31] BioEra. *Welcome to the Home of Bioera!* URL: http://www.bioera.net (visited on 02/01/2019).

[32] C. Vidaurre, T. H. Sander, and A. Schlögl. "BioSig: The Free and Open Source Software Library for Biomedical Signal Processing". In: *Computational Intelligence and Neuroscience* 2011 (2011), 1–12.

[33] BioSig. *The BioSig Project*. URL: http://biosig.sourceforge.net (visited on 02/01/2019).

[34] BrainBay. *BrainBay - an OpenSource Biosignal project*. URL: http://www.shifz.org/brainbay (visited on 02/01/2019).

[35] CloudBrain. *Open-source platform for wearable data analytics.* URL: http://getcloudbrain.com (visited on 02/01/2019).

[36] D. Ciliberti and F. Kloosterman. "Falcon: a highly flexible open-source software for closed-loop neuroscience". In: *Journal of Neural Engineering* 14.4 (2017), p. 045004.

[37] FieldTrip. *Welcome to the FieldTrip website*. URL: http://www.fieldtriptoolbox.org (visited on 02/01/2019).

[38] Z. Tayeb et al. "Gumpy: a Python toolbox suitable for hybrid brain–computer interfaces". In: *Journal of Neural Engineering* 15.6 (2018), p. 065003.

[39] Z. Tayeb et al. *Gumpy: A Toolbox Suitable for Hybrid Brain-Computer Interfaces*. URL: http://gumpy.org (visited on 02/01/2019).

[40] A. Henelius and J. Torniainen. "MIDAS: Open-source framework for distributed online analysis of data streams". In: *SoftwareX* 7 (2018), 156–161.

[41] Neuromore. *Neuromore Studio*. URL: https://www.neuromore.com (visited on 02/01/2019).

[42] Intheon. *NeuroPype: Advanced biosignal processing made simple*. URL: https://www.neuropype.io (visited on 02/01/2019).

[43] Nypipe. *Neuroimaging in Python: Pipelines and Interfaces*. URL: https://nipype.readthedocs.io (visited on 02/01/2019).

[44] J. H. Siegle, A. C. López, Y. A. Patel, K. Abramov, S. Ohayon, and J. Voigts. "Open Ephys: an open-source, plugin-based platform for multichannel electrophysiology". In: *Journal of Neural Engineering* 14.4 (2017), p. 045003.

[45] OpenEphys. *open-source electrophysiology*. URL: http://www.open-ephys.org (visited on 02/01/2019).

[46] Y. Renard et al. "OpenViBE: An Open-Source Software Platform to Design, Test, and Use Brain–Computer Interfaces in Real and Virtual Environments". In: *Presence* 19.1 (2010), 35–53.

[47] INRIA. *OpenVIBE Software for Brain Computer Interfaces and Real Time Neurosciences*. URL: http://openvibe.inria.fr (visited on 02/01/2019).

[48] Pyacq. *Pyacq*. URL: https://github.com/pyacq (visited on 02/01/2019).

[49] P. Clisson et al. *Timeflux*. 2018–. URL: https://timeflux.io.

[50] IEEE Standards Association. *IEEE 1588-2008 - IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems*. URL: https://standards.ieee.org/standard/1588-2008.html (visited on 02/01/2019).