OpenElectrophy: an electrophysiological data- and analysis-sharing framework

Samuel Garcia, Nicolas Fourcaud-Trocmé

OPENELECTROPHY: AN ELECTROPHYSIOLOGICAL DATA- AND ANALYSIS-SHARING FRAMEWORK

Samuel Garcia and Nicolas Fourcaud-Trocmé
Neurosciences Sensorielles Comportement Cognition, CNRS UMR5020, Université Lyon 1
sgarcia@olfac.univ-lyon1.fr

ABSTRACT
Progress in experimental tools and design is allowing the acquisition of increasingly large datasets. Storage, manipulation and efficient analyses of such large amounts of data is now a primary issue. We present OpenElectrophy, an electrophysiological data and analysis sharing framework developed to fill this niche. Based on the object-oriented language Python, it stores all experiment data and meta-data in a single central SQL-type database. It also provides a graphic user interface to visualize and explore the data, and a library of functions for user analysis scripting in Python. It implements multiple spike sorting methods, and oscillation detection based on the ridge extraction method due to Roux et. al. [1]. OpenElectrophy is open-source and is freely available for download at http://neuralensemble.org/trac/OpenElectrophy.

KEY WORDS
Software, database, spike sorting, oscillation

1. Introduction

Recent developments in electrophysiology experimental techniques have lead to increases in the amount of data produced. It is now common to record continuous signals simultaneously from many electrodes with a sampling rate of 10 kHz or more. This increase in raw data flow has been accompanied by an increase in the complexity of the experimental protocol, meta-data management, and the subsequent analyses. Several commercial software products have been developed to tackle the increasing data management demands of state-of-the-art electrophysiology. However, as such commercial software products have not always evolved as rapidly as the needs of the field, several open source projects have appeared which are developed by the researcher community. Some aimed at performing very specific and well-known analysis based on a highly developed graphical user interface (GUI, mainly for time-frequency analyses), while other are simply function libraries which provides powerful and more flexible tools but not user friendly for electrophysiologists more concerned by the experimental part of their work (mainly in the spike detection domain). Moreover, none of the available software or toolboxes addresses the problem of how to simply and conjointly manipulate experimental data and meta-data.

OpenElectrophy [2] was designed more as a framework for data analysis than a piece of completely frozen analysis software. For example, it is not specific to a given type of electrophysiological signal, and does not directly perform a specific type of analysis at the request of a researcher with a "point-and-click" scheme. Rather, it provides tools to facilitate data storage, exploration and analysis script writing. It gathers the best of the two open source approaches described previously, both in terms of purpose (time-frequency analysis and spike sorting) and in terms of user interface (GUI and toolboxes). In addition, it includes generic tools for conjointly manipulating both experimental data and meta-data.
The project's main philosophy has three parts: first, for each experiment, the data and meta-data are all stored in a single central SQL-type database. This strategy allows for flexibility in mixing both types of data in the subsequent analyses. Second, it provides a GUI that is useful for exploring the data and detecting events of interest (oscillations or spikes). Third, it contains a library of "methods" (high-level functions) to aid in the writing of analysis scripts, both in the interfacing of these scripts with the database and in the manipulation of the data. This library is written with the open source object-oriented language Python which can be interfaced with many other languages (R, C, MATLAB, FORTRAN) and thus allows the reuse of previously written code.
This article presents the design and use of OpenElectrophy. It is organized into two sections. We first compare OpenElectrophy to similar projects and detail the advantages, drawbacks and differences of purpose for each project. Second, we present the OpenElectrophy technical choices, work flow and the general way in which it is used.

2. Comparisons with other projects and the main goals of OpenElectrophy

A detailed comparison of OpenElectrophy with similar projects has already been done [2]. We will give here only the main points and conclusion of this comparison.
First many commercial products exist for the analysis of electrophysiological signals and are in wide-spread use. Despite their high quality GUI and continuous development, these project use proprietary languages which prevent code-sharing and reuse; and have limited uptake of tools being developed by the scientific community. Besides, they store data with proprietary file formats whose specifications are generally not

available making long-term storage or sharing of data problematic since anyone who wants to access the data needs the right software. An attempt has been made to deal with this issue by the neuroshare initiative [3] but it provides only reading capabilities on windows platform.

On the open source side, we find two main project families: magneto- or electro-encephalography (MEG/EEG) signal oriented and spike analysis oriented.

In the MEG/EEG software family, we find MATLAB based projects with comprehensive GUI for non-progammer users and whose main capabilities include time-frequency analysis, analyses of event-related potentials, 3D plotting methods and source localization.

In the spike sorting software family, we have on one side numerous function libraries in various languages (R, C++, MATLAB) generally written to introduce a new spike sorting method and provide only limited GUIs. On the other side, some projects are a collection of scripts dedicated to the analysis of spike trains after spike sorting has already been completed.

Finally, there is only a few number of projects mixing both spike sorting and time-frequency analysis and none of them include any database framework or meta-data management.

One of the recent and closest project to OpenElectrophy is CARMEN [4], a consortium which aims to create and maintain a web portal to a virtual laboratory dedicated to the sharing of electrophysiological data and analysis scripts.

OpenElectrophy was written for several reasons:

- to have a project useful for all types of experiments, mixing time-frequency and spike analyses, compatible with large datasets produced in neuroscience labs and capable to reuse existing tools when available
- to have a project which includes different spike sorting methods and provide a simple way to compare them and add new ones
- to have a project that directly manage data and meta-data through a SQL-type database that allows sustainable storage and provide tools to exchange data written in other file formats
- to have a free project based solely on other free projects
- to have a project wich provides both a high quality GUI to explore the data and to make initial analysis steps like oscillation and spike detections, and also a library of Python classes and methods useful to write further analysis scripts

Finally, we must point out that at its current development stage, OpenElectrophy is primarily designed for the LFP and spike community rather than the multi-channel EEG community. For example, it does not currently include any advanced visualization tools, such as 3D scalp plot or source localization techniques.

3. Technical choices and workflow

3.1 Data integration

To work with OpenElectrophy, the first step consists in integrating data into the central database. OpenElectrophy includes a input-ouput (IO) module which can read most of the common file formats used to store electrophysiological data: Axon, EEGLab, ELAN, Elphy, micromed, NeuroExplorer, Plexon, PYNN, Spike2, WinWCP [5]. On Windows platform, OpenElectrophy can also use libraries available in Neuroshare and thus read from AlphaOmega and TDT file types [6]. Finally, OpenElectrophy can read standard ASCII files and includes an example module which generates complete sets of arbitrary signals useful to test the software or related analysis script.

While reading a set of files, OpenElectrophy stores all the available information in the central database. This database is an SQL-type database, either a MySQL server or a SQLite local file [7]. In both cases the interface with OpenElectrophy is done via the object-relational mapper SQLAlchemy (Python module) [8]. As brief reminder, a database system is a collection of tables which are collection of fields. Tables are linked to one another by indexes or keys. Putting data into a database is equivalent to splitting it up in an atomic way and organizing it into different tables. The logical or hierarchical organization between tables is not known a priori, but is formed while exploring the data, as opposed to file systems, which are organized into directories and sub-directories with a fixed organization. Thus, it is possible to have multiple views of the same database. This mechanism, while apparently basic, proves to be flexible and efficient. To work with this system, the user must learn structured query language (SQL). This language permits the user to reconstruct, filter and sort the data. The user can also add fields or tables at a later point without affecting previous work.

A crucial point is the design of the table's schema: the list of tables, and their contents and links. The database scheme used by OpenElectrophy is presented in figure 1. It has been designed in cooperation with developers of the Neurotools [9] project in order to facilitate the interaction between both projects. It has been designed to introduce generic notions of any electrophysiological experiments. In particular a central component of the scheme is the *segment* table which contains a coherent set of analog signals (typically electrode recordings) and discrete signals (like triggers or spikes). The *recordingpoint* table links signals recorded from the same electrode accross segments while the *block* table group a set of segments and their associated recording points. Finally *spike, spiketrain, neuron* and *oscillation* tables store events of interest after their detection.

Interestingly, the database scheme is not fixed and can be adapted to the need of the user by adding other tables or fields. Thanks to the ORM, these new fields are visible from OpenElectrophy and can be edited or used to select data appropriately. However, while importing

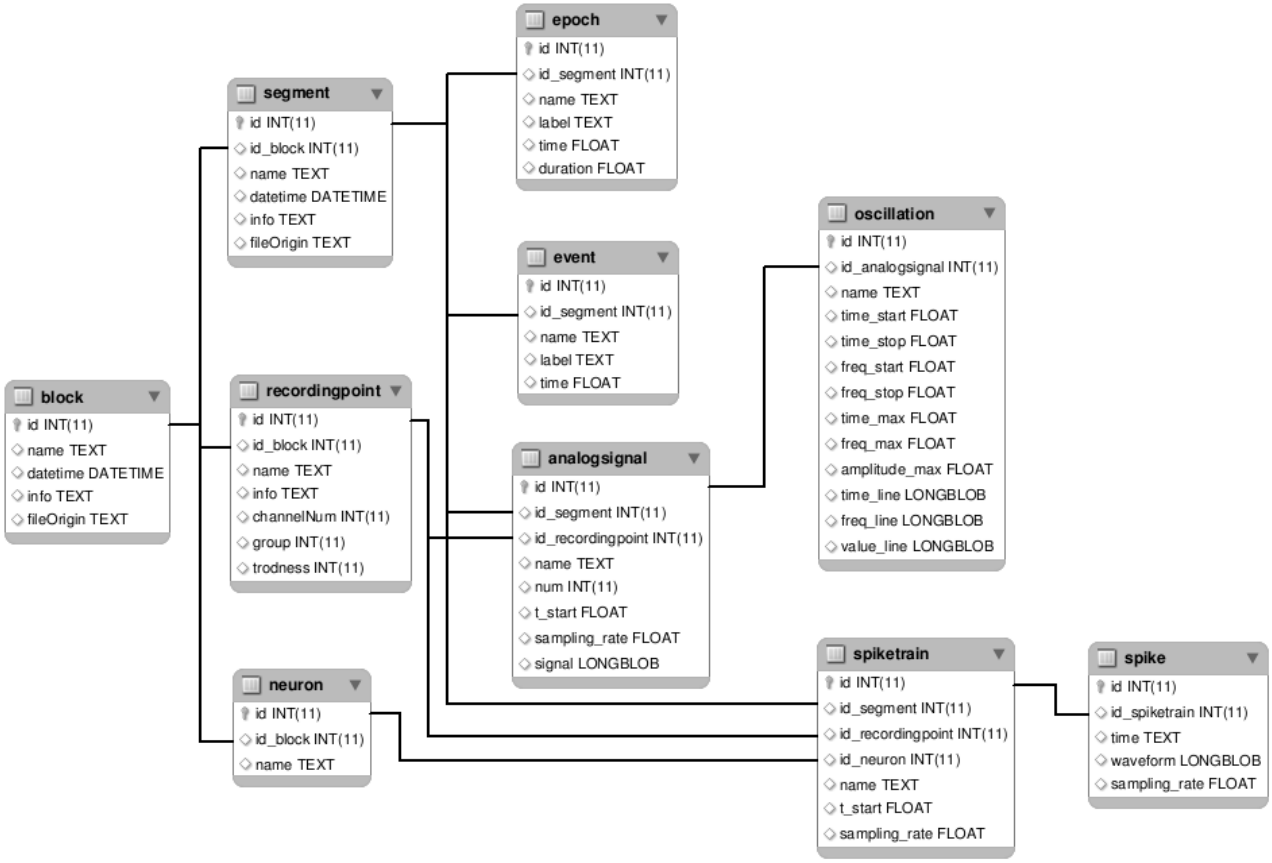


Figure 1: Database scheme. It presents the different tables and fields present in the database. The lines represent the relationship between tables by linking the related indices. For example, each member of the table recordingpoint *is part of a* block*, and contains* analogsignal*s and* spiketrain*s.*

new data files as explained above, the OpenElectrophy IO fills only the basic database scheme, but IO classes can be derived with Python scripting to fill the new fields directly at the time of data importation, from the filename or auxiliary files for example.

3.2 Data exploration and visualization

The main window of OpenElectrophy is shown in figure 2A. It integrates one or many tree views with which the user can explore the database. The first tree view node is one of the database table and its descendants are automatically found according to the index references found in each table. Each tree view is fully customizable by the user who can thus sort its data according to its need. In a second step, most of the OpenElectrophy object can be visualized with various plotting method. For example, analog signals can be plotted raw, filtered, or as time-frequency map.

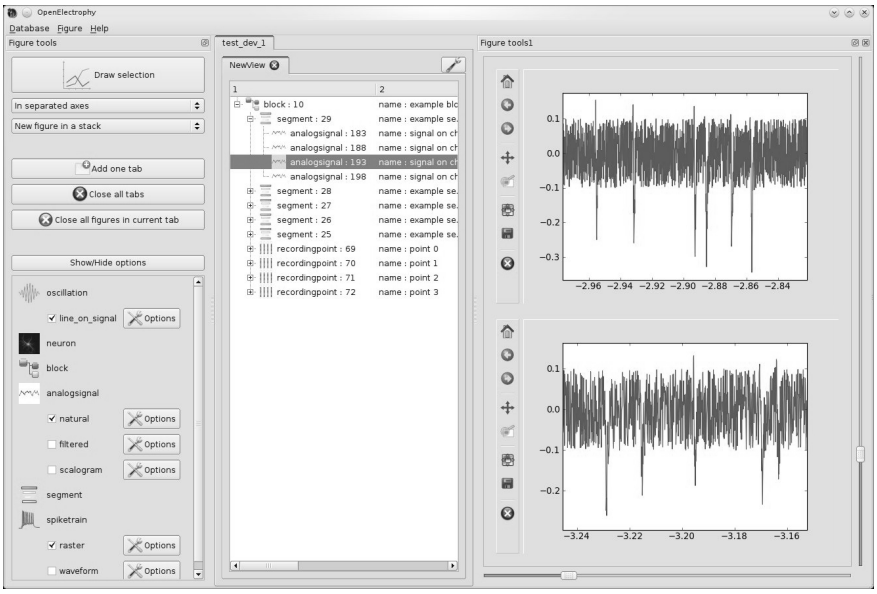
3.3 Spike extraction

The next major step is the extraction of the phenomena of interest: spikes and transient oscillatory events (see next section). In these two cases, a graphical interface helps in searching for parameters that allow for good detection. This step is crucial for subsequent stages of the analysis. There are two possible methods for detection: individual detection, which is done signal-by-signal, or bulk detection, which is done by applying the same parameters to an ensemble of signals targeted by an SQL query that is directly written in the OpenElectrophy GUI.

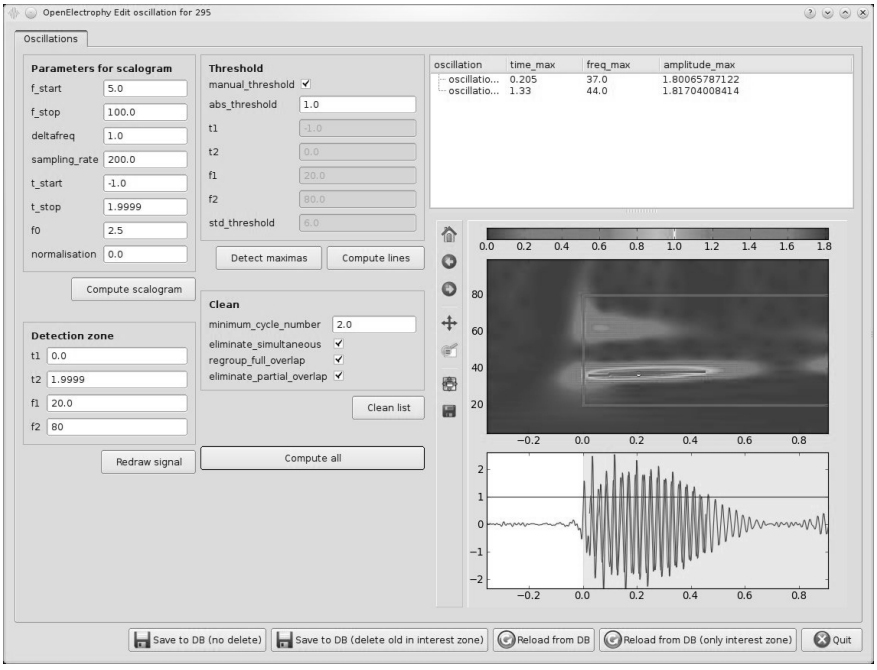
With regard to spike detection, the GUI is shown in figure 2C. The detection has been thought as a set of independent operations for which different methods are available. The list of operations is: time filtering, event detection, feature extraction and clustering. This modular implementation allows the user to test different spike sorting methods and to adapt each step of the detection to its own data. Moreover, implementing new methods requires to script only a minimal part of the code to make it usable by OpenElectrophy. Importantly, the spike extraction GUI has been thought to work with single or multi-electrode detection and includes many graphic tools to check and finely tune the detection (N-dimensional viewer, manual spike selection, clustering on subsets of events, etc). Thus any new method implemented in OpenElectrophy need only to include the computational part of the method, everything else being already present.

After the detection, the central idea of the framework is to store individual spike events in the SQL *spike* table and group them using the *spiketrain, neuron* and *recordingpoint* tables. All studies on spike discharge will deal directly with these three tables using SQL queries, but will benefit from all of the tables when working with protocol information and context meta-data.

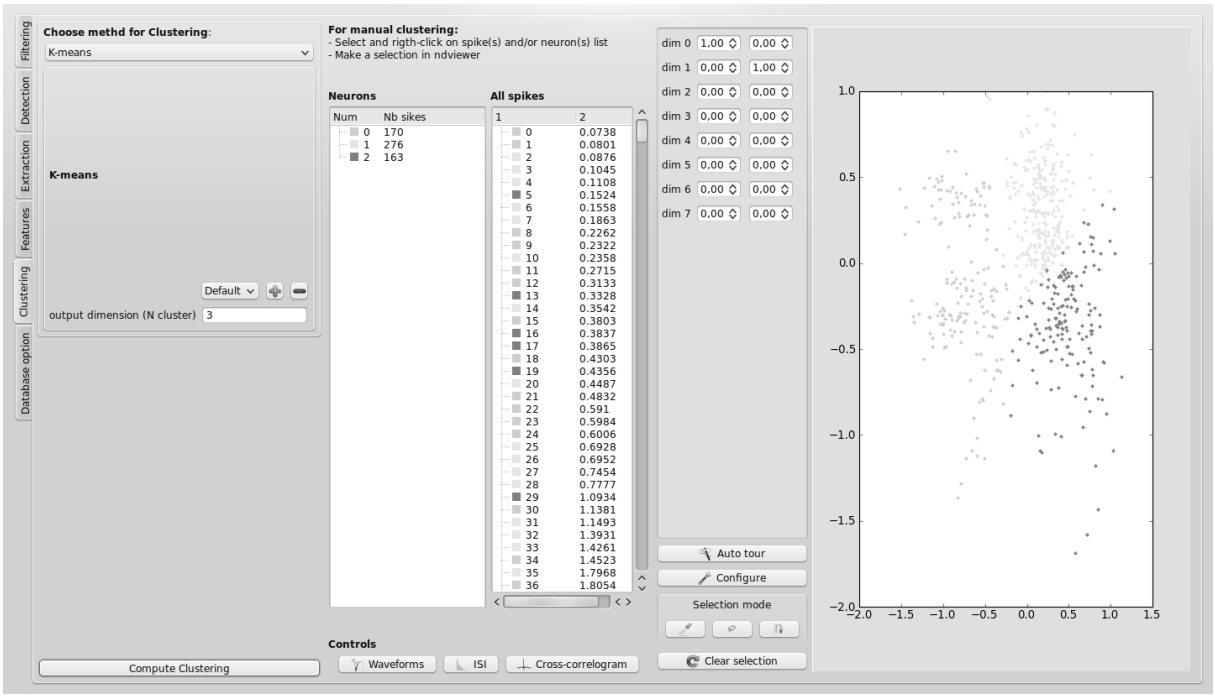
A



B



C



*OpenElectrophy GUI. **A**: main OpenELectrophy window with from left to right: plot option toolbar, database tree view, data plots. **B**: oscillation detection window. On the left, the user can choose the parameters defining: the wavelet time-frequency scalogram, the oscillation detection area, the oscillation detection threshold, some optional "clean" methods used to merge recovering oscillations or eliminate simultaneous oscillations. On the right is shown the list of currently detected oscillations and their plots on the scalogram and on the raw signal. **C**: Spike extraction window. Each tab on the left is dedicated to a specific step of the spike sorting (see text). The filtering tab includes fast-fourier transform, median, Butterworth and Bessel filters, the detection tab threshold is based either on an absolute value or the standard deviation or the median deviation of the raw signal, the projection tab includes principal or independent component analysis and "only max" or no projection, and finally the clustering tab includes K-means or paramagnetic clusterings. By choosing the method at each step, the user can thus find the spike sorting method the best adapted to its data. Additionaly, many tools are available to check extracted spiketrains (autocorrelogram, ISI histograms, waveforms...) and to manually adjust or finely tune the clustering (manual spike selection, N-dimensional viewer...)*

3.4 Oscillation detection

The oscillation detection method is based on a recent approach described by Roux et al. [1]. Briefly, the idea is to compute the time-frequency map of the signal using the Morlet wavelet transform and then detect oscillations as ridges on the power map. The result is a list of oscillations for each signal with for each oscillation instantaneous phase and frequency lines. The GUI devoted to oscillation detection is shown in figure 2B. After the detection, each oscillation is stored in the *oscillation* table. Thus, for studies on multi-frequency oscillatory regimes (e.g., theta, gamma, and beta bands), the analysis is computed directly in this table, although it again also benefits from the data stored in all other tables.

3.5 Analysis

Analysis is the final stage of the OpenElectrophy workflow, which transforms the now pre-processed data into meaningful results. The OpenElectrophy framework does not provide ready-made "point-and-click" analyses for obtaining a given result. Rather, it is necessary to write scripts in Python to perform statistical tests or other specific analyses. Here, the management of the data in a central database simplifies the selection of the data to analyze, and the Python classes provided by OpenElectrophy ease the manipulation of the data to match a given analysis. Additionally, the Python SciPy module [10] provides many standard and high-level analysis tools, and the Matplotlib module [11] offers extensive 2D and basic 3D plotting methods.

Writing analysis scripts can seem difficult for researchers not familiar with programming, but the power and flexibility of this approach is quickly preferred over the restrictive convenience of a GUI. For example, to our knowledge none of the available software for doing spike analysis provides a GUI as an alternative to analysis scripting. Starting with simple script examples is usually sufficient to allow beginners to compose very sophisticated analyses. Thus, OpenElectrophy does not constrain data analysis with a fixed GUI, but allows for the use of user programmable scripts.

As already mentioned, a major advantage of using the Python scripting language is its ability to interface with other languages. Packages like Mlabwrap, rpy, cython or SciPy.weave [12] enable to use pre-existent code from MATLAB, R, or C/C++. In this sense, Python can be seen as a high-level glue language which can, with only a few lines, execute code written with other languages, less flexible and more time consuming while developing but also often more efficient for intensive computations. Employing these tools, the list of external modules that can be linked to OpenElectrophy to help write analysis scripts is long: the International Neuroinformatics Coordinating Facility [13] provides a list of tools available for studying neural data. In particular, OpenElectrophy, as a framework for managing data, would likely complement recent Python-based approaches to neural data studies, such as PyEntropy [14] for information theory and PyMVPA [15] for machine learning.

Details on how to use OpenElectrophy classes for scripting are available on the OpenElectrophy documentation page.

4. Conclusion

In summary, we have presented OpenElectrophy, an open source project aimed at facilitating the management and manipulation of electrophysiological data along with experiment meta-data. The key contribution of OpenElectrophy is the framework architecture: SQL-type database married to Python + SciPy, all of which are reliable, widely used and free tools. We have shown that in OpenElectrophy framework, all of the data and meta-data are recorded in a central database via its extensive IO module which can read data from most of commonly used electrophysiological recording setups. Data and meta-data can then be combined for further analyses, allowing the user, for example, to fuse electrophysiological and behavioral data. We have also shown how OpenElectrophy uses the Python language to simplify interaction with the database and manipulation of data during the writing of analysis scripts. Another primary feature of OpenElectrophy is the integration of the detection and storage of spikes and transient oscillatory events found in electrophysiological recordings.

The OpenElectrophy project is free and open source, which means that anyone can download, use, modify or extend it and then share his work with the whole user community. It is hosted in a forge with a Trac system [16], which offers SVN as a version control system and a wiki for live documentation. A mailing listing for discussion between users and developers is available [17].

At the moment, the OpenElectrophy GUI adequately covers the exploration of data, spike sorting and detection of transient oscillations. The analyses must be computed with Python scripts, which need to be provided by the user. Obviously, these scripts can be written from scratch, but as we already have mentioned, one of the advantages of Python is that it can be interfaced with previously developed analysis toolboxes. Thus, it will be useful in the future to provide, either directly in OpenElectrophy or as script examples (which could be available on the documentation pages for OpenElectrophy), simple ways to interface the data managed by OpenElectrophy with other open source toolboxes, such as, e.g., PyMVPA, PyEntropy or NeuroTools. Interestingly, some of these projects are hosted by the neuralensemble community [16]. It is a developer community which aims at creating interoperable and complementary Python-based tools. In this direction, the recent neo [18] project proposes generic classes to manipulate "electrophysiological" signals and an import/export module as exhaustive as possible. These modules and classes, extensively used by OpenElectrophy, should be used by other projects like Neurotools and thus further facilitate interaction between projects.

Among the future developments, we aim at integrating into OpenElectrophy most of the spike sorting methods currently in use. These methods are written with different languages and interfaces which prevent to simply compare them (see [19] for examples). The ability of Python to call functions written with other languages and our modular construction of spike sorting in OpenElectrophy should allow us to achieve this task. In particular, for each step of the spike extraction it will be necessary to identify in other projects which functions are performing the equivalent step (clustering for example) and then write an interface class able to call the external functions with the correct data arrangement. OpenElectrophy includes for now only few spike extraction methods but after the completion of this work, we hope it can serve as a benchmarking tool for spike sorting methods.
Finally, from a technical point of view, OpenElectrophy is currently limited by the computer memory size to manipulate very long signals at high sampling rates. The use of the concept of BLOB streaming [20], which consists in loading BLOB (binary) fields into a stream chunk by chunk, while using MySQL to read continuous electrode data should be a great improvement.

References

[1] S. G. Roux, T. Cenier, S. Garcia, P. Litaudon & N. Buonviso, A wavelet-based method for local phase extraction from a multi-frequency oscillatory signal., J Neurosci Methods, 160, 2007, 135-143.Roux et al.

[2] S. Garcia and N. Fourcaud-Trocmé, OpenElectrophy: an electrophysiological data- and analysis-sharing framework. *Front. Neuroinform.* *3*(14), 2009, doi:10.3389/neuro.11.014.2009

[3] http://neuroshare.org/

[4] http://www.carmen.org.uk/

[5]Axon:http://www.moleculardevices.com/home.html, http://sccn.ucsd.edu/eeglab, ELAN software package (INSERM U821, Lyon), Elphy: http://www.unic.cnrs-gif.fr/software.html, http://www.micromed.eu/, NeuroExplorer, http://www.plexoninc.com/, http://neuralensemble.org/trac/PyNN, Spike2: http://www.ced.co.uk/, WinWCP: http://spider.science.strath.ac.uk/sipbs/software.htm

[6] http://www.alphaomega-eng.com/, http://www.tdt.com/

[7] http://www.sqlite.org/, http://www.mysql.com/

[8] http://www.sqlalchemy.org/

[9] http://neuralensemble.org/trac/NeuroTools

[10] http://www.scipy.org/

[11] http://matplotlib.sourceforge.net/

[12] http://mlabwrap.sourceforge.net/, http://rpy.sourceforge.net/, http://www.cython.org/, http://www.scipy.org/Weave

[13] http://software.incf.org/

[14] R.A.. Ince, R.S. Petersen, D.C. Swa ,and S. Panzeri, Python for information theoretic analysis of neural data. *Front. Neuroinform.* *3*(4), 2009, doi: 10.3389/neuro.11.004.2009.

[15] M. Hanke, Y.O. Halchenko, P.B. Sederberg, E. Olivetti, I. Fründ, J.W. Rieger, C.S. Herrmann, J.V. Haxby, S. Hanson and S. Pollmann. PyMVPA: a unifying approach to the analysis of neuroscientific data. *Front. Neuroinform*. *3*(3), 2009, doi: 10.3389/neuro.11.003.2009.

[16] http://neuralensemble.org/trac/OpenElectrophy

[17] http://groups.google.fr/group/openelectrophy

[18] http://neuralensemble.org/trac/neo

[19] STAR: http://sites.google.com/site/spiketrainanalysiswithr/
Chronux: http://chronux.org/
Klutakwik: http://klustakwik.sourceforge.net/
Wave_clus: http://www.vis.caltech.edu/~rodri/Wave_clus/Wave_clus_home.htm
Mclust: http://www.neuroinf.org/lists/comp-neuro/Archive/2000/0065.html

[20] http://blobstreaming.org/