Proceedings of the
8th Graz Brain-Computer Interface Conference 2019 DOI: 10.3217/978-3-85125-682-6-15

RESONANCE - A BCI FRAMEWORK FOR WORKING WITH MULTIPLE DATA SOURCES

Y.O. Nuzhdin

E-mail: nuzhdin.urii@gmail.com

ABSTRACT: Resonance is a framework for creating reproducible BCI experiments with multiple sources of data processed together. It is a cross-platform tool created with C++. It offers ways to create visual environments using QML and perform data processing with R and Python. Resonance has proved its work with multiple brain signal capturing devices including EEG, MEG, Eye tracking and number of other devices. Resonance framework does not exclude developer from the process of experiment creation, but it allows to focus on details of an experiment by using user-friendly languages and concepts. The framework is actively developed and was already used for various experiments in several laboratories. All data-processing code is open sources[6][5].

INTRODUCTION

Many novel non-invasive BCIs are trying to achieve maximum signal efficiency and accuracy with the help of multiple tools. That can be data acquiring devices like electroencephalography (EEG), magnetic resonance imaging (MRI), magnetic encephalography (MEG), eye trackers or stimulators like visual and auditory stimulators, transcranial magnetic stimulation (TMS), electro myostimulation (EMS). Using multiple data source helps understanding brain signal deeply and making better interfaces, however it requires a simultaneous work and a very precise synchronization between different devices.

There is a number of software tools for performing BCI experiments[2], and that list is growing[12]. These tools are created with different programming languages, has different capabilities and features. Previously, we successfully used some of them in experiments[11][1]. However, for new experiments we were looking for a platform that would be capable of processing data from multiple devices simultaneously and online.

Resonance was designed as a framework that can tight up multiple devices and provide an experimenter with an ability to design, acquire, view and analyze data in a user-friendly and unified manner.

The development started in 2014 by an independent team and to date several experiments have been performed with the Resonance framework [3][4][10][13].

FEATURES

Resonance was never meant as a tool that excludes a developer from designing an experiment. Instead it provides a set of tools and general approach that lowers requirements for knowledge and skills. It allows one to concentrate on details of experiment instead of low-level machine interaction details.

Within Resonance system we distinguish following aspects of an experiment: data collection, aggregation, processing and storage, experimenter feedback loop and subject feedback loop. We use the term 'loop' because not only experimentation system affects subject, but a subject may also affect experimentation system. Experimenter can monitor diagnostic data and change settings during an experiment. The subject can affect experiment in two different ways: by being a data source (if we measure parameters, which subject can control) and by changing a procedure and parameters of an experiment (for example by pressing a button when one is ready).

Our goal is to provide an experimenter with implementations for some of these aspects without artificial limitations that simplify creation of the framework but limit abilities to make complicated experiments.

Resonance provides a set of tools to fulfill requirements of named aspects:

- Data collection - Resonance implements drivers for devices. Hence an experimenter has access to all configuration options and working modes provided by a device. These options and modes could be changed during experiment's execution. It is also possible to monitor device failures.
- Data aggregation and data processing - Resonance provides libraries on R[6] and Python[5] that allow to combine and process data from multiple sources.
- Data storage - Resonance provides a facility to store data. It uses its own open data format because it stores every single bit of data produced by the system, which allows for example to detect problems with hardware after an unsuccessful experiment.
- Experimenter and subject feedback loops - Resonance provides libraries for QML to control all aspects of an experiment, including handy widgets like data visualization. In fact, every single UI piece of Resonance is implemented with QML modules and is available open source, so if you find any of existing tools useful you can copy it to your experiment. QML is a declarative language for designing

 DOI: 10.3217/978-3-85125-682-6-15

user interface-centric applications. Interface's behavior, designed with QML, can be scripted through JavaScript. The tool is multi-platform: available on Linux, Windows, MacOS and Android. It uses video acceleration to produce fast graphical output. From our experience it is not only possible but also very easy to implement tempting features like stimuli presentation for a single frame at desired time without any knowledge of how video system works.

It is important to mention that Resonance can be integrated with a third-party software without any limitations.

SYSTEM USAGE EXAMPLE

We will illustrate abilities of the system at the non-typical BCI experiment performed in Lomonosov Moscow State University[13]. In this experiment we recorded EEG and EMG during TMS stimulation. At each run subject performed two types of mental tasks - one of three motor imagery tasks (experimental condition) and a visual attention task (reference condition). In EMG analysis, the amplitude of TMS-evoked motor potentials (MEP) was measured peak to peak (between negative and positive phases of potential). Potentials with any signs of raised muscular activity during preceding 1000 ms interval were rejected manually. MEP amplitudes in experimental condition (motor imagery) were normalized to mean amplitude of the reference condition ("visual attention" or "blank screen") of the same run. The mean of the subject's MEP amplitudes during fingers flexing imagery task minus 100% was used as a corticospinal excitability increase measure for further analysis.

During TMS measurement online EMG-feedback was displayed at the right side of the screen as a vertical bar with a real-time RMS (root-mean-square) value (300 ms window, 100 ms step)(figure 2). Participants were asked to find hand position with minimal ongoing EMG amplitude and to keep the corresponding bar level constant during the whole run.

To control quality of TMS stimulation during the experiment EMG responses to stimulation with amplitude and latency measurements were shown on the experimenter's screen. Both EEG and EMG data were recorded for offline analysis.

At the center of subject's screen was an image indicating current task. At the right side of subject's screen was a bar showing subject's muscle activity. Value for this bar was calculated from EMG measures online to help a subject to reach a muscle relaxation. On a separate screen an experimenter saw an interface with all parameters of the experiment. At the right-side column TMS-invoked motor potentials (MEP) were displayed. Each row contained EMG data, amplitude and latency values for single stimulation (figure 1).

SYSTEM ARCHITECTURE

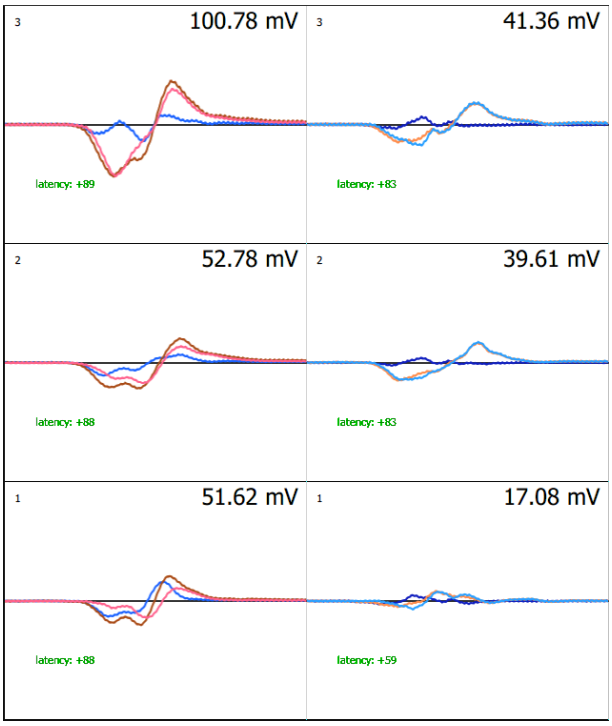

Figure 1: Part of experimenter's screen with MEP, latency and amplitudes measurements. Colors are changed for printing.

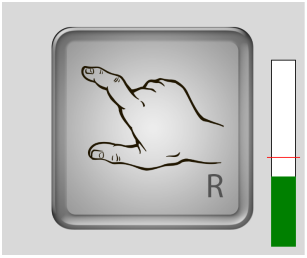

Figure 2: Subject screen. Pictogram of a hand represents mental imagery task cue and bar on the right side shows real-time RMS of subject's EMG from forearm muscles.

The framework provides a general data-flow-centric architecture, which should fit any possible experiment. In this approach we distinguish Resonance services: nodes of data processing and control. Each service can publish data streams, its own parameters and state (e.g. operation mode of a service). All communications between services go through TCP/IP network and services can be started at the same or different machines. Each service can discover other services in a local network, receive data streams, update parameters of other services and send commands to change state (for example start calibration or recording). This approach makes every service independent from others, which brings modularity to system and helps with debugging and analyzing problems.
In the example above were 5 services -

- an EEG driver
- an EMG driver (EEG and EMG were recorded by two different devices)
- a service for subject's screen

 DOI: 10.3217/978-3-85125-682-6-15

- a service for experimenter's screen
- a service for EMG data processing using R[9] script engine

Data from EEG and EMG devices was collected by experimenter's service and recorded to a file by standard Resonance facility. Data from EMG device was also acquired by R script engine. This service was executing R script to calculate RMS of EMG. This service also gathered myographic data after TMS stimulation to display results on experimenter's screen. Subject service was consuming RMS data from R script engine and stimuli commands from experimenter's service. Experimenter service was responsible for experimentation cycle. It contained a finite state machine to control a stimuli sequence and UI module to display myographic data. Myographic data was updated after each stimulation by detecting corresponding event in a runtime.

ONLINE AND OFFLINE DATA PROCESSING

Making data processing algorithms is not easy, especially when you need to model data processing before or after the experiment. Typical approach to this task is to provide system with a set of data transformation blocks, which you can combine to perform required data processing steps. The positive side is that it is relatively easy to combine and configure such blocks. First problem with this approach is that experimenter is limited by the number of kinds of these blocks. Also creating these blocks requires a good knowledge of the internals of a BCI system and a language with which that system is written (usually C++). Second problem is that it is hard to use these blocks for offline data processing, as they are highly integrated with BCI system, while offline data processing is usually performed by Python, R or Matlab.

Having separate code for online and offline processing makes experiment development very hard. Achieving exactly same computational result in computer systems requires knowledge of how computation is made step-by-step. This means that a developer of an experiment must translate online processing to an offline tool or hope that they will produce results which differ insignificantly. Achieving equal results for offline and online is not a requirement for every application, but that limits in ability to detect irregularities or test additional hypotheses post-hoc. For example if you discover an unusual behavior in an online experiment it will be difficult to figure out if it was due to computational errors or anything else.

In order to overcome this problem Resonance provides a library that can be used for both online and offline data processing. The data flow is still constructed from blocks, but these blocks (as well as all data processing) are written in a language which is more familiar for experimenter (R and python for now) and these blocks are not coupled with the details of online data processing. That means that you can get exactly same results using data from online experiment, and even more, you can model online performance of the system offline, using a language you are familiar with. In fact data processing library is completely abstract from any Resonance-specific data processing, and with minor adaptations could be used with any system which supports R or python language in runtime.

Using Resonance library allows not only to execute online and offline processing, but also helps to visualize it. For example, figure 3 is the automatically generated visualization of data processing plan for EMG processing from the example experiment.

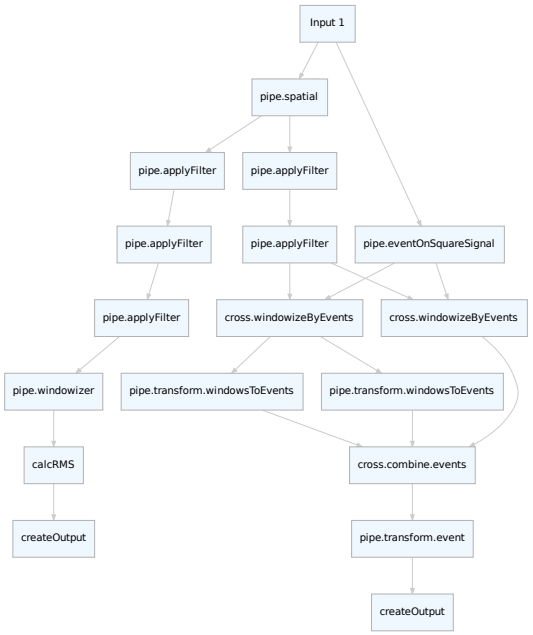

Figure 3: Data processing plan for EMG processing. Arguments for the processing steps are not visualized on this chart.

DATA AGGREGATION AND SYNCHRONIZATION

Data aggregation:

For processing digital signals you can work in two different domains - in sample space and in time space.

- Working in sample space means that you quantify time in your system by a number of samples of the signal you are processing.
- Working in time space means that you measure your time by some timer which is not bound with the signal and that timer has finer resolution than sampling rate of your signal.

Working in the sample space seems very natural when you deal with single signal source. But unfortunately, most of real-world applications never work with the single source of the signal. For example - when you present any stimulus, you have two sources of data - one for EEG

 DOI: 10.3217/978-3-85125-682-6-15

and another for moments in time when stimulus was presented. In this situation you usually introduce an additional channel with event data or some kind of flags. Such solution works for some applications, but it cannot be generalized. When you try to work with multiple signals even if they have same sampling rate you will face the problem of phase alignment for these signals. Introducing every new signal source will add at least half of the period of this signal to time desynchronization (or even more if the frequencies of signals are not dividable without the remainder). If signals have sampling rates which are not multiples to each other there is a problem of sub sampling - when one sample of signal A corresponds to a variable number of samples of signal B. If you try to process such signals in parallel you will introduce synchronization deviations, and if you bring the signals to a common frequency you will introduce phase, amplitude and spectrum deviations in signal.

Working in time space makes sense if you use timer with resolution smaller than the minimal period of processed signals. In this case applied transformations still can change timestamps of a samples, but as these timestamps are finer than a period of a signal they would not affect your signal. However, working with timestamps of the recording device may introduce unnecessary complexity to the calculation process. Recording devices do not produce data blocks at exact time intervals. For example, EEG recording device may show deviations between timestamps of the blocks which will usually be bigger than a period of the signal.

Resonance framework provides the following solution to synchronization problem:

1. an initial timestamp for the signal is averaged from first few blocks of data
2. timestamps for samples are recalculated from sampling rate
3. during the data processing Resonance also performs a control, so that there are no delays or deviations in data retrieval

Synchronization:

Synchronization between signals is very easy when you work in a time space - you just align closest timings together and your signals are synchronized, and again, by this operation you introduce phase deviation which is smaller than sampling rate, so it is not reflected in your signal as well. Of course, you need to take into account that different computers use different timers and you need to align these timers if you record data with separate machines. However, if you record data with one machine you will have access to a very precise timer (tens of nanoseconds) of the machine, which is enough for any imaginable signal. You can try to achieve time synchronization among different machines via network, but it is more precise and reliable to physically connect recording devices and perform a synchronization based on the same signal recorded by several devices.

In the example experiment there were 3 time-aligned devices - EEG , EMG and TMS. In order to synchronize them EEG and EMG were connected to a TMS trigger channel. TMS sent pulse through this channel each time a stimulation was performed. By that means we achieved not only data-synchronization between EEG and EMG, but also very reliable source of time information when TMS stimulation was performed. That time information was used to extract EMG data related to the stimulation, that was presented on experimenter's screen.

Performance considerations:

When talking about working in signal or time space we usually mean working with separate samples. But for computers calculating separate samples is too inefficient and introduces a big overhead, so we are forced to collect and process data in blocks. From the other side - gathering block of samples introduces latency, and you cannot immediately process collected data. The decision whether blocks are equal in size or not should be made by an experimenter, not dictated by a framework. In order to decide one need to take into account performance requirements of one's application, the way hardware works (for example is it capable to return data blocks of variable size) and what data processing algorithms are used in experiment. Resonance supports blocks of variable size, and each block has a timestamp of creation and a timestamp of receiving. That allows monitoring delays in a network or in blocks' processing. Resonance's R and Python open source libraries provide abstraction over data stream, which is independent of block size and amount. That allows experimenter to try experiment with different block size settings in order to achieve required performance and latency.

PLANS FOR THE FUTURE

For the upcoming year there are few big improvements planned:

- Demo experiments. Currently Resonance distributive contains only basic signal processing tools. Our plan is to share some experiments and tools we made with Resonance. That will provide a basis for other experimenters to build and reproduce their experiments. In order to achieve that existing implementations should be reviewed and documented. These demo experiments will be freely available at the Resonance website[7], as well as instructions about how to install and run them.
- Extend support for devices. Currently there are several recording devices natively supported by Resonance (actiChamp, NVX and openBCI eeg recorders, eyelink and eyetribe eyetrackers, neuromep-4 myostimulator) Many other devices could be connected using fieldtrip[8] protocol, but fieldtrip does not allow to control device parameters and modes. In upcoming year, we will extend native support for at least EEG, MEG and eyetracking devices,

 DOI: 10.3217/978-3-85125-682-6-15

and probably implement other popular protocols like Lab Streaming Layer.

- Code improvements. Resonance claims to be the system that exactly reproduces experiments. This functionality is the core of the system and was extensively tested in simulations and real applications. Last year there was a big improvement in number of automated tests for this functionality. These tests explicitly proof that everything you can do with Resonance have exactly the same behavior online and offline, every time you run application on any machine. For the upcoming year we have a goal to completely cover R and python Resonance libraries with tests.

List of improvements, updates and plans, and other information is available on the website of the framework[7]. All source code for online and offline data processing is freely available on GitHub repositories for R[6] and Python[5].

ACKNOWLEDGEMENTS

Authors would like to acknowledge Anatoly Vasilyev and Sergey Shishkin for providing useful advices and feedback and for their help in testing.

References

[1] A.A. Fedorova et al. A fast "single-stimulus" brain switch. In: Proceedings of the 6th International Brain-Computer Interface Conference 2014. Sept. 2014.

[2] Brunner Clemens et al. BCI Software Platforms. English. In: Toward Practical BCIs: Bridging the Gap from Research to Real-World Applications. Springer, 2013, 303–331.

[3] Nuzhdin Yuri O. et al. Passive Detection of feedback Expectation: towards Fluent Hybrid eye-brain-Computer Interfaces. In: From Vision to Reality - Proceedings of the 7th Graz Brain-Computer Interface Conference, GBCIC 2017, Graz, Steiermark, Austria, September 18-22, 2017. 2017.

[4] Nuzhdin Yuri O. et al. The Expectation Based Eye-Brain-Computer Interface: An Attempt of Online Test. In: Proceedings of the 2017 ACM Workshop on An Application-oriented Approach to BCI out of the Laboratory. BCIforReal '17. Limassol, Cyprus: ACM, 2017, 39–42.

[5] Nuzhdin Yury. *Resonance Python package*. `https://github.com/tz-lom/Resonance-engine-python`.

[6] Nuzhdin Yury. *Resonance R package*. `https://github.com/tz-lom/Resonance-engine-R`.

[7] Nuzhdin Yury. *Website of the Resonance project*. `http://resonance.bcilab.net`.

[8] Oostenveld Robert, Fries Pascal, Maris Eric, Schoffelen Jan-Mathijs. FieldTrip: Open Source Software for Advanced Analysis of MEG, EEG, and Invasive Electrophysiological Data. Computational Intelligence and Neuroscience. 2011;2011.

[9] R Core Team. *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing. Vienna, Austria, 2018.

[10] Shishkin Sergei L. et al. EEG Negativity in Fixations Used for Gaze-Based Control: Toward Converting Intentions into Actions with an Eye-Brain-Computer Interface. Frontiers in Neuroscience. 2016;10:528.

[11] Shishkin S.L., Nikolaev A.A., Nuzhdin Y.O., Zhigalov A.Y., Ganin I.P., Kaplan A.Y. Calibration of the P300 BCI with the single-stimulus protocol. In: Proceedings of the 5th Graz Brain-Computer Interface Conference, GBCIC 2011, Graz, Steiermark, Austria, September 22-24, 2011. 2011, 256–259.

[12] Smetanin Nikolai, Volkova Ksenia, Zabodaev Stanislav, Lebedev Mikhail A., Ossadtchi Alexei. NFBLab—A Versatile Software for Neurofeedback and Brain-Computer Interface Research. Frontiers in Neuroinformatics. 2018;12:100.

[13] Vasilyev Anatoly, Liburkina Sofya, Yakovlev Lev, Perepelkina Olga, Kaplan Alexander. Assessing motor imagery in brain-computer interface training: Psychological and neurophysiological correlates. Neuropsychologia. 2017;97:56–65.