The Journal of Open Source Software

PhysioLabXR: A Python Platform for Real-Time, Multi-modal, Brain–Computer Interfaces and Extended Reality Experiments

Ziheng 'Leo' Li [1*¶], **Haowen 'John' Wei** [1*], **Ziwen Xie** [1], **Yunxiang Peng** [1], **June Pyo Suh** [1], **Steven Feiner** [1], **and Paul Sajda** [1]

1 Columbia University, New York, New York, United States of America ¶ Corresponding author * These authors contributed equally.

DOI: 10.21105/joss.05854

Software

- Review
- Repository
- Archive

Editor: Marcel Stimberg

Reviewers:

- @lucask07
- @nastaran62

Submitted: 09 September 2023
Published: 11 January 2024

Summary

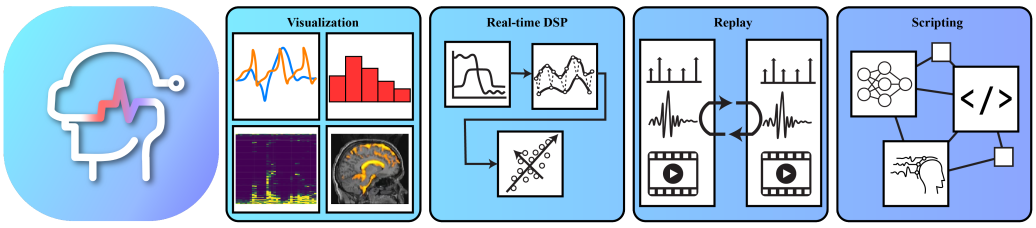


Figure 1: PhysioLabXR includes various visualization methods, digital signal processing modules, support for recording and replaying experiments, and a scripting interface to deploy custom pipelines.

PhysioLabXR is a Python-based open-source software platform for developing experiments for neuroscience and human–computer interaction (HCI) that involve real-time and multi-modal physiological data processing and interactive interfaces. *PhysioLabXR* provides native support for data sources such as electrophysiological sensors (e.g., EEG, EMG, and EOG), fNIRS, eye trackers, cameras, microphones, and screen capture, and implements the popular data transfer protocols Lab Streaming Layer (LSL; Kothe & Mandel, n.d.) and ZeroMQ (ZMQ; ZeroMQ, 2021). It features multi-stream visualization methods, real-time digital signal processing (DSP) modules, support for recording and replay experiments, and a Python-based scripting interface for creating custom pipelines.

PhysioLabXR has an architecture optimized through concurrency and parallelism to ensure efficient performance. We provide a set of detailed tutorials covering all features and example applications, such as a P300 speller with a Unity frontend (Unity Technologies, 2005) and a mental arithmetic experiment interfacing with PsychoPy (Peirce, 2007). An accompanying set of benchmarks demonstrates the ability of *PhysioLabXR* to handle high-throughput and multi-stream data reliably and efficiently. Published use cases show its versatility for VR and screen-based experiments (Koorathota, 2023; Lapborisuth et al., 2023) and sensor fusion studies (Wei et al., 2022) [1].

[1] *PhysioLabXR* was formerly called *RealityNavigation*, and *RNApp* in older publications.

Statement of Need

Recent years have seen a growing interest in multi-modal experiments, often involving closed-loop interaction systems, in neuroscience and human–computer interaction (HCI). Many emerging paradigms have found new roots in extended reality (XR) environments, including virtual reality (VR) and augmented reality (AR). Such experiments are increasingly fusing multiple modalities and combining different physiological measurements. For example, one sensor can generate events to extract meaningful data intervals from other sensors, such as fixation-related potential (FRP) studies in which EEG epochs are locked to visual fixations from eye trackers (Nikolaev et al., 2016). Multiple physiological signals can also be combined to enhance their predictive power for use in applications ranging from emotion recognition (He et al., 2020; Koelstra et al., 2011) to movement actuation via sensorimotor rhythms (Sollfrank et al., 2016). Further, multi-modal paradigms can facilitate the exploration of how different physiological systems interact; for example, pupil dilation can be used as a proxy for the locus coeruleus activity as measured via functional magnetic resonance imaging (fMRI; Murphy et al., 2014).

Despite the prevalence of these experiments, software tools for real-time physiological data handling are surprisingly few and far between. They can be categorized into two groups: device-specific tools and device-independent tools. Device-specific tools, which are typically proprietary, offer data visualization and analysis (Lührs & Goebel, 2017; *NIRx*, n.d.; Tobii AB, 2023) for the hardware to which they are tied. However, they often lack support for multi-modal experiments. To address this, researchers have created custom data pipelines aided by third-party data transfer protocols such as LSL and ZMQ (Baltrušaitis et al., 2016; Kothe & Mandel, n.d.; MacInnes et al., 2020; Michalareas et al., 2022; Wang et al., 2023). This approach is typically time-consuming and requires substantial effort to adapt to new experiments. In addition, the data transfer middleware typically does not allow researchers to visually inspect data streams in real-time. This can be a crucial feature for many experiments, particularly those involving devices prone to failure and artifacts during operation, such as in EEG and fNIRS. Real-time visualization allows experimenters to react promptly to sensor failures and prevents wasting valuable participant time.

Device-independent tools, including popular platforms, such as OpenVibe (Renard et al., 2010), MNE Scan (Esch et al., 2018), NeuroPype (Neuropype, 2023), and iMotion (iMotions, 2023), support real-time visualization. However, they are primarily written in statically compiled languages, limiting customization, and some are closed-source commercial products, such as NeuroPype and iMotion. Python's rise in popularity as a programming language (Srinath, 2017) has made it an obvious choice for developing new device-independent tools that allow customization through rapid prototyping. However, using Python as a backbone language for high-precision and high-throughput data necessitates significant optimization to match the performance level of a compiled language. Octopus-sensing (Saffaryazdi et al., 2022) is an example of a Python-based platform that supports the acquisition and visualization of multi-modal data.

Nevertheless, there remains a gap for an all-in-one open-source platform that supports multi-modal data visualization, and rapid prototyping for developing experiment pipelines in complex XR environments, while addressing the optimization challenges of basing on an interpreted language such as Python.

Benefits

PhysioLabXR is a complete all-in-one GUI application for visualizing, recording, and replaying neuroscience and HCI experiments, and deploying end-to-end DSP & machine learning (ML) pipelines in XR. It offers the following benefits: (1) a user-friendly graphical user interface (GUI) for working with both physiological and behavioral data; (2) a reliable, robust, high-performance backend capable of synchronizing and processing multi-modal and high-throughput data in a scalable manner; (3) a workflow that streamlines the hitherto time-consuming and challenging steps in experiment cycles, including visualizing, recording, and analyzing data offline (e.g., to understand physiological phenomena) or online (e.g., to provide neurofeedback in a brain-computer interface); (4) flexibility and ease of setup as a cross-platform solution; and (5) an extensive developer API, which encourages users to extend the platform with custom hardware and real-time processing scripts.

Python dominates the implementation from frontend GUI to backend servers without sacrificing performance, thanks in part to its concurrent runtime architecture. Selected portions, such as real-time DSP, are written in Cython (Behnel et al., 2010), a statically compiled language with Python-like syntax, to further improve performance. Users can use *PhysioLabXR* as a scaffold and leverage Python's extensive APIs to shape the platform according to their needs.

Users can write Python scripts to interact with any data stream and communicate processed results with built-in I/O modules. This flexibility allows users to design closed-loop systems, including deploying ML models and sending predictions to and from *PhysioLabXR*.

PhysioLabXR can be used with popular stimulus-presentation software such as Unity (Unity Technologies, 2005), PsychoPy (Peirce, 2007) and other analysis software, including MATLAB (MathWorks Inc., 2021). For experiments already utilizing LSL and ZMQ for data transfer, the software provides convenient network stream connectivity with these two widely-used data middleware. As its name implies, *PhysioLabXR* has extensive support for XR, including headset-based VR and AR. This builds on our previous work, where we developed an environment to support neuroscience experiments that utilize Unity and other advanced stimulus paradigms (Jangraw et al., 2014).

PhysioLabXR adheres to industry-standard software development guidelines, including continuous integration. Its modular software architecture simplifies the learning curve for users who wish to add custom functionality, such as support for new sensors.

PhysioLabXR: Working with Streams

All functionality in *PhysioLabXR* is based on *Streams*. A stream is a sequence of data points that arrive in real-time, with each frame of data carrying a timestamp, whether it is from physiological sensors, video cameras, microphones, screen capture, or software-generated data. *PhysioLabXR* provides a unified interface for working with streams for visualization, recording, replaying, and DSP. Each feature addresses different requirements in experiments involving real-time data collection and processing. Here, we provide a brief overview of these features:

- *Data stream API* establishes a connection with data sources, either through native plugins or network protocols (LSL or ZMQ).
- *Visualization* helps users visually inspect their data in real-time to understand their data better.
- *Recording* lets users capture experimental data in real-time and export them for further analysis.
- *Replaying* enables users to play back data streams from past experiments and, if needed, test their data processing script and algorithm in real-time as if the experiment is running live.
- *DSP* is another powerful feature allowing users to apply predefined signal processing algorithms to their data streams.

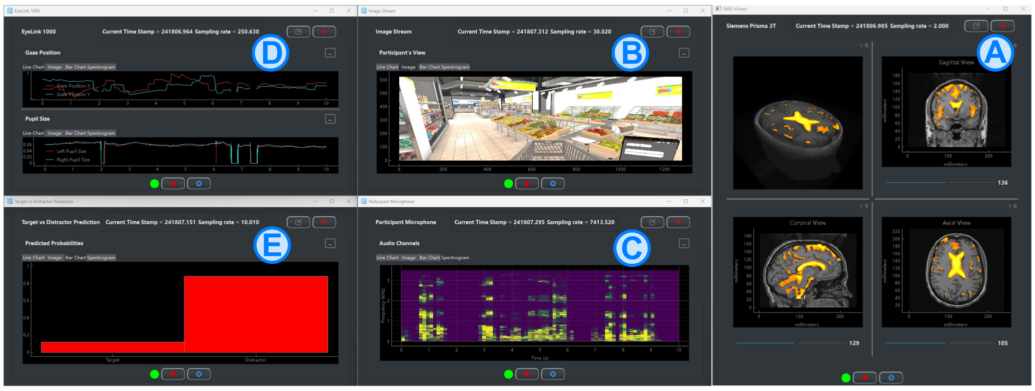
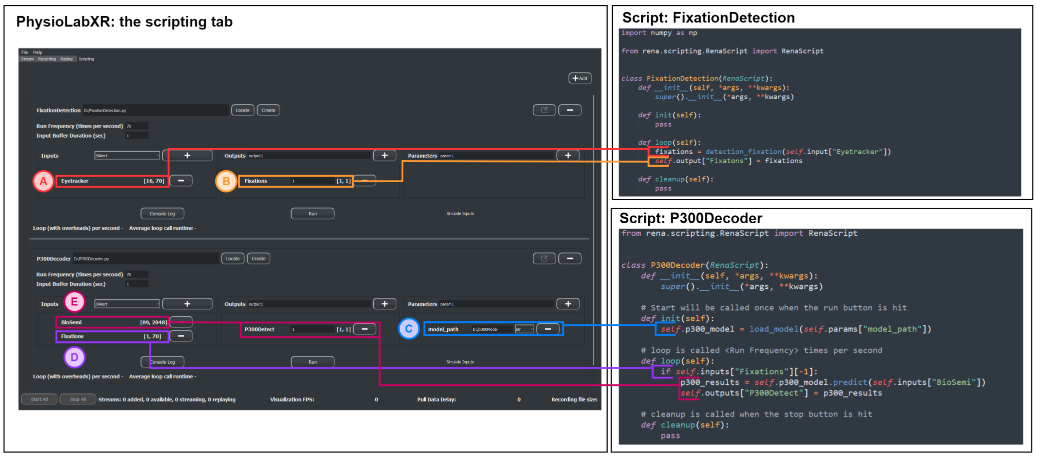


Figure 2: Example use case of *PhysioLabXR* in a memory formation and retrieval experiment involving real-time processing of pupillometry and fMRI streams. This example demonstrates the diverse visualization options provided. In this experiment, the participant is asked to navigate a virtual shopping mall and respond verbally during their task. (A) The 3D fMRI visualizer shows fMRI data streamed in real-time. (B) The experimenter uses PhysioLabXR to monitor and record the scene from the participant's first-person view while they perform the task. (C) The participant's speech is captured using a microphone connected to the software that visualizes the audio data as a spectrogram. (D) Eye movement and pupillometry data are recorded through an eye tracker outside the scanner that receives the participant's eye image via a mirror. The time series of the eye-tracking data are plotted in a line chart. (E) Simultaneously, an ML model deployed through PhysioLabXR's scripting interface predicts from the fMRI data and pupillometry if a target memory is retrieved, with the two-class inference result visualized as a bar plot.

Scripting Interface



Script: FixationDetection

```python
import numpy as np

from rena.scripting.RenaScript import RenaScript


class FixationDetection(RenaScript):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    def init(self):
        pass

    def loop(self):
        fixations = detection_fixation(self.input["Eyetracker"])
        self.output["Fixatons"] = fixations

    def cleanup(self):
        pass
```

Script: P300Decoder

```python
from rena.scripting.RenaScript import RenaScript


class P300Decoder(RenaScript):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)

    # Start will be called once when the run button is hit
    def init(self):
        self.p300_model = load_model(self.params["model_path"])

    # loop is called <Run Frequency> times per second
    def loop(self):
        if self.inputs["Fixations"][-1]:
            p300_results = self.p300_model.predict(self.inputs["BioSemi"])
            self.outputs["P300Detect"] = p300_results

    # cleanup is called when the stop button is hit
    def cleanup(self):
        pass
```

Figure 3: Example script setup for a fixation-related potential (FRP) experiment. The FixationDetection script (upper right) identifies fixations from the eye-tracking stream, while the P300Detector script (bottom right) decodes EEG data locked to detected fixations, and determines if a target object elicits an FRP. This setup is similar to the experiment conducted by Rämä and Baccino (Rämä & Baccino, 2010). (A) Eye-tracking data is processed by the "FixationDetection" script. (B) The fixation results are streamed through the output LSL outlet "Fixations." (C) In the *.init* function of P300Detector, the P300 classifier model is loaded from the file system path in the script parameter "model_path." (D–E) If a fixation is detected, the model takes the EEG epoch time-locked to the fixation and makes a prediction. A loop call is completed by writing the prediction results to the output stream "P300Detect."

The scripting interface allows researchers to build diverse experiment paradigms. It enables the execution of user-defined Python scripts, empowering users to create and deploy custom data processing pipelines. With Python's versatility and open-source libraries encouraging exploration of novel applications such as closed-loop neurofeedback, users can train and run ML models in real-time, using PyTorch (Paszke et al., 2019), scikit-learn (Pedregosa et al., 2011), and other libraries. Users can communicate results from scripts to external applications using built-in networking APIs, including LSL and ZMQ. The script widget offers a straightforward way to add and run scripts, adjust attributes that influence the data processing pipeline's behavior, and monitor performance. These attributes include defining the streams to use as inputs, setting input buffer duration, controlling run frequency, creating outputs to visualize pipeline results or communicate with other programs, and utilizing exposed parameters that allow variable adjustments during runtime. A script in *PhysioLabXR* consists of three abstract methods—`init`, `loop`, and `cleanup`—which users can override to specify behavior. Built-in scripts support commonly used algorithms such as fixation detection and band-power computation and connection to popular devices such as Tobii eye trackers (Tobii AB, 2023) and OpenBCI EEG caps (*OpenBCI*, n.d.).

All these features can be used in combination, enhancing the overall potential of PhysioLabXR. For example, filters can first be applied to EEG data, and the user can visualize the filtered stream as it drives a BCI with a custom script.

Software Design Principles

Creating an efficient runtime experience is a significant challenge for large-scale Python software when dealing with high-throughput data, complex graphics, and frequent I/O operation from serialization, given the interpreted nature of the language. To make this possible, *PhysioLabXR* is optimized through a combination of concurrency, parallelism, and a modular software architecture.

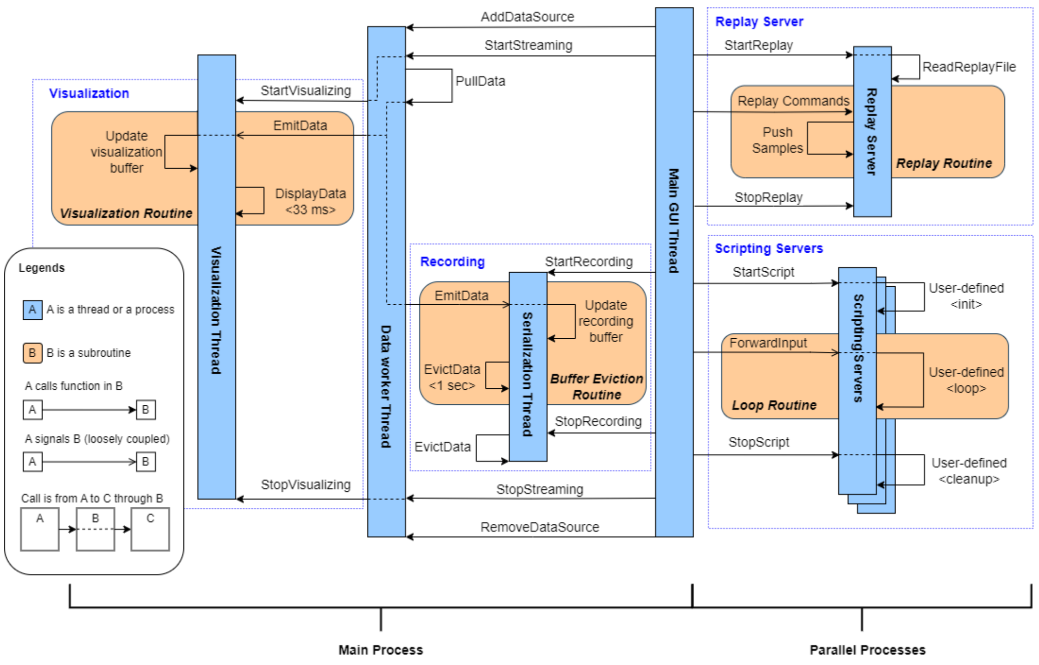


Figure 4: Sequence diagram showing the information exchange between *PhysioLabXR*'s threads and processes. The main process contains concurrent threads of the main GUI controller, serialization, data worker, and visualization. When the user adds a new data source, the GUI thread forks a data worker and a visualization thread. More demanding operations run on separate server processes, including replay and scripting. User commands such as *StartStreaming*, *StartReplay*, and *StopRecording* are passed from the main GUI to the corresponding threads or processes.

The architecture of *PhysioLabXR* follows rigorous software design patterns, minimizing maintenance efforts while maximizing scalability. Unit testing is at the core of our development process; each software feature, backend, and GUI frontend, is tested on both functionality and performance. The software is built using continuous integration: each commit to the main branch triggers a full test routine to ensure compatibility.

Limitation and Future Scope

PhysioLabXR aims to be a versatile platform for real-time experiments, primarily for, but not limited to, HCI and neuroscience. It is designed to be a community-driven project, with our core team of developers maintaining architectural integrity and functional correctness. At the same time, we welcome contributions from researchers and practitioners in related fields to build on this scaffolding and expand its capabilities. While the stream interface supports data sent through LSL or ZMQ, we are currently developing native plugins for sensors that lack network support. These plugins will enable the use of certain brands of fMRI, TMS (transcranial magnetic stimulation), and invasive neuroimaging devices (e.g., Neuropixels, Interuniversity Microelectronics Centre, 2023). We are also adding real-time analysis and processing modules for more modalities, such as real-time source localization for EEG and speech recognition for audio. Moreover, the current scripting interface is designed to provide maximum flexibility, thus requiring users to write Python code for their pipelines. In coming releases, we plan to make the scripting features more accessible to users with less programming experience by providing a visual programming interface and code generation.

Conclusion

PhysioLabXR is a pioneering tool in the era of spatial and physiological computing, addressing the increasing demand for a comprehensive software platform that drives multi-modal data integration and close-loop interaction. *PhysioLabXR* offers an array of interconnected functionalities, including visualization, recording, replay, real-time DSP modules, and a scripting interface. These all empower researchers and practitioners to explore novel experiment paradigms and design intricate feedback loops. With its Python-based frontend and C++-powered backend, the framework is built with scalability in mind while having optimized performance. The continued development of *PhysioLabXR* will be driven by the community and aims to fuel research insights into the intersection between the brain, behavior, and human-computer interaction.

Acknowledgments

This project was partly funded by the Columbia/ARL Human-Guided Intelligent Systems (HGIS) Program (W911NF-23-2-0067), a Vannevar Bush Faculty Fellowship from the US Department of Defense (N00014-20-1-2027) and a Center of Excellence grant from the Air Force Office of Scientific Research (FA9550-22-1-0337). This work was also supported by the Army Research Laboratory STRONG Program (W911NF-19-2-0135). We would like to express our gratitude for the support from our colleagues at the Laboratory for Intelligent Imaging and Neural Computing (LIINC), the Computer Graphics and User Interfaces (CGUI) Lab and the Artificial Intelligence for Vision Science (AI4VS) Lab at Columbia University, and Human-Computer Interaction (HCI) Lab at Worcester Polytechnic Institute. We would also like to thank all the community members who have contributed to *PhysioLabXR*.

Licensing and Availability

PhysioLabXR is an open-source project distributed under the BSD 3-Clause License. Researchers are welcome to modify the software to meet their specific needs and share their modifications with the community. We provide the following links to help access related resources:

- **Website:** The official *PhysioLabXR* website serves as a central hub for its information and updates. It can be accessed at https://www.physiolabxr.org.
- **Documentation:** The documentation includes tutorials for its features, tutorials, example use cases, and developer guides. It is hosted at https://physiolabxrdocs.readthedocs.io/en/latest/index.html. For testing and demonstration purposes, the documentation links to (otherwise unpublished) example recordings performed on co-authors of this manuscript.
- **GitHub Repository:** Users can access the repository at https://github.com/PhysioLabXR/PhysioLabXR. Users can submit bug reports and feature requests through the GitHub issue tracker.

References

Baltrušaitis, T., Robinson, P., & Morency, L.-P. (2016). Openface: An open source facial behavior analysis toolkit. *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 1–10. https://doi.org/10.1109/WACV.2016.7477553

Behnel, S., Bradshaw, R., Citro, C., Dalcin, L., Seljebotn, D. S., & Smith, K. (2010). Cython: The best of both worlds. *Computing in Science & Engineering*, *13*(2), 31–39. https://doi.org/10.1109/MCSE.2010.118

Esch, L., Sun, L., Klüber, V., Lew, S., Baumgarten, D., Grant, P. E., Okada, Y., Haueisen, J., Hämäläinen, M. S., & Dinh, C. (2018). MNE scan: Software for real-time processing of electrophysiological data. *Journal of Neuroscience Methods*, *303*, 55–67. https://doi.org/10.1016/j.jneumeth.2018.03.020

He, Z., Li, Z., Yang, F., Wang, L., Li, J., Zhou, C., & Pan, J. (2020). Advances in multimodal emotion recognition based on brain–computer interfaces. *Brain Sciences*, *10*(10), 687. https://doi.org/10.3390/brainsci10100687

iMotions. (2023). *iMotion*. https://imotions.com/

Interuniversity Microelectronics Centre. (2023). *Neuropixels*. https://www.neuropixels.org/

Jangraw, D. C., Johri, A., Gribetz, M., & Sajda, P. (2014). NEDE: An open-source scripting suite for developing experiments in 3D virtual environments. *Journal of Neuroscience Methods*, *235*, 245–251. https://doi.org/10.1016/j.jneumeth.2014.06.033

Koelstra, S., Muhl, C., Soleymani, M., Lee, J.-S., Yazdani, A., Ebrahimi, T., Pun, T., Nijholt, A., & Patras, I. (2011). Deap: A database for emotion analysis; using physiological signals. *IEEE Transactions on Affective Computing*, *3*(1), 18–31. https://doi.org/10.1109/T-AFFC.2011.15

Koorathota, S. C. (2023). *Multimodal deep learning systems for analysis of human behavior, preference, and state* [PhD thesis]. Columbia University.

Kothe, C., & Mandel, C. (n.d.). *A software framework for synchronizing a large array of data collection and stimulation devices*. https://github.com/sccn/labstreaminglayer

Lapborisuth, P., Koorathota, S., & Sajda, P. (2023). Pupil-linked arousal modulates network-level EEG signatures of attention reorienting during immersive multitasking. *Journal of Neural Engineering*. https://doi.org/10.1088/1741-2552/acf1cb

Lührs, M., & Goebel, R. (2017). Turbo-satori: A neurofeedback and brain–computer interface toolbox for real-time functional near-infrared spectroscopy. *Neurophotonics*, *4*(4), 041504–041504. https://doi.org/10.1117/1.NPh.4.4.041504

MacInnes, J. J., Adcock, R. A., Stocco, A., Prat, C. S., Rao, R. P., & Dickerson, K. C. (2020). Pyneal: Open source real-time fMRI software. *Frontiers in Neuroscience*, *14*, 900. https://doi.org/10.3389/fnins.2020.00900

MathWorks Inc. (2021). *MATLAB version: R2021b*. The MathWorks Inc. https://www.mathworks.com

Michalareas, G., Rudwan, I. M., Lehr, C., Gessini, P., Tavano, A., & Grabenhorst, M. (2022). A scalable and robust system for audience EEG recordings. *bioRxiv*, 2022–2012. https://doi.org/10.1101/2022.12.16.520764

Murphy, P. R., O'connell, R. G., O'sullivan, M., Robertson, I. H., & Balsters, J. H. (2014). Pupil diameter covaries with BOLD activity in human locus coeruleus. *Human Brain Mapping*, *35*(8), 4140–4154. https://doi.org/10.1002/hbm.22466

Neuropype. (2023). *Neuropype*. https://www.neuropype.io/

Nikolaev, A. R., Meghanathan, R. N., & Leeuwen, C. van. (2016). Combining EEG and eye movement recording in free viewing: Pitfalls and possibilities. *Brain and Cognition*, *107*, 55–83. https://doi.org/10.1016/j.bandc.2016.06.004

NIRx. (n.d.). https://www.nirx.net/.

OpenBCI. (n.d.). https://openbci.com/.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., & others. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., & others. (2011). Scikit-learn: Machine learning in python. *The Journal of Machine Learning Research*, *12*, 2825–2830.

Peirce, J. W. (2007). PsychoPy—psychophysics software in python. *Journal of Neuroscience Methods*, *162*(1-2), 8–13. https://doi.org/10.1016/j.jneumeth.2006.11.017

Rämä, P., & Baccino, T. (2010). Eye fixation-related potentials (EFRPs) during object identification. *Visual Neuroscience*, *27*(5-6), 187–192. https://doi.org/10.1017/S0952523810000283

Renard, Y., Lotte, F., Gibert, G., Congedo, M., Maby, E., Delannoy, V., Bertrand, O., & Lécuyer, A. (2010). Openvibe: An open-source software platform to design, test, and use brain–computer interfaces in real and virtual environments. *Presence*, *19*(1), 35–53. https://doi.org/10.1162/pres.19.1.35

Saffaryazdi, N., Gharibnavaz, A., & Billinghurst, M. (2022). Octopus sensing: A python library for human behavior studies. *Journal of Open Source Software*, *7*(71), 4045. https://doi.org/10.21105/joss.04045

Sollfrank, T., Ramsay, A., Perdikis, S., Williamson, J., Murray-Smith, R., Leeb, R., Millán, J., & Kübler, A. (2016). The effect of multimodal and enriched feedback on SMR-BCI performance. *Clinical Neurophysiology*, *127*(1), 490–498. https://doi.org/10.1016/j.clinph.2015.06.004

Srinath, K. (2017). Python–the fastest growing programming language. *International Research Journal of Engineering and Technology*, *4*(12), 354–357.

Tobii AB. (2023). *Tobii*. Tobii AB. https://www.tobii.com/

Unity Technologies. (2005). *Unity*. https://unity.com/

Wang, Q., Zhang, Q., Sun, W., Boulay, C., Kim, K., & Barmaki, R. L. (2023). A scoping review of the use of lab streaming layer framework in virtual and augmented reality research. *Virtual Reality*, 1–16. https://doi.org/10.1007/s10055-023-00799-8

Wei, H., Li, Z., Galvan, A. D., Su, Z., Zhang, X., Pahlavan, K., & Solovey, E. T. (2022). IndexPen: Two-finger text input with millimeter-wave radar. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, *6*(2), 1–39. https://doi.org/10.1145/3534601

ZeroMQ. (2021). *ZeroMQ - the intelligent transport layer*. https://zeromq.org/