Pybehave: a hardware agnostic, Python-based framework for controlling behavioral neuroscience experiments

Evan M. Dastin-van Rijn [1¶], **Joel Nielsen**[1], **Elizabeth M. Sachse** [1], **Christina Li**[1], **Megan E. Mensinger**[1], **Stefanie G. Simpson**[1], **Michelle C. Buccini**[1], **Francesca A. Iacobucci**[1], **David J. Titus** [1], and **Alik S. Widge** [1]

1 Department of Psychiatry and Behavioral Sciences, University of Minnesota Medical Center, Minneapolis, MN 55454, United States of America ¶ Corresponding author

Summary

This work presents our pybehave framework for developing behavioral tasks for use in experimental animal neuroscience. In contrast to other platforms, pybehave is built around a hardware-agnostic and highly object-oriented design philosophy. Pybehave separates code for task design from specific hardware implementations to streamline development, accessibility, and data sharing. This approach, combined with task-specific graphical user interfaces, expedites and simplifies the creation and visualization of complex behavioral tasks. User created task definition files can interact with hardware-specific source files, both written in Python. Any and all local configuration can be handled separately from the underlying task code.

Statement of need

Operant animal behavior training and monitoring is fundamental to scientific inquiry across fields (Krakauer et al., 2017). In many cases, a behavior of relevance, or its neural substrate, is best studied through a controlled laboratory task. These tasks require tight integration of the hardware components with which animals interact (IR beams, levers, lights, food dispensers, etc.) and the overarching software that coordinates these components to elicit desired behaviors. There are a plethora of options for systems to facilitate behavioral tasks, from commercial solutions (Panlab, Lafayette Instruments, Med Associates) to open-source packages (Akam et al., 2022; Dastin-van Rijn et al., 2023; Hwang et al., 2019) enabling a large variety of behavioral paradigms. Many of these systems are designed for the same behavioral paradigms with only slight differences in hardware, sensory modalities, or geometry. However, while the actual mechanics of these paradigms remain relatively similar, different solutions will often rely on vastly different software interfaces (Cardinal & Aitken, 2010; Lopes et al., 2015). Especially with commercial systems, behavioral tasks are often programmed in proprietary formats. This approach significantly raises the barrier to entry, leads to outdated software, and prevents sharing of tasks across labs.

Research in human behavior does not suffer from many of the aforementioned issues. Human behavioral tasks are generally run through a graphical interface implemented in a standard programming language like Python (Peirce et al., 2019), Javascript (Leeuw, 2015), or Matlab (Brainard, 1997). These tasks are readily compatible with most machines and are frequently shared between labs and used across multiple studies (Provenza et al., 2021). Protocols, data, and task code can be easily included in a manuscript and accessed and modified by future researchers. However, unlike experiments in animal behavior, human experiments rarely

require hardware beyond a monitor and standard input device (keyboard/mouse). Instead, most animal platforms, even from open source developers, restrict their software to certain types of hardware (Akam et al., 2022; Hwang et al., 2019). For example, `pycontrol` is only compatible with their companion microcontroller and input devices and `MonkeyLogic` can only communicate with DAQs manufactured by National Instruments. To address these limitations, we developed `pybehave` as a framework for abstracting standard hardware components to enable an implementation-independent format for developing and running behavioral tasks.

Benefits

Pybehave is a complete framework for building and running behavioral neuroscience experiments. It offers the following benefits: (1) hardware independence; (2) a flexible, programmatic system for developing tasks; (3) a highly extensible graphical interface for configuring and executing tasks; (4) options for task-specific visualization; (5) simultaneous control of multiple experiments; (6) options for locally configuring task variables and protocols; and (5) an extensive developer API, which allows users to extend the platform with tie-ins for custom hardware, event logging, or software connections.

Software Design Principles

To ensure flexibility while maintaining low-latency, pybehave is optimized through a combination of multiprocessing and multithreading along with separation of its features (events, hardware sources, tasks, etc.) into a modular software architecture. Additionally, pybehave uses two different GUI frameworks (QT and pygame) for user interfacing and task visualization/stimulus display respectively (Figure 1).
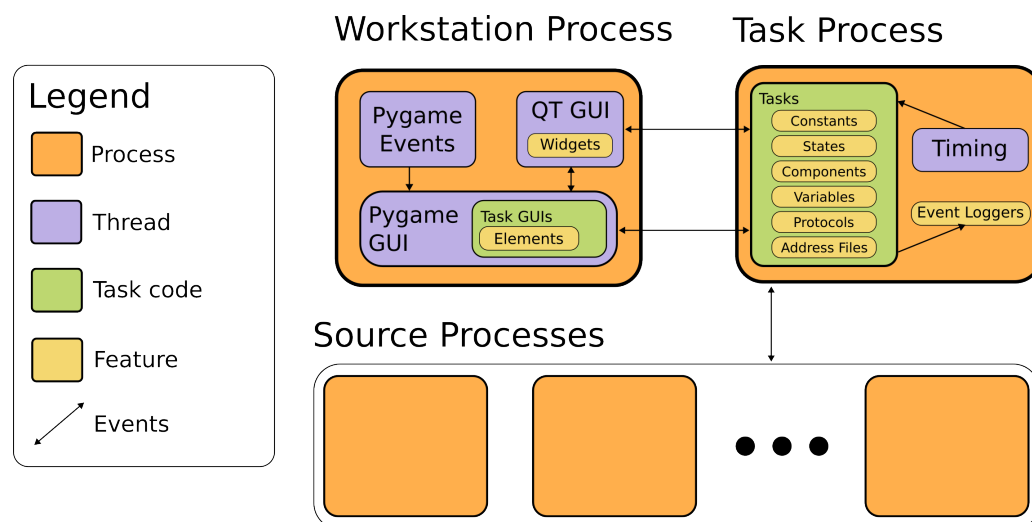

Figure 1: Framework diagram showing the information exchange between the `pybehave` threads and processes. The workstation process handles the interface and task GUIs. When Tasks are added from the workstation, they are initialized in the task process. Each Source with a connection to an external hardware or software system communicates with their `pybehave` equivalent in the Task process. All events sent between processes are mediated via inter-process communication over Pipes.

Tutorials and ongoing usage

A variety of tutorials are included in the repository aimed at all levels of usage, from technicians running tasks or analyzing behavioral data to developers aiming to build new tasks or integrate

additional hardware. Pybehave has already been applied to implement a variety of behavioral tasks which have been included in a separate repository for users to pull from directly or modify. These tasks are being run in a number of ongoing studies spanning standard operant conditioning (Dastin-van Rijn et al., 2023; Mensinger et al., 2023), evoked responses (Sachse et al., 2023), and video assays.

Acknowledgements

References

Akam, T., Lustig, A., Rowland, J. M., Kapanaiah, S. K., Esteve-Agraz, J., Panniello, M., Márquez, C., Kohl, M. M., Kätzel, D., Costa, R. M., & Walton, M. E. (2022). Open-source, Python-based, hardware and software for controlling behavioural neuroscience experiments. *eLife*, *11*, e67846. https://doi.org/10.7554/eLife.67846

Brainard, D. H. (1997). The Psychophysics Toolbox. *Spatial Vision*, *10*(4), 433–436. https://doi.org/10.1163/156856897X00357

Cardinal, R. N., & Aitken, M. R. F. (2010). Whisker: A client–server high-performance multimedia research control system. *Behavior Research Methods*, *42*(4), 1059–1071. https://doi.org/10.3758/BRM.42.4.1059

Dastin-van Rijn, E. M., Sachse, E., Iacobucci, F., Mensinger, M., & Widge, A. S. (2023). *OSCAR: An open-source controller for animal research*. bioRxiv. https://doi.org/10.1101/2023.02.03.527033

Hwang, J., Mitz, A. R., & Murray, E. A. (2019). NIMH MonkeyLogic: Behavioral control and data acquisition in MATLAB. *Journal of Neuroscience Methods*, *323*, 13–21. https://doi.org/10.1016/j.jneumeth.2019.05.002

Krakauer, J. W., Ghazanfar, A. A., Gomez-Marin, A., MacIver, M. A., & Poeppel, D. (2017). Neuroscience Needs Behavior: Correcting a Reductionist Bias. *Neuron*, *93*(3), 480–490. https://doi.org/10.1016/j.neuron.2016.12.041

Leeuw, J. R. de. (2015). jsPsych: A JavaScript library for creating behavioral experiments in a Web browser. *Behavior Research Methods*, *47*(1), 1–12. https://doi.org/10.3758/s13428-014-0458-y

Lopes, G., Bonacchi, N., Frazão, J., Neto, J. P., Atallah, B. V., Soares, S., Moreira, L., Matias, S., Itskov, P. M., Correia, P. A., Medina, R. E., Calcaterra, L., Dreosti, E., Paton, J. J., & Kampff, A. R. (2015). Bonsai: An event-based framework for processing and controlling data streams. *Frontiers in Neuroinformatics*, *9*. https://doi.org/10.3389/fninf.2015.00007

Mensinger, M., Wald, A., Sachse, E. M., Rijn, E. M. D., Reimer, A. E., & Widge, A. S. (2023). 462. Deep Brain Stimulation Does Not Affect Impulsivity in a Rodent 5-Choice Serial Reaction Time Task. *Biological Psychiatry*, *93*(9), S281–S282. https://doi.org/10.1016/j.biopsych.2023.02.702

Peirce, J., Gray, J. R., Simpson, S., MacAskill, M., Höchenberger, R., Sogo, H., Kastman, E., & Lindeløv, J. K. (2019). PsychoPy2: Experiments in behavior made easy. *Behavior*

Research Methods, *51*(1), 195–203. https://doi.org/10.3758/s13428-018-01193-y

Provenza, N. R., Gelin, L. F. F., Mahaphanit, W., McGrath, M. C., Dastin-van Rijn, E. M., Fan, Y., Dhar, R., Frank, M. J., Restrepo, M. I., Goodman, W. K., & Borton, D. A. (2021). Honeycomb: A template for reproducible psychophysiological tasks for clinic, laboratory, and home use. *Brazilian Journal of Psychiatry*, *44*, 147–155. https://doi.org/10.1590/1516-4446-2020-1675

Sachse, E., Rijn, E. M. D., Mensinger, M. E., Iacobucci, F. A., Reimer, A. E., & Widge, A. S. (2023). 534. Optogenetic Deep Brain Stimulation of mPFC Axons in Mid-Striatum Improves Cognitive Flexibility. *Biological Psychiatry*, *93*(9), S310. https://doi.org/10.1016/j.biopsych.2023.02.774