








lys_instr: A Python Package for Automating Scientific Measurements

Ziqian Wang^{1,2}, Hidenori Tsuji², Toshiya Shiratori³, and Asuka Nakamura^{2,3}

¹ Research Institute for Quantum and Chemical Innovation, Institutes of Innovation for Future Society, Nagoya University, Japan  ² RIKEN Center for Emergent Matter Science, Japan  ³ Department of Applied Physics, The University of Tokyo, Japan   Corresponding author

DOI: [10.xxxxxx/draft](https://doi.org/10.xxxxxx/draft)

Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [Open Journals](#) 

Reviewers:

- [@openjournals](#)

Submitted: 01 January 1970

Published: unpublished

License

Authors of papers retain copyright and release the work under a

Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))

Summary

Modern experiments increasingly demand automation frameworks capable of coordinating diverse scientific instruments while remaining flexible and easy to customize. Existing solutions, however, often require substantial adaptation or manual handling of low-level communication and threading. We present `lys_instr`, a Python package that addresses these challenges through an object-oriented, multi-layered architecture for instrument control, workflow coordination, and GUI construction. It enables researchers to rapidly build responsive, asynchronous measurement systems with minimal coding effort. Seamlessly integrated with the `lys` platform (Nakamura, 2023), `lys_instr` unifies experiment control, data acquisition, and visualization, offering an efficient foundation for next-generation, automation-driven experimental research.

Statement of need

Modern scientific research increasingly relies on comprehensive measurements across wide parameter spaces to fully understand physical phenomena. As experiments grow in complexity—with longer measurement times and a greater diversity of instruments—efficient automation has become essential. Measurement automation is now evolving beyond simple parameter scans toward informatics-driven, condition-based optimization, paving the way for AI-assisted experimental workflow management. This progress demands robust software infrastructure capable of high integration and flexible logic control.

However, building such a system remains nontrivial for researchers. At the low level, specific instrument methods tightly coupled to diverse communication protocols (e.g., TCP/IP, VISA, serial, etc.) limit interchangeability and flexibility across systems. At the high level, coordinating workflows involving conditional logic, iterative processes, and advanced algorithms from different libraries can lead to redundant implementations of similar functionality across different contexts. Moreover, designing graphical user interfaces (GUIs) for these low- and high-level functionalities typically involves complex multithreading, which requires familiarity with GUI libraries and the underlying operating system (OS) event-handling mechanisms. Existing frameworks such as QCoDeS (Nielsen et al., 2025), PyMeasure (PyMeasure Developers, 2024), PyLabControl (Steiner & LISE-B26, 2024), LabVIEW (National Instruments Corporation, 2024), and MATLAB's Instrument Control Toolbox (The MathWorks, Inc., 2024) provide powerful ecosystems for instrument control and measurement scripting, but require users to handle low-level communications and high-level workflow logic themselves. These challenges impose substantial overhead on researchers designing custom measurement systems.

To address these issues, we introduce `lys_instr`—an object-oriented framework that abstracts common control patterns from experiment-specific implementations, reducing coding and

design costs while enabling flexible and efficient automation.

Design philosophy

lys_instr adopts a three-layer architecture organized by functional separation: Base Layer for device controller abstraction, Top Layer for workflow coordination, and Connection Layer in between for complete control-system assembly (Figure 1). Each layer applies object-oriented design patterns from GoF (Gamma et al., 1994), according to its responsibilities, enhancing flexibility, modularity, and usability. The framework builds on the lys platform, leveraging its powerful multidimensional data visualization capabilities.

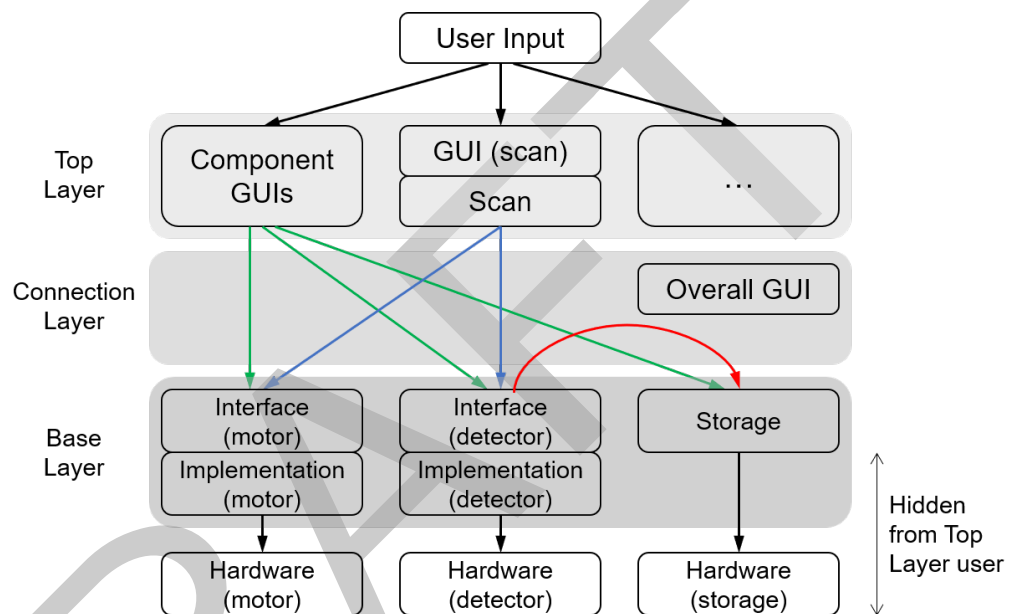


Figure 1: Schematic of the code architecture of lys_instr.

1. Base Layer: Device Controller Abstraction

This layer defines abstract interfaces that standardize core instrument controllers. The interfaces encapsulate the concrete implementations, following the *Template Method* design pattern. Typically, most measurement systems include two types of components: *controllers*, which adjust experimental parameters such as external fields, temperature, or physical positions, and *detectors*, which record experimental data, e.g., cameras, spectrometers. Accordingly, lys_instr provides standardized *controller* and *detector* interfaces that unify instrument behavior, allowing higher layers to operate on different devices uniformly through common interfaces. Users only need to provide device-specific subclasses that inherit from these interfaces to handle communication with their respective hardware devices. Moreover, each interface manages its own thread(s), ensuring responsiveness and asynchronous operation without blocking other controllers or the GUIs in higher layers. This structure enables users to create controller objects that can be readily integrated into higher-level workflows with minimal device-specific coding.

2. Top Layer: Workflow Coordination

This layer implements workflows common across many setups. Most measurements share similar procedural structures, such as a *scan* process in which data are sequentially recorded while parameters like fields, temperature, or positions are varied. These workflows are standardized using the abstract interfaces defined in the Base Layer, independent of any specific hardware

69 devices, following the *Bridge* and *Composite* design patterns. For example, `lys_instr` provides
70 a standardized *scan* routine that calls *controller* and *detector* interface methods without
71 requiring knowledge of the underlying concrete implementations. This abstraction allows such
72 workflows to be reused across different hardware configurations, greatly improving coding
73 efficiency. In addition, `lys_instr` includes prebuilt GUI components corresponding to each
74 Base Layer component, enabling direct GUI-based control through the same abstract methods.
75 This design cleanly separates workflow logic from device-specific details, simplifying extension
76 to complex measurement systems. Moreover, the GUI communicates with Base Layer interfaces
77 via event-driven messaging, following the *Observer* design pattern to ensure low coupling
78 and high extensibility. With this layer, users can design measurement workflows from scratch
79 without manually creating GUI components.

80 3. Connection Layer: Control-System Assembly

81 This layer enables flexible assembly of components from the Base and Top Layers into a
82 complete control system by managing connections within and across layers. Following the
83 *Mediator* design pattern, it connects abstract Base Layer interfaces (and, through them, the
84 corresponding hardware devices) to enable automatic data flow, and links GUI components
85 to their respective interfaces, fully hiding device-specific implementations from this layer and
86 above. It also organizes the GUI components into a cohesive application for user interaction.
87 This design grants users maximum freedom to construct tailored control systems without
88 handling low-level tasks such as inter-device communication or multi-threading. Several prebuilt
89 GUI templates for common scenarios are provided for quick hands-on use.

90 Overall, `lys_instr` provides prebuilt support for standard device controllers, common experi-
91 mental workflows, and GUI components and assemblies, so users generally need to implement
92 only device-specific subclasses to handle communication with their hardware. This enables
93 rapid integration of new instruments into automated measurement workflows with minimal
94 coding and design effort.

95 Example of Constructed GUI

96 With `lys_instr`, users can easily construct a GUI like the one shown in [Figure 2](#). In this
97 example, the `lys_instr` window is embedded in the `lys` platform, with Sector A for storage,
98 Sector B for detector, and Sector C for controllers. Multi-dimensional, nested scan sequences
99 can be defined via the visual interface in the Scan tab in Sector C. `lys` tools in the outer
100 window tabs allow customization of data display, enabling advanced, on-the-fly customization
101 of data visualization.

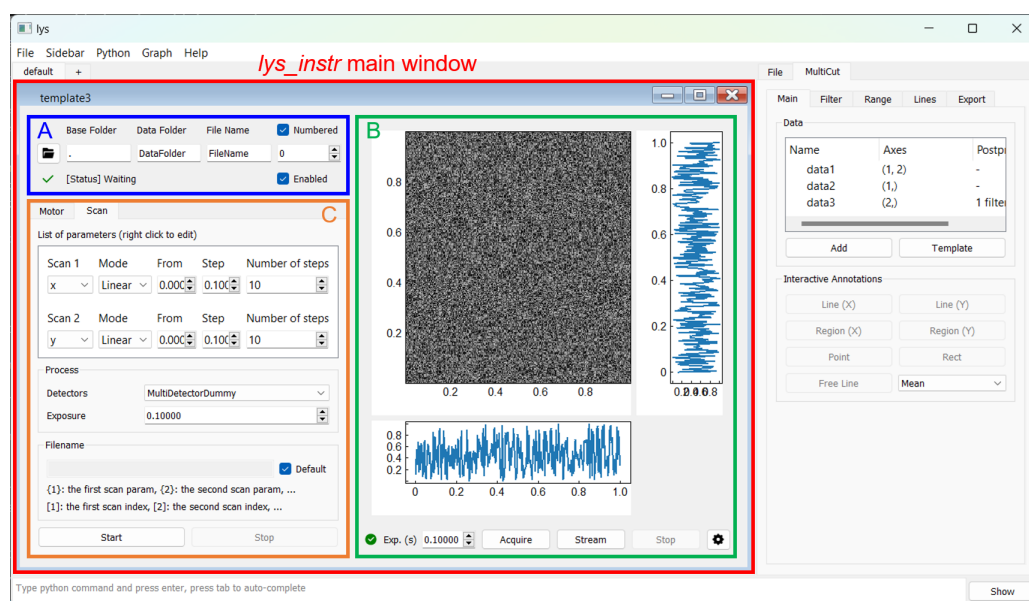


Figure 2: Example GUI of `lys_instr`. The main window, embedded in the `lys` window, contains three sectors: Storage panel (A), Detector panel (B), and controller panel (C). The Scan tab in (C) enables dynamic configuration of multi-dimensional, nested experimental workflows.

Projects using the software

`lys_instr` has been deployed in complex, real-world scientific instruments, supporting multiple peer-reviewed publications. It automates ultrafast electron diffraction and transmission electron microscopy systems, coordinating ultrafast laser excitation and pulsed electron beam detection in pump-probe experiments (Koga et al., 2024; Nakamura et al., 2020, 2021, 2022, 2023; Shimojima et al., 2021, 2023a, 2023b). It enables precise control of electromagnetic lenses and electron deflectors for advanced microscopy involving electron-beam precession, a capability that would be difficult to achieve without `lys_instr` (Hayashi et al., 2025; Shiratori et al., 2024).

Acknowledgements

We acknowledge valuable comments from Takahiro Shimojima and Kyoko Ishizaka. This work was partially supported by Grant-in-Aid for Scientific Research (KAKENHI) (Grant No.25K00057) JST PRESTO Grant No.JPMJPR24JA.

References

- Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.
- Hayashi, S., Han, D., Tsuji, H., Ishizaka, K., & Nakamura, A. (2025). Development of precession lorentz transmission electron microscopy. *arXiv Preprint arXiv:2505.05790*. <https://arxiv.org/abs/2505.05790>
- Koga, J., Chiashi, Y., Nakamura, A., Akiba, T., Takahashi, H., Shimojima, T., Ishiwata, S., & Ishizaka, K. (2024). Unusual photoinduced crystal structure dynamics in TaTe₂ with double zigzag chain superstructure. *Applied Physics Express*, 17(4), 042007. <https://doi.org/10.35848/1882-0786/ad3b61>

- 125 Nakamura, A. (2023). *lys*: Interactive Multi-Dimensional Data Analysis and Visualization
 126 Platform. *Journal of Open Source Software*, 8(92), 5869. [https://doi.org/10.21105/joss.](https://doi.org/10.21105/joss.05869)
 127 [05869](https://doi.org/10.21105/joss.05869)
- 128 Nakamura, A., Shimojima, T., Chiashi, Y., Kamitani, M., Sakai, H., Ishiwata, S., Li, H., &
 129 Ishizaka, K. (2020). Nanoscale Imaging of Unusual Photoacoustic Waves in Thin Flake
 130 VTe₂. *Nano Letters*, 20(7), 4932–4938. <https://doi.org/10.1021/acs.nanolett.0c01006>
- 131 Nakamura, A., Shimojima, T., & Ishizaka, K. (2021). Finite-element Simulation of Pho-
 132 toinduced Strain Dynamics in Silicon Thin Plates. *Structural Dynamics*, 8(2), 024103.
 133 <https://doi.org/10.1063/4.0000059>
- 134 Nakamura, A., Shimojima, T., & Ishizaka, K. (2022). Visualizing Optically-Induced Strains
 135 by Five-Dimensional Ultrafast Electron Microscopy. *Faraday Discussions*, 237, 27–39.
 136 <https://doi.org/10.1039/D2FD00062H>
- 137 Nakamura, A., Shimojima, T., & Ishizaka, K. (2023). Characterizing an Optically Induced
 138 Sub-micrometer Gigahertz Acoustic Wave in a Silicon Thin Plate. *Nano Letters*, 23(7),
 139 2490–2495. <https://doi.org/10.1021/acs.nanolett.2c03938>
- 140 National Instruments Corporation. (2024). *LabVIEW*. National Instruments Corporation.
 141 <https://www.ni.com/en-us/shop/labview.html>
- 142 Nielsen, J. H., Astafev, M., Nielsen, W. H. P., & others. (2025). *QCoDeS: Modular data*
 143 *acquisition framework* (Version v0.54.2). <https://doi.org/10.5281/zenodo.17459861>
- 144 PyMeasure Developers. (2024). *PyMeasure: Scientific measurement library for instruments,*
 145 *experiments, and live-plotting*. (Version 0.14.0). Zenodo. [https://doi.org/10.5281/zenodo.](https://doi.org/10.5281/zenodo.11241567)
 146 [11241567](https://doi.org/10.5281/zenodo.11241567)
- 147 Shimojima, T., Nakamura, A., & Ishizaka, K. (2023a). Development of Five-Dimensional
 148 Scanning Transmission Electron Microscopy. *Review of Scientific Instruments*, 94(2),
 149 023705. <https://doi.org/10.1063/5.0106517>
- 150 Shimojima, T., Nakamura, A., & Ishizaka, K. (2023b). Development and Applications
 151 of Ultrafast Transmission Electron Microscopy. *Microscopy*, 72(4), 287–298. <https://doi.org/10.1093/jmicro/dfad021>
- 152
- 153 Shimojima, T., Nakamura, A., Yu, X., Karube, K., Taguchi, Y., Tokura, Y., & Ishizaka, K.
 154 (2021). Nano-to-Micro Spatiotemporal Imaging of Magnetic Skyrmion's Life Cycle. *Science*
 155 *Advances*, 7(25), eabg1322. <https://doi.org/10.1126/sciadv.abg1322>
- 156 Shiratori, T., Koga, J., Shimojima, T., Ishizaka, K., & Nakamura, A. (2024). Development of
 157 ultrafast four-dimensional precession electron diffraction. *Ultramicroscopy*, 267, 114064.
 158 <https://doi.org/10.1016/j.ultramic.2024.114064>
- 159 Steiner, J. F., & LISE-B26. (2024). *PyLabControl: Laboratory equipment control for scientific*
 160 *experiments*. (Version 1.3.0). <https://github.com/LISE-B26/pylabcontrol>
- 161 The MathWorks, Inc. (2024). *MATLAB instrument control toolbox*. The MathWorks, Inc.
 162 <https://www.mathworks.com/products/instrument.html>