

¹ lys_instr: A Python Package for Automating Scientific Measurements

³ **Ziqian Wang**  ^{1,2}¶, **Hidenori Tsuji**², **Toshiya Shiratori**  ³, and **Asuka Nakamura**  ^{2,3}

⁵ 1 Research Institute for Quantum and Chemical Innovation, Institutes of Innovation for Future Society,
⁶ Nagoya University, Japan  2 RIKEN Center for Emergent Matter Science, Japan  3 Department
⁷ 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Published: unpublished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 ¹⁹ Creative Commons Attribution 4.0 International License ([CC BY 4.0](#))²⁰

⁸ Summary

⁹ Modern experiments increasingly demand automation frameworks that coordinate diverse
¹⁰ scientific instruments while remaining flexible and customizable. Existing solutions, however,
¹¹ often require explicit management of low-level communication and concurrency, resulting in
¹² substantial development overhead. We present lys_instr, a Python package that addresses
¹³ these challenges through an object-oriented, multi-layered architecture for instrument control,
¹⁴ workflow coordination, and GUI construction. It enables researchers to rapidly build responsive,
¹⁵ asynchronous measurement systems with minimal coding effort. Seamlessly integrated with
¹⁶ the lys platform ([Nakamura, 2023](#)), lys_instr unifies experiment control, data acquisition,
¹⁷ and visualization, providing an efficient foundation for next-generation automation-driven
¹⁸ experimental research.

²¹ Statement of need

²¹ Modern scientific research increasingly relies on comprehensive measurements across wide parameter spaces to elucidate physical phenomena. As experiments grow in complexity—with longer measurement times and greater instrument diversity—efficient automation has become essential. Measurement automation is evolving beyond simple parameter scans toward informatics-driven, condition-based optimization, paving the way for AI-assisted experimental workflow management. This shift demands robust software infrastructure with high integrability and flexible logic control.

²⁷ However, building such a measurement system remains time-consuming for researchers. At the low level, instruments are often tightly coupled to diverse communication protocols (e.g., TCP/IP, VISA, serial), limiting interchangeability and cross-system flexibility. At the high level, coordinating workflows that combine conditional logic, iterative processes, and advanced algorithms across multiple libraries frequently leads to redundant implementations, reducing development efficiency. For example, capturing temperature-dependent images with a camera or acquiring temperature-dependent spectra are routine tasks. Although the underlying workflow—iterative parameter adjustment followed by data acquisition—is conceptually identical, such logic is often reimplemented independently across experiments. Moreover, implementing graphical user interfaces (GUIs) for these low- and high-level functionalities typically involves complex multithreading, requiring familiarity with GUI frameworks and operating system (OS) event-handling mechanisms. These challenges impose substantial development overhead and highlight the need for a control framework that balances architectural flexibility with reduced implementation complexity.

41 State of the field

42 In principle, the challenges in measurement system development described above can be
43 addressed by frameworks that adopt well-known object-oriented GoF design patterns ([Gamma](#)
44 [et al., 1994](#)). Encapsulating low-level communication protocols behind standardized interfaces
45 enables the development of high-level workflows that are independent of specific instruments,
46 while still allowing device-specific customization through inheritance. Such loose coupling
47 improves reusability of high-level logic across different experimental setups. Furthermore,
48 the *Template Method* design pattern allows complex multithreading-related functionalities
49 to be implemented within superclass definitions, enabling users to write measurement logic
50 without explicitly handling thread management. Reusable GUI components can likewise be
51 constructed on top of these abstract interfaces, significantly reducing implementation effort.
52 This design philosophy also enhances stability, as most components can be developed and
53 tested independently of physical hardware.

54 However, existing software platforms do not explicitly adopt this interface-centered design
55 philosophy. Commercial platforms such as LabVIEW ([National Instruments Corporation,](#)
56 [2024](#)) and MATLAB's Instrument Control Toolbox ([The MathWorks, Inc., 2024](#)) provide
57 mature environments for instrument communication, workflow execution, and GUI development.
58 Python-based frameworks including QCoDeS ([Nielsen et al., 2025](#)), PyMeasure ([PyMeasure](#)
59 [Developers, 2024](#)), and PyOpticon ([Randall & Majumdar, 2025](#)) likewise offer instrument
60 drivers, experiment routines, and graphical components. While these tools provide powerful
61 capabilities, flexible workflow orchestration—particularly for conditional logic, nested procedures,
62 and multithreaded execution—often requires substantial user-defined implementation. In many
63 cases, these platforms function either as general-purpose programming environments or as
64 collections of concrete drivers and predefined workflows, rather than as a framework that
65 defines a unified set of abstract interfaces for measurement systems. Therefore, development
66 of an interface-driven architecture grounded in object-oriented design patterns is essential for
67 building reusable, low-code, flexible, and stable measurement systems.

68 Software design

69 To address this gap, `lys_instr` introduces a layered architecture that spans low-level instrument
70 interfaces to high-level workflow and GUI integration. Most importantly, it defines a unified
71 set of abstract interfaces in the lowest layer that capture the common functionalities shared by
72 many scientific instruments. This design enables the reuse of high-level GUIs and workflow
73 logic, ranging from simple parameter sweeps to informatics-driven adaptive experiments,
74 as discussed above. `lys_instr` adopts a three-layer architecture organized by functional
75 separation ([Figure 1](#)). Each layer applies object-oriented design patterns described by Gamma
76 et al. ([Gamma et al., 1994](#)) according to its responsibilities, thereby enhancing flexibility,
77 modularity, and usability.

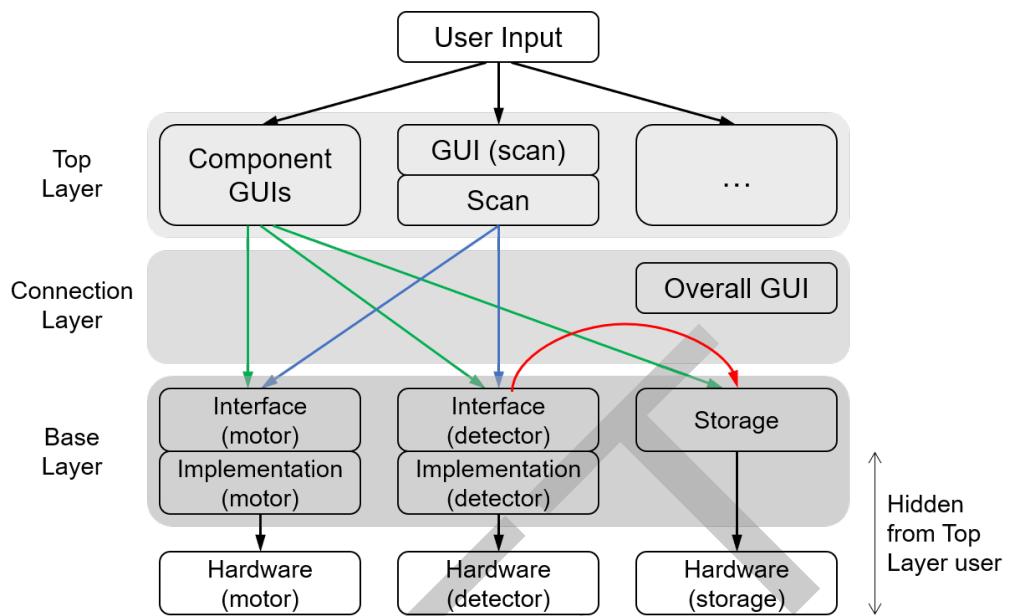


Figure 1: Schematic of the code architecture of lys_instr.

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

108 3. Connection Layer: Control-System Assembly

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

118 Overall, `lys_instr` provides prebuilt support for standard device controllers, common
 119 experimental workflows, and GUI components and assemblies, so users generally need to
 120 implement only device-specific subclasses to handle communication with their hardware. This
 121 enables rapid integration of new instruments into automated measurement workflows with
 122 minimal coding and design effort. A potential limitation of this architecture is that highly
 123 unconventional or non-standard measurement workflows may require customization beyond
 124 the predefined abstractions. However, the layered interface design covers the vast majority of
 125 multi-parameter experimental scenarios encountered in typical laboratory environments.

126 Example of constructed GUI

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

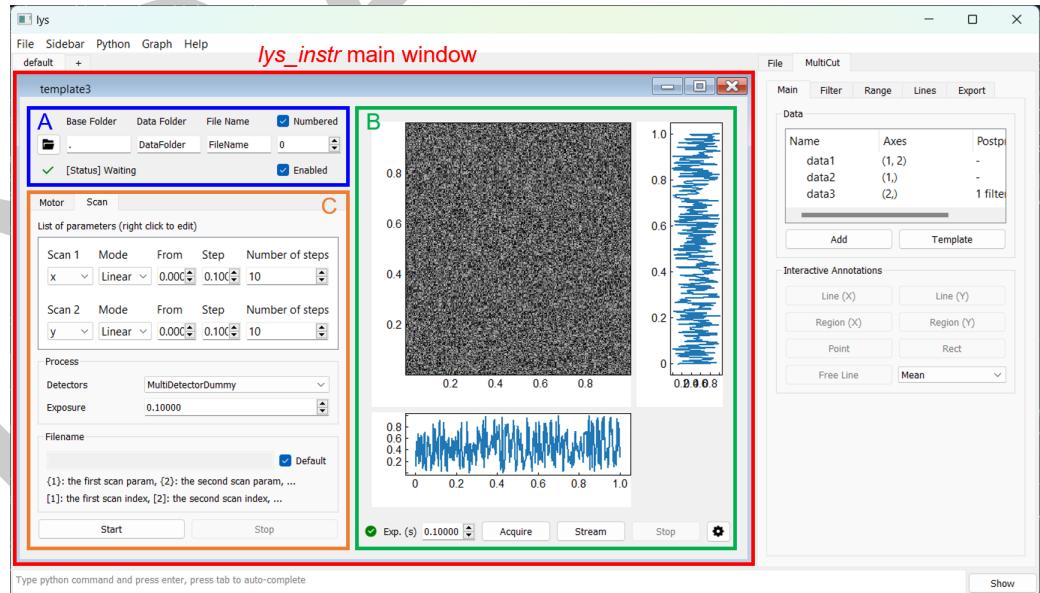


Figure 2: Example GUI of `lys_instr`. The main window, embedded in the `lys` window, contains three sectors: Storage panel (A), Detector panel (B), and controller panel (C). The Scan tab in (C) enables dynamic configuration of multi-dimensional, nested experimental workflows.

133 Research impact statement

134 lys_instr has been deployed in complex, real-world scientific experiments and has supported
135 multiple peer-reviewed publications. It automates ultrafast electron diffraction (UED) and
136 ultrafast transmission electron microscopy (UTEM) systems, coordinating ultrafast laser
137 excitation and pulsed electron beam detection in pump-probe experiments (Koga et al., 2024;
138 Nakamura et al., 2020, 2021, 2022, 2023; Shimojima et al., 2021, 2023a, 2023b). It enables
139 precise control of electromagnetic lenses and electron deflectors for advanced microscopy
140 involving electron-beam precession, a capability that would be difficult to implement without
141 lys_instr (Hayashi et al., 2026; Shiratori et al., 2024).

142 The software has demonstrated seamless control of transmission electron microscopes from
143 multiple manufacturers across different institutes, including RIKEN Center for Emergent Matter
144 Science and Nagoya University, illustrating reproducible performance and hardware-independent
145 workflow management. Through integration with sister packages in the lys family, including
146 lys_em (lys_em Developers, 2026) and lys_fem (lys_fem Developers, 2026), lys_instr
147 supports complex multi-instrument automation within a research-driven ecosystem, enabling
148 efficient deployment of advanced workflows while preserving modularity and extensibility.

149 AI usage disclosure

150 Generative AI tools were used to provide debugging suggestions during the final stages of
151 software development. All code was implemented, reviewed, and verified on real hardware,
152 with functionality confirmed through unit tests and experimental validation.

153 Acknowledgements

154 We acknowledge valuable comments from Takahiro Shimojima and Kyoko Ishizaka. This
155 work was partially supported by Grant-in-Aid for Scientific Research (KAKENHI) Grants
156 No. 21K13889 and No. 25K00057, and JST PRESTO Grant No. JPMJPR24JA.

157 References

- 158 Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of*
159 *reusable object-oriented software*. Addison-Wesley. ISBN: 978-0201633610
- 160 Hayashi, S., Han, D., Tsuji, H., Ishizaka, K., & Nakamura, A. (2026). Development of
161 precession lorentz transmission electron microscopy. *Ultramicroscopy*, 280, 114276. <https://doi.org/10.1016/j.ultramic.2025.114276>
- 162 Koga, J., Chiashi, Y., Nakamura, A., Akiba, T., Takahashi, H., Shimojima, T., Ishiwata,
163 S., & Ishizaka, K. (2024). Unusual photoinduced crystal structure dynamics in TaTe₂
164 with double zigzag chain superstructure. *Applied Physics Express*, 17(4), 042007. <https://doi.org/10.35848/1882-0786/ad3b61>
- 165 lys_em Developers. (2026). *Lys_em: Python package for electron microscopy control and*
166 *data analysis*. https://github.com/a-tock/lys_em
- 167 lys_fem Developers. (2026). *Lys_fem: Python package for finite-element modeling and*
168 *experiment integration*. https://github.com/lys-devel/lys_fem
- 169 Nakamura, A. (2023). lys: Interactive Multi-Dimensional Data Analysis and Visualization
170 Platform. *Journal of Open Source Software*, 8(92), 5869. <https://doi.org/10.21105/joss.05869>

- 174 Nakamura, A., Shimojima, T., Chiashi, Y., Kamitani, M., Sakai, H., Ishiwata, S., Li, H., &
175 Ishizaka, K. (2020). Nanoscale Imaging of Unusual Photoacoustic Waves in Thin Flake
176 VTe2. *Nano Letters*, 20(7), 4932–4938. <https://doi.org/10.1021/acs.nanolett.0c01006>
- 177 Nakamura, A., Shimojima, T., & Ishizaka, K. (2021). Finite-element Simulation of
178 Photoinduced Strain Dynamics in Silicon Thin Plates. *Structural Dynamics*, 8(2), 024103.
179 <https://doi.org/10.1063/4.0000059>
- 180 Nakamura, A., Shimojima, T., & Ishizaka, K. (2022). Visualizing Optically-Induced Strains
181 by Five-Dimensional Ultrafast Electron Microscopy. *Faraday Discussions*, 237, 27–39.
182 <https://doi.org/10.1039/D2FD00062H>
- 183 Nakamura, A., Shimojima, T., & Ishizaka, K. (2023). Characterizing an Optically Induced
184 Sub-micrometer Gigahertz Acoustic Wave in a Silicon Thin Plate. *Nano Letters*, 23(7),
185 2490–2495. <https://doi.org/10.1021/acs.nanolett.2c03938>
- 186 National Instruments Corporation. (2024). *LabVIEW*. National Instruments Corporation.
187 <https://www.ni.com/en-us/shop/labview.html>
- 188 Nielsen, J. H., Astafev, M., Nielsen, W. H. P., & others. (2025). *QCoDeS: Modular data
189 acquisition framework* (Version v0.54.2). <https://doi.org/10.5281/zenodo.17459861>
- 190 PyMeasure Developers. (2024). *PyMeasure: Scientific measurement library for instruments,
191 experiments, and live-plotting*. (Version 0.14.0). Zenodo. <https://doi.org/10.5281/zenodo.11241567>
- 193 Randall, R., & Majumdar, A. (2025). PyOpticon: An open-source python package for laboratory
194 control, automation, and visualization. *Chemistry of Materials*, 37(13), 4585–4592. <https://doi.org/10.1021/acs.chemmater.5c00644>
- 196 Shimojima, T., Nakamura, A., & Ishizaka, K. (2023a). Development of Five-Dimensional
197 Scanning Transmission Electron Microscopy. *Review of Scientific Instruments*, 94(2),
198 023705. <https://doi.org/10.1063/5.0106517>
- 199 Shimojima, T., Nakamura, A., & Ishizaka, K. (2023b). Development and Applications
200 of Ultrafast Transmission Electron Microscopy. *Microscopy*, 72(4), 287–298. <https://doi.org/10.1093/jmicro/dfad021>
- 202 Shimojima, T., Nakamura, A., Yu, X., Karube, K., Taguchi, Y., Tokura, Y., & Ishizaka, K.
203 (2021). Nano-to-Micro Spatiotemporal Imaging of Magnetic Skyrmion's Life Cycle. *Science
204 Advances*, 7(25), eabg1322. <https://doi.org/10.1126/sciadv.abg1322>
- 205 Shiratori, T., Koga, J., Shimojima, T., Ishizaka, K., & Nakamura, A. (2024). Development of
206 ultrafast four-dimensional precession electron diffraction. *Ultramicroscopy*, 267, 114064.
207 <https://doi.org/10.1016/j.ultramic.2024.114064>
- 208 The MathWorks, Inc. (2024). *MATLAB instrument control toolbox*. The MathWorks, Inc.
209 <https://www.mathworks.com/products/instrument.html>