SPHERExLabTools (SLT): A Python Data Acquisition System for SPHEREx Characterization and Calibration

Sam Condon\textsuperscript{a}, Marco Viero\textsuperscript{a}, James Bock\textsuperscript{a, b}, Howard Hui\textsuperscript{a}, Phil Korngut\textsuperscript{a}, Hiromasa Miyasaka\textsuperscript{a}, Ken Manatt\textsuperscript{b}, Chi Nguyen\textsuperscript{a}, Hien Nguyen\textsuperscript{b}, and Steve Padin\textsuperscript{a}

\textsuperscript{a}California Institute of Technology, Pasadena, CA, 91125, USA
\textsuperscript{b}Jet Propulsion Laboratory, California Institute of Technology, Pasadena, CA, 91109, USA

ABSTRACT

Selected as the next NASA Medium Class Explorer mission, SPHEREx, the Spectro-Photometer for the History of the Universe, Epoch of Reionization, and Ices Explorer is planned for launch in early 2025. SPHEREx calibration data products include detector spectral response, non-linearity, persistence, and telescope focus error measurements. To produce these calibration products, we have developed a dedicated data acquisition and instrument control system, SPHERExLabTools (SLT). SLT implements driver-level software for control of all testbed instrumentation, graphical interfaces for control of instruments and automated measurements, real-time data visualization, processing, and data archival tools for a variety of output file formats. This work outlines the architecture of the SLT software as a framework for general purpose laboratory data acquisition and instrument control. Initial SPHEREx calibration products acquired while using SLT are also presented.

Keywords: SPHEREx, All-sky survey, ground-support-equipment, data-acquisition

1. INTRODUCTION

The SPHEREx mission will perform an all-sky spectrophotometric survey in the near-infrared (0.75 $\mu$m - 5.0 $\mu$m.) to address major scientific questions consistent with all three major themes of NASA’s astrophysics program. Specifically, SPHEREx will:

1. \textit{Probe the origin and destiny of our Universe.} By measuring galaxy redshifts over a large cosmological volume, SPHEREx provides constraints on the physics of inflation, the superluminal expansion of the Universe that took place $10^{-32}$ s after the Big Bang.

2. \textit{Explore whether planets around other stars could harbor life.} SPHEREx provides absorption spectra of early phase planetary systems in the Milky Way to determine the abundance and composition of biogenic ices.

3. \textit{Explore the origin and evolution of galaxies.} SPHEREx will permit a precise determination of the power spectrum of spatial fluctuations in the extragalactic background light intensity through deep images formed near the ecliptic poles.

SPHEREx instrument parameters are listed in table 1. Previous SPIE proceedings\textsuperscript{1} and\textsuperscript{2} provide detailed information about the scientific rationale, hardware configuration, and mission design of SPHEREx.

By mid-2022, SPHEREx is in its optical payload integration and testing phase to deliver several laboratory calibration and instrument characterization measurements. Two such measurements are discussed in this paper: absolute spectral and focus calibration. To accurately determine photometric redshifts and resolve ice absorption features in the spectra of planetary systems, precise spectral bandpass profiles for all 25 million pixels in SPHEREx must be determined through spectral calibration. To maximize the fraction of background sky pixels and enable source masking to deeper flux levels for extragalactic background light power spectrum measurements, the SPHEREx point-spread-function (PSF) must be minimized through focus calibration. A system capable of performing these characterization/calibration measurements must meet the requirements listed in Table 2.
Table 1: SPHEREx Instrument Parameters

| Parameter                                      | Value                                      |
|-----------------------------------------------|--------------------------------------------|
| Effective Aperture                            | 20 cm                                      |
| Pixel Size                                    | 6.2" x 6.2"                                |
| Field of View                                  | 2 x (3.5' x 11.3'); dichroic               |
| Resolving Power and Wavelength Coverage       | \(\lambda = 0.75 - 2.42 \mu m; R = 41\)   |
|                                               | \(\lambda = 2.42 - 3.82 \mu m; R = 35\)   |
|                                               | \(\lambda = 3.82 - 4.42 \mu m; R = 110\)  |
|                                               | \(\lambda = 4.42 - 5.00 \mu m; R = 130\)  |
| Arrays                                        | 3x Teledyne Hawaii-2RG 2.5 \(\mu m\).     |
|                                               | 3x Teledyne Hawaii-2RG 5.3 \(\mu m\).     |
| Cooling                                       | All-Passive                                |
| 2.5 \(\mu m\). Array Temperature             | < 80K                                      |
| 5.3 \(\mu m\). Array Temperature             | < 55K                                      |

Table 2: SPHEREx characterization/calibration system requirements.

| Requirement                  | Description                                                                 |
|------------------------------|-----------------------------------------------------------------------------|
| 1) Instrument driver framework | A convenient abstraction away from low-level instrument communication details. |
| 2) Manual instrument control | Interfaces to control instruments in a testbed to facilitate the setup of automated measurements. |
| 3) Measurement automation   | A script based framework for specifying automated measurements.              |
| 4) Real-time data visualization and processing | Live plots of multi-channel time-stream data and image feeds. Basic real-time processing such as averaging and histogram display. |
| 5) Data archival tools       | Mechanisms to archive data products along with associated metadata in a variety of file formats. |
| 6) Configurability           | A method to reconfigure all aforementioned components for a variety of measurements. |

The development of a software system meeting the Table 2 requirements is a problem that is not unique to SPHEREEx. Modern experimental laboratories everywhere are faced with the similar challenge of developing such a system. Despite this, there are limited options when it comes to standard, well tested data acquisition and instrument control packages providing these capabilities. Familiar to many is NI’s LabView.\(^3\) LabView has the advantage of being well tested, widely adopted, and with the ample support expected from any large-scale proprietary software system. However, integrating custom hardware with LabView can be cumbersome and the graphical programming environment lacks the flexibility that a true general purpose programming language provides. In the open source landscape, the PyMeasure project\(^4\) provides a Python based measurement automation package. PyMeasure contains a large repository of existing instrument drivers along with a convenient framework for adding new instruments. Measurement automation tasks are easily achieved and measurement control logic is specified through basic Python scripts. In addition, PyMeasure has gathered a very active development community to support further improvements to the package. However, .csv is currently the only output file format supported and the package does not have a convenient mechanism for manual instrument

\(^3\)Send correspondence to S. Condon: scondon@caltech.edu
control. PyHK is an additional Python based open source data acquisition package with adoption in astrophysics and astronomy instrumentation laboratories such as the BICEP and TIME collaborations. Initially developed at Caltech for control and monitoring of cryogenic instruments, PyHK provides an excellent web-based user interface for manually controlling instruments, as well as viewing and interacting with housekeeping data stored over long periods of time. PyHK also contains a robust configuration file based interface for specifying the instruments in use and to customize a web interface for a given experiment. PyHK’s limitations come from the fact that the instrument driver framework is not as robust as PyMeasure's and plain text is the only supported output file format.

Following the evaluation outlined above, it was determined that a custom data acquisition and instrument control system would best suit the needs of SPHEREx characterization/calibration. SPHERExLabTools, hereafter referred to as SLT, is the resulting system. SLT meets all of the requirements and presents a framework for a general purpose laboratory data acquisition and instrument control system. SLT adopts the same instrument driver framework and method of specifying automated measurements as PyMeasure while providing a configuration file based interface that is similar to PyHK. In addition, SLT addresses the limitations of PyMeasure and PyHK by providing support for multiple output file formats, while integrating both manual and automated measurement control mechanisms.

The structure of the remainder of this paper is as follows. Section 2 describes the architecture and configuration mechanism of SLT, sections 3 and 4 describe the SPHEREx spectral and focus calibration measurements and illustrate how SLT is configured for each measurement. Initial calibration results obtained using SLT are also presented in these sections. Finally, section 5 provides concluding remarks and discusses plans for further development of the SLT system.

2. SLT ARCHITECTURE

2.1 Implementation Language and Class Structure

SLT is implemented in pure Python. This decision was motivated by the recent wide adoption of Python in many scientific communities and the existence of the powerful Python data ecosystem including packages like NumPy, SciPy, and Pandas. The implementation of SLT in Python allows us to utilize this ecosystem while working in an environment familiar to many scientists involved in SPHEREx and beyond. In addition, Python’s object-oriented nature facilitates the development of organized modular code for which each requirement of Table 2 can be met through independent configurable modules and/or classes. The SLT class based modular architecture is summarized in Figure 1 while section 2.3 describes in more detail all of the classes outlined in this figure.

2.2 Experiment Control Packages and Configurability

SLT focuses heavily on configurability to support the diversity of measurements and instruments involved in SPHEREx testing. The manner in which SLT is configured is through the specification of Experiment Control Packages. Experiment control packages are Python packages that define the 5 configuration variables described in Table 3. In brief, these configuration variables are read by the SLT Experiment class (6) of Figure 1) to command SLT which instrument drivers and graphical interfaces to instantiate, what the output data formats should be, and where to find user-defined measurement automation scripts (i.e. Procedures, sect. 2.3.1). Table 3 describes the purpose of each of these configuration variables, while Tables 4 through 8 specify each variable’s appropriate contents.
Figure 1: The SLT modular architecture consists of 6 classes which interact in the following manner: 1) Procedures contain scripts which define the logical flow of a measurement; they interact directly with 2) Instruments and generate data which they send to 3) Viewers for real-time graphical display and/or 4) Recorders to archive data. Recorders and Viewers sit idle until data is placed on their associated queue, upon which they perform the appropriate archival or display task. 5) Controllers provide graphical interfaces to interact directly with Instruments or to start/stop Procedure execution. Since both Controllers and Procedures have the ability to interact directly with Instruments, a locking mechanism is placed on Instruments such that only one controller/procedure thread of execution can access any given instrument driver class at a single instant. Finally, the 6) Experiment class provides a top-level wrapper to interpret user configuration files and create instances of all of the previous classes. SLT is a multi-threaded application so many instances of each of these components can execute simultaneously. The color-scheme in this figure indicates the threading structure where each color corresponds to a separate thread of execution.

Table 3: SLT Experiment control package configuration variables. Each variable is a python list of configuration dictionaries (see tables 4 - 8). The numbered elements of Figure 1 correspond to the variable numbers in this table.

| Variable      | Description                                                                 |
|---------------|-----------------------------------------------------------------------------|
| 1) PROCEDURES | List of procedure configuration dictionaries (Table 4) pointing SLT to the user defined measurement automation scripts i.e. (Procedure classes). |
| 2) INSTRUMENT_SUITE | List of hardware configuration dictionaries (Table 5) telling SLT which instrument drivers should be instantiated. |
| 3) VIEWERS    | List of viewer configuration dictionaries (Table 6) defining data visualization tasks. |
| 4) RECORDERS  | List of recorder configuration dictionaries (Table 7) to specify the output file formats that data generated from procedures should be saved to. |
| 5) CONTROLLERS | List of controller configuration dictionaries (Table 8) identifying the instruments and procedures for which graphical control interfaces should be generated. |
Table 4: SLT **procedure configuration dictionary** format. The **1) PROCEDURE** variable is a python list of dictionaries with these key/value pairs.

| Key       | Value Description                                                                 |
|-----------|-----------------------------------------------------------------------------------|
| instance_name | String to provide a name for the procedure instance.                               |
| type     | String identifying the name of the procedure class to instantiate.                 |
| hw       | String name of the instantiated instrument drivers that the procedure needs access to. |
| records | Dictionary to specify output data products and their associated viewer/recorder queues. |
| kwargs  | Dictionary with procedure initialization arguments.                                |

Table 5: SLT **hardware configuration dictionary** format. The **2) INSTRUMENT_SUITE** variable is a python list of dictionaries with these key/value pairs.

| Key       | Value Description                                                                 |
|-----------|-----------------------------------------------------------------------------------|
| instance_name | String to provide a name for the instrument driver instance.                      |
| resource_name | String identifying the instrument communication port.                           |
| manufacturer | String name of the manufacturer of the instrument as labeled in the SLT/PyMeasure instrument repository |
| instrument  | String name of the instrument driver class.                                       |
| params     | Dictionary with instrument parameters to set upon initialization.                 |
| kwargs     | Dictionary with arguments for the driver initialization method.                   |

Table 6: SLT **viewer configuration dictionary** format. The **3) VIEWERS** variable is a python list of dictionaries with these key/value pairs.

| Key       | Value Description                                                                 |
|-----------|-----------------------------------------------------------------------------------|
| instance_name | String name of the viewer instance.                                               |
| type     | String type of the viewer. Currently "LineViewer" and "ImageViewer" are supported to display time stream and image data respectively. |
| kwargs  | Dictionary with viewer initialization arguments.                                  |

Table 7: SLT **recorder configuration dictionary** format. The **4) RECORDERS** variable is a python list of dictionaries with these key/value pairs.

| Key       | Value Description                                                                 |
|-----------|-----------------------------------------------------------------------------------|
| instance_name | String name of the recorder instance.                                            |
| type     | String type of the recorder to specify the output file format.                   |
| kwargs  | Dictionary with recorder initialization arguments.                               |
Table 8: SLT controller configuration dictionary format. The 5) CONTROLLERS variable is a python list of dictionaries with these key/value pairs.

| Key      | Value Description |
|----------|-------------------|
| instance_name | String name of the controller instance. |
| type     | String type of the controller. Either "InstrumentController" or "ProcedureController" |
| hw       | String name of the hardware to be controlled if the type is "InstrumentController." |
| procedure | String name of the procedure to be controlled if the type is "ProcedureController." |
| kwargs   | Dictionary with controller initialization arguments. |

2.3 Class Descriptions and User Interface

2.3.1 Procedures

The central element of both the PyMeasure and SLT packages is the Procedure class. As mentioned in Figure 1, Procedures are used to define the logical flow of a measurement. Procedures contain the scripts which interact directly with instruments, collect and process data, and send data to Viewers and Recorders. The basic working principle of a procedure in PyMeasure and SLT is the same with SLT modifying the manner in which procedure output data is handled. Procedures in PyMeasure and SLT contain the following components which are illustrated in the flowchart of Figure 2:

1. Parameter Objects: Classes defining the configurable elements of a Procedure. Parameter objects map directly to the graphical elements found in Procedure Controllers (sect. 2.3.4) so that they can be set and modified via a graphical interface.

2. startup(): The initial method which executes first when a Procedure is started. In general, startup() is used to perform the initial configuration of instrumentation in a testbed before the real measurement begins.

3. execute(): This method always contains the main body of the measurement control code and executes after startup(). This is where data is collected, processed, and sent out for display and archival.

4. shutdown(): The final method executing after execute(). Here, all instrumentation is placed in an idle state before the next measurement is run.

5. emit(): This method is used to send data out for display and archival. In PyMeasure, Python’s built-in logging package is used to write data records sent out by emit() to csv log files. Since the logging package only supports plain-text data, the output file formats supported by PyMeasure are inherently limited. In SLT we modify this mechanism to support multiple output file formats and to separate data visualization from data archival. As illustrated in Figure 1, all Viewers and Recorders have associated queue objects. When emit() is called in SLT, two arguments are provided: The first is a string that identifies which queue objects data should be sent to and the second is the data itself. Copies of the data are then generated and placed onto every desired queue object. When the Viewer and/or Recorder associated with a given queue object detects that new data is present, the data is retrieved from the queue and the appropriate display/archival task is performed. This approach has the advantage that python queue objects have no type preference so any data type can be input to the queue. Accordingly, data of all shapes and sizes can be supported so long as the appropriate Viewer/Recorder class to handle the data has been implemented. In addition, queues in SLT are independent between each instance of a Viewer or Recorder. This allows information to be displayed in a Viewer but not necessarily saved out with a Recorder and vice versa. Or, multiple different Viewers and Recorders can be configured to process the same output data. An argument against this approach is that each time emit() is called, copies of the data object must be created for every
queue so for large data objects, CPU memory could become a limiting factor. However, with modern CPU memory sizes this limitation effectively does not exist for the vast majority of practical SLT applications (certainly all applications in SPHEREx testing).

Figure 2: SLT Procedure Execution and Structure: Procedures in SLT and PyMeasure operate with the same basic principle, but with different mechanisms for how output data is handled. Shown here is a basic example procedure for the collimator calibration measurement (sect. 4). In this procedure, the startup() method moves the microscope into position and opens or closes the shutter. In execute() images are recorded from the ccd camera and the emit() method is used to send images out to the "camera_stream" recorder and viewer queues. A Procedure Controller is generated to allow measurement operators to configure the microscope position, shutter state, and exposure time.

2.3.2 Instruments

SLT adopts the same instrument driver framework as PyMeasure. This allows seamless integration of the extensive repository of existing PyMeasure drivers into SLT. Additionally, users of SLT can develop drivers for instruments not already supported by SLT or PyMeasure by following the detailed instructions in the Adding Instruments section of the PyMeasure documentation.

2.3.3 Viewers and Recorders

All viewers and recorders sit idle until data is placed on their associated queues, upon which they retrieve the data object and perform their archival or display task. To date, Viewer classes have been developed for multi-channel timestream and 2-dimensional image data display. Recorder classes to support data output to CSV, HDF5, and MAT file formats as well as Structured Query Language (SQL) databases exist.
2.3.4 Controllers

SLT implements two main types of controllers: **Instrument Controllers** and **Procedure Controllers**. Instrument Controllers couple directly to instrument drivers and allow manual control of any instrument supported by SLT and PyMeasure through graphical interfaces. Procedure Controllers are used to execute individual procedures as well as several procedures in a sequence. Users can set and modify the **Parameter Objects** for a given procedure and create **nested for loops** where the parameters of the procedure are modified at each step of the loop.

2.3.5 User Interface

Shown in Figure 3 is the SLT user interface generated with the example *Experiment Control Package* described in section 4. Note that all graphical elements of SLT are implemented using the PyQtGraph graphics and gui library.¹¹

---

3. EXAMPLE: SLT FOR SPHEREX SPECTRAL CALIBRATION

The SPHEREx spectral calibration measurement aims to deliver per-pixel spectral bandpass response profiles for every pixel in the SPHEREx detectors. This measurement is important as precise knowledge of the SPHEREx spectral response is required to accurately measure galaxy redshifts and ice absorption features in the Milky Way. We accomplish SPHEREx spectral calibration by coupling a broadband light source to a monochromator whose output beam is collimated and input to the SPHEREx optics. The monochromator contains diffraction gratings whose diffraction pattern is reflected through a small slit such that a monochromatic light source of a desired
wavelength is generated. The collimated, monochromatic light is then scanned across the entire SPHEREx band in less than 1 nm wavelength steps. SPHEREx detector exposures are recorded at each wavelength. Figure 4 provides a block diagram of the measurement setup and Figure 5 shows the testbed for the spectral calibration measurement of a prototype focal plane assembly model. Table 9 describes the SLT experiment control package configuration variables for the measurement while bandpass results for the prototype calibration are shown for a single pixel in Figure 6.

Figure 4: SPHEREx Spectral Calibration Measurement Schematic: Here, SLT drives the monochromator, a set of shutters including a cold shutter inside of the cryostat, and a neutral density filter attenuation wheel. SLT also communicates directly with the detector data acquisition system while logging the cryostat input optical power level and pressure/temperature housekeeping data. SLT uploads all metadata, housekeeping, and detector data to a central SQL server.

Figure 5: Prototype focal plane array spectral calibration testbed
Table 9: SLT experiment control package for spectral calibration

| Variable        | Description                                                                                                                                 |
|-----------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| 1) PROCEDURES   | **PowerLogProc**: Continuously logs the input optical power level. **SpecCalProc**: Sets the monochromator wavelength, attenuation wheel position, and communicates with the detector data acquisition system. Calculates the average cryostat temperature and pressure. Sends all instrument metadata, cryo housekeeping, and detector exposures to the SpecCalSql recorder. |
| 2) INSTRUMENT_SUITE | Contains hardware configuration dictionaries for the monochromator, attenuation wheel, power monitor detector readout, cryostat temperature and pressure readout, and the detector data acquisition system. |
| 3) VIEWERS      | **PowerView**: Displays the time-stream input optical power level data generated by the PowerLogProc procedure.                                 |
| 4) RECORDERS    | **SpecCalSql**: Writes data from SpecCalProc to a SQL database.                                                                                |
| 5) CONTROLLERS  | Configuration dictionaries to generate manual instrument controllers for all instruments in INSTRUMENT_SUITE and procedure controllers for all procedures in PROCEDURES. |

Figure 6: Spectral calibration bandpass results for a single pixel of the prototype focal plane assembly. The spectral information of this sample pixel is obtained by scanning the monochromator from 0.7 to 1.3 \( \mu \text{m} \) at 2 nm wavelength steps. This 10 hour measurement procedure is performed in a fully automated manner by SLT, running the SpecCalProc procedure described in Table 9.

4. EXAMPLE: SLT FOR SPHEREX FOCUS CALIBRATION

The SPHEREx focus calibration quantifies telescope focus error by determining the axial displacement of the focal plane assembly hardware from the true focal plane of the telescope. Focus calibration is important to verify
that the SPHEREx PSF is such that a sufficient fraction of background sky pixels and source masking can be achieved to produce a spatial power spectrum measurement of the extragalactic background light. We perform this calibration through two measurements: collimator calibration and focus error. The collimator calibration allows us to position a broadband pinhole light-source relative to an off-axis-paraboloid (OAP) such that we generate a collimated beam. We then steer the collimated beam into the SPHEREx optics and move the pinhole through a set of positions so that the beam deviates from collimation. We record SPHEREx exposures at each pinhole position and find the position at which the SPHEREx PSF is minimized. The difference between the initial pinhole position (at which the beam is collimated) and the final pinhole position (at which the SPHEREx PSF is a minimum) determines SPHEREx focus error. We perform the collimator calibration, i.e. positioning of a pinhole to generate a collimated beam, through the following procedure:

1. An LED is fiber coupled to a microscope mounted on a motorized stage.
2. A flat is placed inside the cryo-chamber in front of the SPHEREx optics to reflect incoming optical signal.
3. The LED source propagates through the collimating optics whose output beam is steered into the flat and reflected back into the collimating optics.
4. The reflected beam is re-imaged onto the surface of a CCD camera in the microscope and the spot size on the camera is calculated.
5. The microscope is scanned through several positions at which a camera spot size is calculated such that a microscope position vs. spot size curve is generated.
6. The minimum of a fit to the spot size vs. microscope position curve is taken to be the "best focus position" of the collimator. This is the position at which the pinhole source for the focus error measurement should be placed to generate a collimated beam.

Figure 7: **Collimator calibration measurement schematic.** Here, SLT drives the motorized microscope position and reads out this position. SLT drives a motorized relay to steer the beam into the cryo chamber, and implements a module for CCD camera readout and basic real-time processing. Cryo housekeeping data is logged and written to a SQL data server along with all position information and CCD camera images. Collimator focus curves can be generated on the fly to facilitate a rapid transition to the focus error measurement.
The *SLT experiment control package* utilized for the collimator calibration measurement is described in Table 10 and an example collimator calibration focus curve generated by SLT is shown in Figure 8. The collimating optics / microscope and motorized relay setup are pictured in Figures 9 and 10 respectively.

Table 10: SLT experiment control package for collimator calibration

| Variable               | Description                                                                                                                                                                                                 |
|------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) PROCEDURES          |                                                                                                                                                                                                             |
| CamViewProc            | Continuously records frames from the CCD to send out to the CamView image viewer.                                                                                                                                                      |
| CollimatorFocusProc    | Drives the microscope and relay motors. Reads positions and CCD camera images. Records average cryo temp/pressure. Sends images to CamView and images + position information to the CollimatorFocusRecorder.                                           |
| 2) INSTRUMENT_SUITE    | Contains hardware configuration dictionaries for the CCD camera, the microscope drive motor and microscope position readout, the motorized relay drive motors, cryostat temperature and pressure readout, and the detector data acquisition system. |
| 3) VIEWERS             | CamView: View live camera images recorded from CamViewProc or CollimatorFocusProc                                                                                                                                               |
| 4) RECORDERS           | CollimatorFocusRecorder: Write images + position information from CollimatorFocusProc to an HDF5 file.                                                                                                                               |
| 5) CONTROLLERS         | Configuration dictionaries to generate manual instrument controllers for all instruments in INSTRUMENT_SUITE and procedure controllers for all procedures in PROCEDURES.                                      |

Figure 8: Collimator calibration focus curve: Black dots represent calculated beam sizes, the blue dash is the quadratic fit to the beam size data, and the red X is the minimum of the fit which is taken to be the "best focus position" of the collimator.
5. CONCLUSION

We have presented a high-level overview of the SPHERExLabTools (SLT) data acquisition and instrument control package utilized in SPHEREx optics integration and testing. SLT builds upon and adapts the existing open source packages, PyMeasure\(^4\) and PyHK\(^5\) and presents a framework for a general purpose laboratory data acquisition and instrument control package. The effectiveness of SLT has been demonstrated through the delivery of early SPHEREx calibration data products, including the focal plane prototype spectral response\(^3\) and collimator calibration\(^4\) measurements. SLT development is ongoing and the package continues to improve. An open source release along with work to integrate key features of SLT into the PyMeasure project are planned, with the long term vision of helping to establish an open-source standard in laboratory data-acquisition and instrument control.

ACKNOWLEDGMENTS

This work was carried out at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration (80NM0018D0004).
REFERENCES

[1] Korngut, P. M., Bock, J. J., Akeson, R., Ashby, M., Bleem, L., Boland, J., Bolton, D., Bradford, S., Braun, D., Bryan, S., Capak, P., Chang, T.-C., Coffey, A., Cooray, A., Crill, B., Doré, O., Eifler, T., Feng, C., Habib, S., Heitmann, K., Henmatti, S., Hirata, C., Jeong, W.-S., Kim, M., Kirkpatrick, D., Kowalkowski, T., Krause, E., Lisse, C., Mauskopf, P., Masters, D., McGuire, J., Mehnick, G., Nguyen, H., Nayyeri, H., Oberg, K., dePutter, R., Purcell, W., Rocca, J., Runyan, M., Sandstrom, K., Smith, R., Song, Y.-S., Stickley, N., Stober, J., Susca, S., Teplitz, H., Tolls, V., Unwin, S., Werner, M., Windhorst, R., and Zemcov, M., “SPHEREx: an all-sky NIR spectral survey,” in [Space Telescopes and Instrumentation 2018: Optical, Infrared, and Millimeter Wave], Lystrup, M., MacEwen, H. A., Fazio, G. G., Batalha, N., Siegler, N., and Tong, E. C., eds., 10698, 576 – 586, International Society for Optics and Photonics, SPIE (2018).

[2] Crill, B. P., Werner, M., Akeson, R., Ashby, M., Bleem, L., Bock, J. J., Bryan, S., Burnham, J., Byunh, J., Chang, T.-C., Chiang, Y.-K., Cook, W., Cooray, A., Davis, A., Doré, O., Dowell, C. D., Dubois-Felsmann, G., Eifler, T., Faist, A., Habib, S., Heinrich, C., Heitmann, K., Heaton, G., Hirata, C., Hristov, V., Hui, H., Jeong, W.-S., Kang, J. H., Kecman, B., Kirkpatrick, J. D., Korngut, P. M., Krause, E., Lee, B., Lisse, C., Masters, D., Mauskopf, P., Mehnick, G., Miyasaka, H., Nayyeri, H., Nguyen, H., Öberg, K., Padin, S., Paladini, R., Pourrahmani, M., Pyo, J., Smith, R., Song, Y.-S., Symons, T., Teplitz, H., Tolls, V., Unwin, S., Windhorst, R., Yang, Y., and Zemcov, M., “SPHEREx: NASA’s near-infrared spectrophotometric all-sky survey,” in [Space Telescopes and Instrumentation 2020: Optical, Infrared, and Millimeter Wave], Lystrup, M., Perrin, M. D., Batalha, N., Siegler, N., and Tong, E. C., eds., 11443, 61 – 77, International Society for Optics and Photonics, SPIE (2020).

[3] Bitter, R., Mohiuddin, T., and Nawrocki, M., [LabVIEW: Advanced programming techniques], Crc Press (2006).

[4] Developers, T. P., “Pymeasure scientific package.”

[5] Hunacek, J., “Pylk v3.0.”

[6] Ade, P. A. R., Ahmed, Z., Amirri, M., Barkats, D., Thakur, R. B., Bischoff, C. A., Beck, D., Bock, J. J., Boenish, H., Bullock, E., Buza, V., Cheshire, J. R., I., Connors, J., Cornelison, J., Crumrine, M., Cukierman, A., Denison, E. V., Dierickx, M., Duband, L., Eiben, M., Fatigoni, S., Filiponi, J. P., Fliescher, S., Goeccker-Wald, N., Goldfinger, D. C., Grayson, J., Grimes, P., Hall, G., Halal, G., Halpern, M., Hand, E., Harrison, S., Henderson, S., Hildebrandt, S. R., Hilton, G. C., Humbmayr, J., Hui, H., Irwin, K. D., Kang, J., Karkare, K. S., Karpel, E., Kefeli, S., Kernasovskiy, S. A., Kovac, J. M., Kuo, C. L., Lau, K., Leitch, E. M., Lennox, A., Megerian, K. G., Minutolo, L., Moncelsi, L., Nakato, Y., Namikawa, T., Nguyen, H. T., O’Brien, R., Ogburn, R. W., I., Palladino, S., Prouve, T., Pryke, C., Racine, B., Reintsema, C. D., Richter, S., Schillaci, A., Schwarz, R., Schmitt, B. L., Sheehy, C. D., Soliman, A., Germaine, T. S., Steinbach, B., Sudiwala, R. V., Teply, G. P., Thompson, K. L., Tolon, J. E., Tucker, C., Turner, A. D., Umiltà, C., Vergès, C., Vieregg, A. G., Wandui, A., Weber, A. C., Wiebe, D. V., Willmert, J., Wong, C. L., Wu, W. L. K., Yang, H., Yoon, K. W., Young, E., Yu, C., Zeng, L., Zhang, C., and Zhang, S., “BICEP/Keck XV: The BICEP3 Cosmic Microwave Background Polarimeter and the First Three-year Data Set,” 927, 77 (Mar. 2022).

[7] Crites, A. T., Bock, J. J., Bradford, C. M., Chang, T. C., Cooray, A. R., Duband, L., Gong, Y., Halley-Dunsheath, S., Hunacek, J., Koch, P. M., Li, C. T., O’Brien, R. C., Prouve, T., Shirokoff, E., Silva, M. B., Staniszewski, Z., Uzgil, B., and Zemcov, M., “The TIME-Pilot intensity mapping experiment,” in [Space Telescopes and Instrumentation for Astronomy VII], Holland, W. S. and Zmuidzinas, J., eds., Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series 9153, 91531W (Aug. 2014).
Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors, “SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python,” Nature Methods 17, 261–272 (2020).

[10] Wes McKinney, “Data Structures for Statistical Computing in Python,” in Proceedings of the 9th Python in Science Conference, Stéfan van der Walt and Jarrod Millman, eds., 56 – 61 (2010).

[11] Developers, T. P., “Pyqtgraph: Scientific graphics and gui library for python.”

[12] Hui, H., Bock, J., Condon, S., Dowell, C., Hollister, M., Korngut, P., Nguyen, C., Nguyen, H., Padin, S., and Viero, M., “Spectral calibration of spherex focal plane,” in Astronomical Telescopes and Instrumentation, Proc. SPIE (2022).