SoftwareX 6 (2017) 85–90

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Qudi: A modular python suite for experiment control and data processing

Jan M. Binder [a], Alexander Stark [a,b], Nikolas Tomek [a], Jochen Scheuer [a], Florian Frank [a], Kay D. Jahnke [a], Christoph Müller [a], Simon Schmitt [a], Mathias H. Metsch [a], Thomas Unden [a], Tobias Gehring [b], Alexander Huck [b], Ulrik L. Andersen [b], Lachlan J. Rogers [a,*], Fedor Jelezko [a,c]

[a] Institute for Quantum Optics, Ulm University, Albert-Einstein-Allee 11, Ulm 89081, Germany
[b] Department of Physics, Technical University of Denmark, Fysikvej, Kongens Lyngby 2800, Denmark
[c] Center for Integrated Quantum Science and Technology (IQst), Ulm University, 89081, Germany

ARTICLE INFO

Article history:
Received 25 November 2016
Received in revised form
30 January 2017
Accepted 2 February 2017

Keywords:
Python 3
Qt
Experiment control
Automation
Measurement software
Framework
Modular

ABSTRACT

Qudi is a general, modular, multi-operating system suite written in Python 3 for controlling laboratory experiments. It provides a structured environment by separating functionality into hardware abstraction, experiment logic and user interface layers. The core feature set comprises a graphical user interface, live data visualization, distributed execution over networks, rapid prototyping via Jupyter notebooks, configuration management, and data recording. Currently, the included modules are focused on confocal microscopy, quantum optics and quantum information experiments, but an expansion into other fields is possible and encouraged.

Code metadata

Current code version	0.6
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-16-00092
Legal Code License	GNU General Public License v3
Code versioning system used	git
Software code languages, tools, and services used	Python3
Compilation requirements, operating environments & dependencies	Environment: Anaconda, Python 3.4+, Python packages: comtypes (Windows only), cycler, fysom, gitpython, influxdb, IPython, jedi, jupyter-client, lmfit, lxml, manhole, matplotlib, numpy, PyDAQmx, pycallgraph, pyqtgraph, PyQt4, qtconsole, qtpy, RPi.GPIO (Raspberry Pi only), rpyc, ruamel.yaml, scipy, spidev (Linux only), statsmodels, traitlets, visa, pywin32 (Windows only), zmq
If available Link to developer documentation/manual	https://ulm-iqo.github.io/qudi-generated-docs/html-docs/
Support email for questions	qudi@uni-ulm.de

* Corresponding author.
E-mail address: lachlan.j.rogers@quantum.diamonds (L.J. Rogers).

http://dx.doi.org/10.1016/j.softx.2017.02.001

Software metadata

Current software version	0.6
Permanent link to executables of this version	https://github.com/Ulm-IQO/qudi/releases/tag/v0.6
Legal Software License	GNU General Public License v3
Computing platforms/Operating Systems	Linux, OS X, Microsoft Windows
Installation requirements & dependencies	Environment: Anaconda, Python 3.4+, Python packages: comtypes (Windows only), cycler, fysom, gitpython, influxdb, IPython, jedi, jupyter-client, lmfit, lxml, manhole, matplotlib, numpy, PyDAQmx, pycallgraph, pyqtgraph, PyQt4, qtconsole, qtpy, RPi.GPIO (Raspberry Pi only), rpyc, ruamel.yaml, scipy, spidev (Linux only), statsmodels, traitlets, visa, pywin32 (Windows only), zmq
If available, link to user manual — if formally published include a reference to the publication in the reference list	https://ulm-iqo.github.io/qudi-generated-docs/html-docs/
Support email for questions	qudi@uni-ulm.de

1. Motivation and significance

Modern scientific experiments typically rely on multiple hardware devices working together in a coordinated fashion. In many instances, the hardware devices are commercial products with programming interfaces for direct control via custom software. The unique combination of such devices is then specific to a given experiment. Efficient control of such experiments requires software that is capable of coordinating the operation of multiple devices. In addition, data interpretation is facilitated by rapid data processing and visualization.

These challenges are exemplified when studying color centers in diamond as solid state quantum emitters for sensing, spin manipulation and quantum information technologies. It is typical for such experiments to be performed on a "home-built confocal microscope" [1–5]. As evidenced by the 2014 Nobel Prize in Chemistry, these techniques have expanded beyond the context of physics and now this kind of microscope is pushing advances in biology [6–8] and nanotechnology [9,10]. A wide range of hardware is used for such experiments, but there is a paucity of mature and flexible lab control software to operate the apparatus.

Here, we present Qudi, a Python software suite for controlling complex experiments and managing the acquisition and processing of measurement data. Despite being developed in the context of quantum optics laboratories, the core Qudi framework is broadly applicable to many scenarios involving coordinated operation of multiple experiment devices. The free and open-source nature of Qudi makes it possible for anyone to use and modify the software to fit their research needs, and the modular code design simplifies this task. Qudi continues to be actively developed, but it is already mature enough for reliable laboratory use [11].

2. Software description

2.1. Why Python?

Python was chosen as the programming language for Qudi because of its conceptual synergy with the goals of the project. As a dynamic, strongly typed, scripting language, Python has become a popular choice for scientific programming [12,13] as the importance of scientific software increases [14]. Python's high level of abstraction makes it human-readable and concise, providing a direct advantage for laboratory programming typically performed by scientists rather than dedicated software developers. Source code availability under an open-source license, the built-in modular structure of Python and good community support lower the initial hurdle to learn the language. Additionally, most laboratory hardware has at least an application programming interface (API) specified for the C programming language, which can be accessed by Python.

Scripting languages cannot replace established compiled programming languages for tasks where processing performance or memory efficiency is required but they are very useful to glue together different components in order to benefit from the advantages each of them can offer [15]. This is closely aligned with the concept of Qudi "gluing" together various devices and control methods for specific complex experiments.

2.2. Qudi design

The Qudi suite consists of a collection of modules that are loaded and connected together by a manager component according to settings given in a configuration file as shown in Fig. 1(a). The program startup code and manager were initially derived from similar elements contained within the neurophysiology software ACQ4 [16]. Startup is initiated by a single executable python file, and the manager component provides core functions for logging, error handling, configuration reading, and remote access. Additionally, the manager also administers the other modules by providing functionality for module loading, module dependency resolution and connection, concurrent execution and network access to modules running on other computers. This core infrastructure makes it easier to rapidly develop modules for new experiments by providing structure and starting points.

A typical Qudi session will proceed as follows. On startup, the supervisor process, for example an IDE, creates a Qudi process. In this Qudi process, the manager component reads the configuration file, sets up the log file and loads the modules designated in the startup section of the configuration file. Typically, the startup section will – but does not have to – contain at least the Manager GUI and the tray icon module. Laboratory operation and experiment control are performed by science modules, which are specified in the configuration file along with any hardware-specific parameters. Science modules can be loaded for the desired measurement from the Manager GUI or a Jupyter notebook. Some of the science modules in Qudi were inspired by the pi3diamond software [3–5,17–19].

The science modules are divided into three categories: hardware interaction, experiment "logic", and user interface. These categories and the relationships between them are illustrated in Fig. 1(b). The division into hardware, logic, and interface represents a clear separation of tasks that improves reliability and flexibility of the Qudi code. It also simplifies the implementation of new experiment modules. The fundamental three-fold distinction is at the basis of Qudi's adaptability, and makes Qudi an experiment control software in contrast to a general software framework.

2.2.1. Logic modules

Logic modules control and synchronize a given experiment. They pass input parameters from the user interface to the respective hardware modules, and process measurement data in the desired way. These modules control the information exchange