Interactive ROS Tutorials with Jupyter Notebooks

Enric Cervera*

Robotic Intelligence Lab
Universitat Jaume-I de Castelló, Spain,
ecervera@uji.es

Abstract. Collaboration is a major strength of ROS, as it was designed specifically for groups with different expertise (mapping, navigation, vision, etc) to collaborate and build upon each other's work. The ROS Wiki is such an example,[1] with the official documentation for ROS packages, as provided by their developers, and every user has dived into its ROS tutorials for learning the basic concepts. Those tutorials consist of web pages with embedded examples of source code that the user can run in a separate terminal. We aim to transform the static tutorials into interactive documents where the user not only reads but also runs the code inside the same web browser, by using Jupyter Notebook technology.[2] Our goal is to make the tutorials more concise and effective, avoiding the tedious and error-prone copying-and-pasting of code. Also, the integration of execution results provides the user with step-by-step feedback, for a more illuminating learning experience.

Keywords: tutorial, interaction, literate computing

1 Introduction

The ROS Tutorials are the primary source of information for learning ROS. They consist of a collection of web pages about the installation and setup of the system, the core concepts, and the development of robot applications in the programming languages C++ or Python.

Each web page presents a specific issue, combining an informative description interleaved with snippets of code or commands. The code must be first copied and pasted into an editor, then run in a terminal. The commands can be copy-pasted directly into the terminal, and executed. Such a workflow requires several views simultaneously:

- Documentation (teaching materials)
- Edition of the source code

* ORCID id: 0000-0002-5386-8968. The views and opinions expressed in this article are those of the author, not affiliated with any of the projects mentioned.

[1] http://wiki.ros.org/

[2] http://jupyter.org/

– Execution and monitoring of results

For a proper management of these views, several windows are needed in the user desktop. A typical layout is shown in Fig. 1, with the browser window at the left, displaying the documentation, the editor window at the top right, and the terminal window for execution at the bottom right.

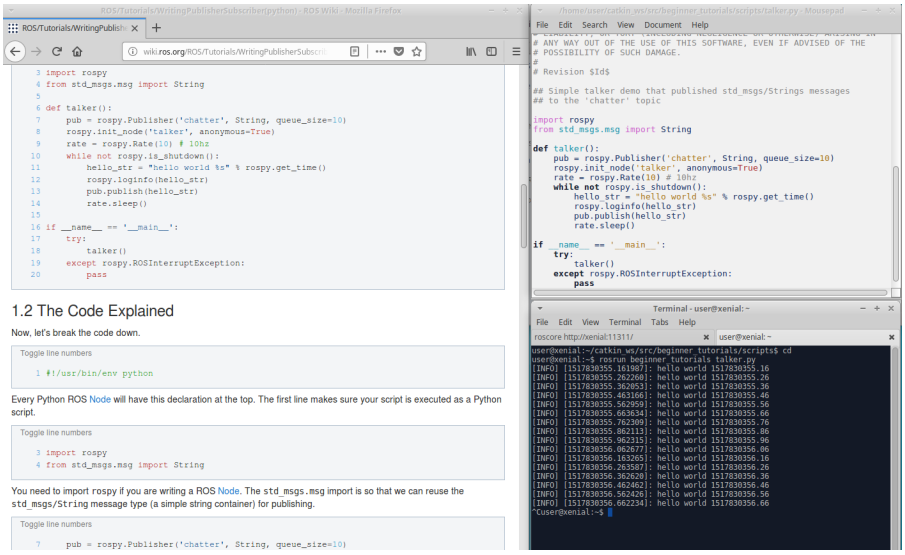

Fig. 1. Desktop layout: left window is the browser with the documentation web page; top-right is the editor window with the source code; bottom-right is the terminal window for the execution of the program.

There are some disadvantages in this arrangement, the most obvious one is the duplication of code: the programming statements are shown in the documentation, then they need to be copied into the development editor. In fact, some tutorials feature two versions of the same code, a complete one for easy copying and pasting, and a step-by-step version for the explanations.

In addition, the execution of the code takes place in a third window (the console), losing the context of the source code. Moreover, the complete program is run, making it difficult to execute the statements step-by-step, as opposed to the explanations in the tutorial.

Jupyter notebooks provide an integrated environment for the execution of code statements, interleaved in the same web pages of documentation. The code can be executed in cells, or snippets of code with a few statements, and the results are displayed below the same cell of code. Code cells can be edited and run freely by the user, interactively. As a result, a single window combines the three necessary views (teaching materials, edition, execution), as shown in Fig. 2.

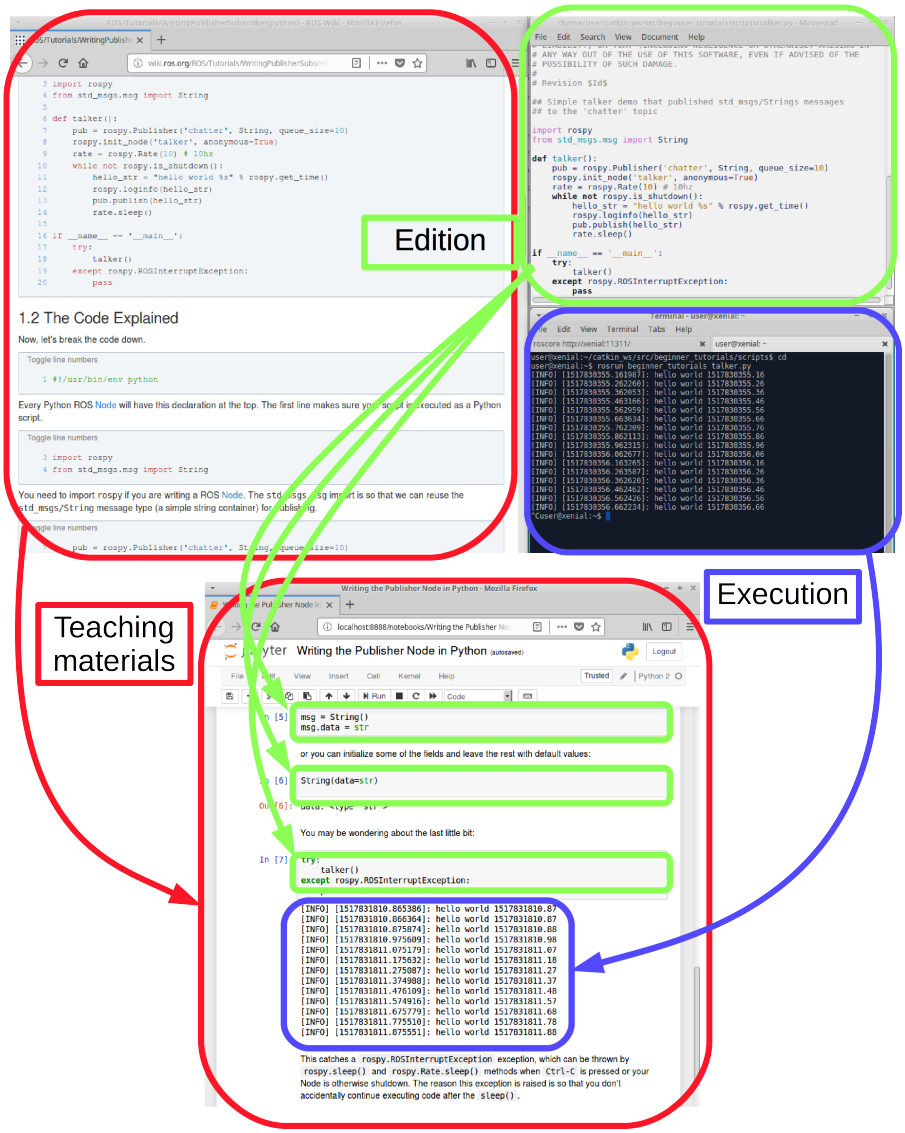

Fig. 2. Interactive ROS Tutorial on a Jupyter notebook: the teaching materials, source code, and execution results are all presented together in the browser window.

This paper describes how jupyter notebooks work (Section 2), how ROS and its tutorials can be integrated in the workflow (Section 3), and a couple of examples of representative tutorials regarding ROS topics and services (Section 4). Finally, Section 5 draws some conclusions and outlines the work in progress.

2 Jupyter Notebooks

The Jupyter Notebook is an interactive computing environment that extends the console-based approach in a new direction: it provides a web-based application suitable for capturing the whole computation process of developing, documenting, and executing code. Its roots lie in the Interactive Python environment [5], and the ideas behind the proprietary-based Mathematica Notebook environment [12].

Jupyter Notebooks are experiencing an immense success in recent years, applied to education and research [7] in a number of disciplines like computer vision, machine learning, or even the analysis of gravitational wave data [9].

Jupyter implements a "literate computing" scheme [4], inspired in the "literate programming" paradigm [3], a software development style pioneered by Stanford computer scientist, Donald Knuth, that emphasizes an approach where exposition with human-friendly text is punctuated with code blocks. Such an approach seems more adequate for demonstration, teaching and research purposes, especially for science and technology.

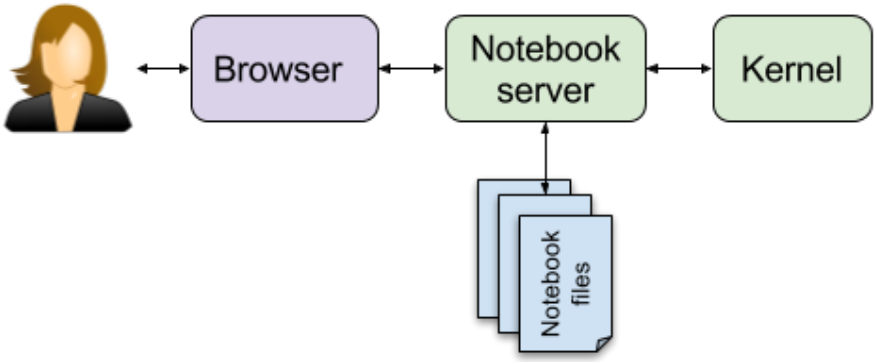

Fig. 3. Jupyter notebook elements.

As a server-client application, there are several processes involved in the user interaction, as seen in Fig. 3. The kernel is the separate process that actually runs the code. The notebook server stores code and output, together with markdown notes (a lightweight markup language that can be converted to HTML), in an editable document called a notebook. The user interacts with the browser, which

is connected to the notebook server for rendering notebooks, and sending user input to the kernel.

There are kernels available for many languages, starting from Julia, Python and R (the original languages Jupyter was developed for —hence the acronym) but also to other compiled languages like Java or C++.

3 Integration of Jupyter and ROS

Jupyter notebooks are being used in robotic Internet sites such The Construct [8], which also integrates 3D simulations with Gazebo. In our approach we will only focus on the programming task.

ROS development is mainly based on the languages C++ and Python. The former is perhaps the most widely used client library, and it is designed for high performance. The latter provides the advantages of an object-oriented scripting language: it favors implementation speed so that algorithms can be quickly prototyped and tested. Python is a core dependency of ROS, since some tools are developed in that language.

3.1 Local Install

Jupyter can be easily installed using Python's package manager,[3] or third-party distributions like Anaconda.[4] Once installed, it is launched from the terminal, where the server keeps running in the background. At the same time, a new browser window opens and displays the notebook folder, or a specific notebook.

The user can then interact with the notebook, with an environment similar to that of the console, i.e. all the C++ or Python libraries are accessible as in any ROS application. Practical examples of tutorials will be shown in Section 4.

Obviously, ROS is a prerequisite that should be available in the system, either in a local installation or as a Docker image.[5]

3.2 Notebooks in the Cloud

Since Jupyter Notebooks use a client/server approach, it is possible to run the server in a different computer than the client. Both machines surely must be connected, but the connection protocol (http - the WWW protocol) makes it possible to run the server in any computer connected to Internet.

Free cloud services for IPython and Jupyter already exist, where their basic functionality can be tested.[6]

A recent and interesting initiative is Binder,[7] which not only enables sharing of live notebooks in a computational environment, but authors can publish

[3] https://packaging.python.org/tutorials/installing-packages/
[4] https://www.anaconda.com/
[5] http://wiki.ros.org/docker/
[6] https://try.jupyter.org/
[7] https://mybinder.org/

notebooks in a source code repository along with an environment specification. By pointing the Binder web service at the repository, a temporary environment is automatically created with the notebooks and any libraries and data required to run them. This allows authors to publish their code in an interactive and immediately verifiable form [2].

An environment specification for ROS can be created based on the ROS docker images.[8] These images provide a simplified and consistent platform to build and deploy applications, with an easy way to develop, reuse and ship software [11].

We have upgraded the ROS docker images with the necessary packages for running Jupyter along ROS, so that the example tutorials presented in the next section are stored in a GitHub repository that can be run either in a local environment, or in the cloud using the Binder service.[9]

4 Tutorial Examples

For demonstration purposes, we have adapted the tutorials for some of the most fundamental concepts in ROS: topics and services [6]. Topics are named buses over which nodes exchange messages. They have anonymous publish / subscribe semantics: nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic.

The publish / subscribe model is a very flexible communication paradigm, but its many-to-many one-way transport is not appropriate for request / reply interactions. For that purpose, ROS features another paradigm called Service, which is defined by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it were a remote procedure call.

We have developed the Jupyter Notebook versions of the ROS tutorials for writing a simple publisher and subscriber of topics, and a simple service and client. The primary programming language is Python, but a working example in C++ will also be presented.

4.1 Writing a Simple Publisher and Subscriber

This tutorial consists of one section for the publisher node, and another section for the subscriber node.[10] Each section is divided into two sub-sections, a first part with the code, and a second one with the "code explained". This last part consists of the code split into chunks of a few lines, followed by paragraphs of description about each code fragment.

[8] https://hub.docker.com/_/ros/
[9] https://github.com/RobInLabUJI/ROS-Tutorials
[10] http://wiki.ros.org/ROS/Tutorials/WritingServiceClient(python)

We first divide the materials into two notebooks, for the publisher and the subscriber. The first section is not necessary, since the code in the "explanatory" section can be readily executed in the notebook. So this second section is transferred to the notebook by simply moving the source code into "code cells" and the description into the "markdown cells" (Fig. 4).

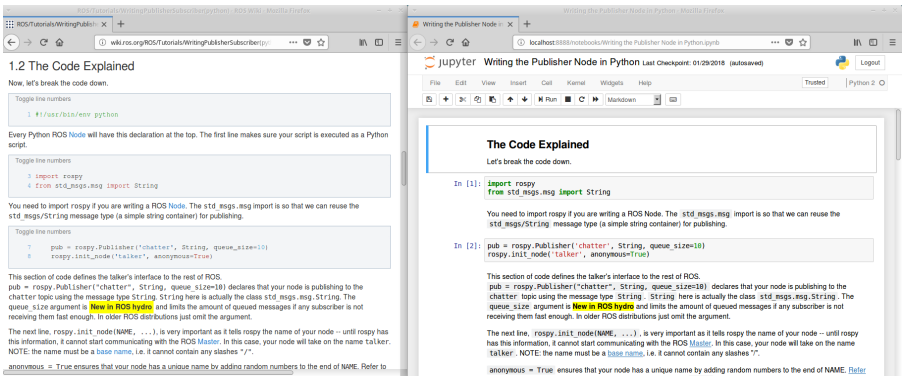

Fig. 4. Adapting the ROS Topics tutorial webpage (left) to a Jupyter Notebook (right). The source code is transferred to code cells, and the explanatory text is copied into markdown cells. The layout remains very similar, and the links are kept.

Some additional simplifications apply: the initial declaration for Python scripts is not necessary, since the Jupyter environment is already running a Python kernel. For the same reason, it is not necessary to add the lines of code for checking that a "main" application is running. One should keep in mind that for the code to be re-useable in another execution environment, the full source code should be used.

The subscriber code is adapted in a similar way. The resulting document has fewer lines than the original, since there is no duplicated code, and some unnecessary statements have been removed. In addition, it can be readily executed from the same browser, without need to launch the script in a console terminal (Fig. 5).

4.2 Writing a Simple Service and Client

The structure of this tutorial is similar to the previous one.[11] Again, the tutorial is split into two notebooks, for the service and client respectively. Each notebook consists of the contents of the section "code explained", either as code or markdown cells.

The code can be readily executed on each notebook in parallel, and the results of the remote call are shown in both notebooks too (Fig. 6).

[11] http://wiki.ros.org/ROS/Tutorials/WritingServiceClient(python)

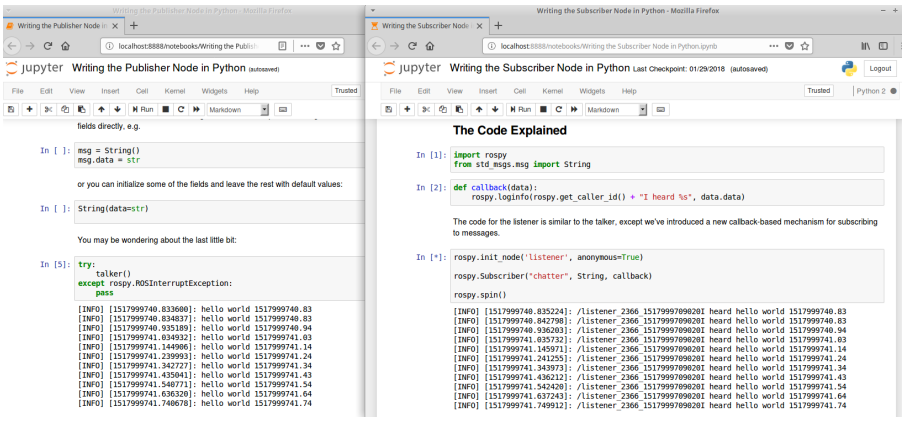

Fig. 5. Published and subscriber code running in Jupyter notebooks. The output of each node is displayed in the corresponding notebook.

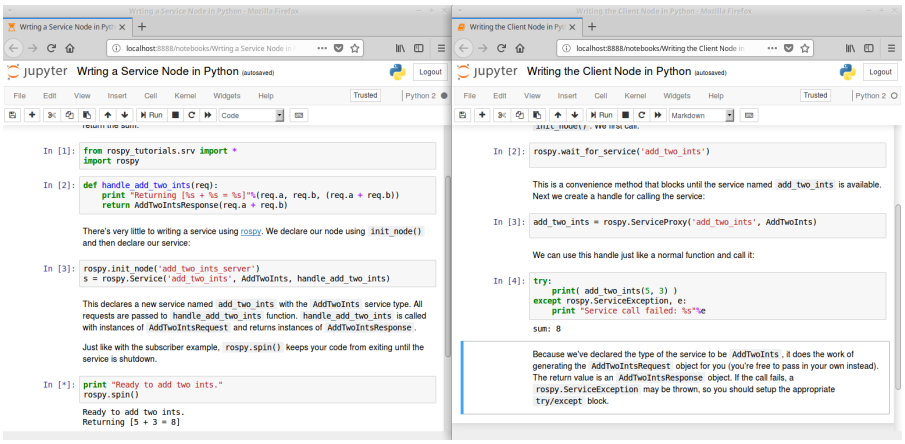

Fig. 6. Service and client code running in Jupyter notebooks. The output of each node is displayed in the corresponding notebook.

4.3 Adapting Tutorials in C++

Though Jupyter has evolved from IPython, much effort has been devoted to the separation between the kernel running the code, and the notebook frontend. Nowadays, more than 40 different kernels are available, each for a huge variety of programming languages.[12]

One of such kernels is based on Cling, an interactive C++ Interpreter [10] developed within ROOT, a framework for data processing created at CERN [1].

We have also adapted the C++ versions of the topics and services tutorials to jupyter notebooks. Similar to Python, slight changes are needed: the main function is removed, since the code is run interactively step-by-step; blocks of code cannot be split across cells, e.g. loops or if-else constructs. Care must be taken with the declarations, since any second run raises an error. The solution is to restart the kernel, which eliminates all declared variables for a fresh start.

The results of a C++ notebook for the tutorial on ROS services is shown in Fig. 7. These results are coming from a local install, since at this moment the Binder version of the tutorial only features the Python kernel.

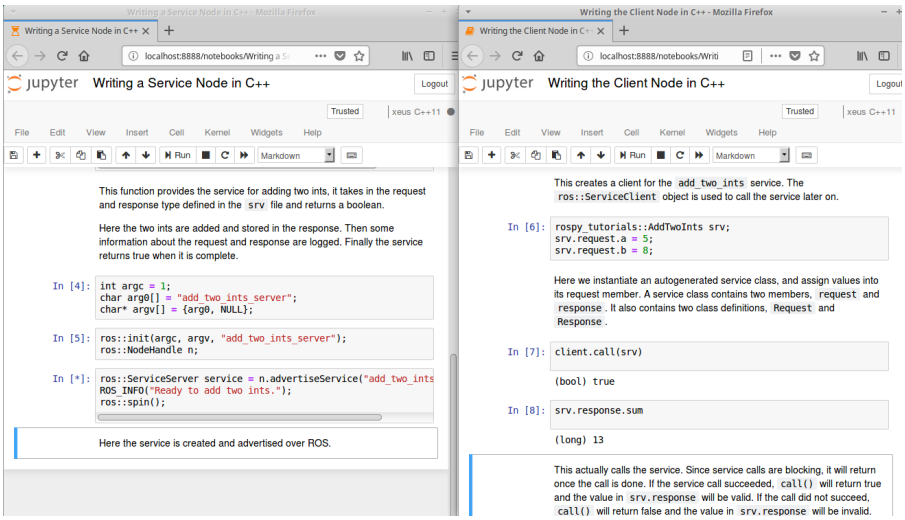

Fig. 7. Service and client C++ code running in Jupyter notebooks.

5 Conclusions and Future Work

Jupyter notebooks bring new live to ROS tutorials; instead of copying-and-pasting code and running it in a terminal console, the code is interleaved in

[12] https://github.com/jupyter/jupyter/wiki/Jupyter-kernels

the same tutorial, where it can be run, and its output is displayed in context. Tutorials become more concise, and the user workflow is greatly simplified, thus enhancing the learning process.

Jupyter is an open-source, cross-platform project that has become enormously popular in the recent years. A variety of programming languages is supported, including C++ and Python, the most widely-used languages in ROS. In future extensions, we plan to test languages like Lisp, Go, Haskell, Java, Node.js, Julia, and Ruby: all of them are available both in Jupyter kernels and ROS client libraries.[13] [14]

The notebook platform is easily installed in a local computer alongside with ROS, or it can be deployed in a cloud server. The Binder web service allows any author to publish a tutorial in a public web repository that becomes live instantly, allowing other users to try ROS without even installing it in their computers.

Though the adaptation of existing tutorials to interactive notebooks can be time-consuming, as it is done manually by now, we encourage the ROS community to use these tools for the development of new materials, since teaching of robotics and ROS can greatly benefit from them.

In the future, we plan to evaluate groups of students for quantitative measurements of the user proficiency in comparison with the classical tutorial approach.

Acknowledgments

Support of IEEE RAS through the CEMRA program (Creation of Educational Material for Robotics and Automation) is gratefully acknowledged. This paper describes research done at the Robotic Intelligence Laboratory. Support for this laboratory is provided in part by Ministerio de Economia y Competitividad (DPI2015-69041-R), by Generalitat Valenciana (PROMETEOII/2014/028) and by Universitat Jaume I (P1-1B2014-52).

References

1. Antcheva, I., Ballintijn, M., Bellenot, B., Biskup, M., Brun, R., Buncic, N., Canal, P., Casadei, D., Couet, O., Fine, V., et al.: Root—a c++ framework for petabyte data storage, statistical analysis and visualization. Computer Physics Communications **182**(6), 1384–1385 (2011)
2. Kluyver, T., Ragan-Kelley, B., Pérez, F., Granger, B.E., Bussonnier, M., Frederic, J., Kelley, K., Hamrick, J.B., Grout, J., Corlay, S., et al.: Jupyter notebooks-a publishing format for reproducible computational workflows. In: F. Loizides, B. Schmidt (eds.) Positioning and Power in Academic Publishing: Players, Agents and Agendas, pp. 87–90 (2016)
3. Knuth, D.E.: Literate programming. CSLI Lecture Notes, Stanford, CA: Center for the Study of Language and Information (CSLI), 1992 (1992)

[13] https://github.com/jupyter/jupyter/wiki/Jupyter-kernels
[14] http://wiki.ros.org/Client%20Libraries

4. Millman, K.J., Pérez, F., Stodden, V., Leisch, F., Peng, R.: Developing open-source scientific practice. Implementing Reproducible Research **149** (2014)
5. Pérez, F., Granger, B.E.: Ipython: a system for interactive scientific computing. Computing in Science & Engineering **9**(3) (2007)
6. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: Ros: an open-source robot operating system. In: ICRA workshop on open source software, vol. 3, p. 5. Kobe, Japan (2009)
7. Shen, H.: Interactive notebooks: Sharing the code. Nature News **515**(7525), 151 (2014)
8. Tellez, R.: A thousand robots for each student: Using cloud robot simulations to teach robotics. In: M. Merdan, W. Lepuschitz, G. Koppensteiner, R. Balogh (eds.) Robotics in Education, pp. 143–155. Springer International Publishing, Cham (2017)
9. Vallisneri, M., Kanner, J., Williams, R., Weinstein, A., Stephens, B.: The ligo open science center. In: Journal of Physics: Conference Series, vol. 610, p. 012021. IOP Publishing (2015)
10. Vasilev, V., Canal, P., Naumann, A., Russo, P.: Cling–the new interactive interpreter for root 6. In: Journal of Physics: Conference Series, vol. 396, p. 052071. IOP Publishing (2012)
11. White, R., Christensen, H.: Ros and docker. In: Robot Operating System (ROS), pp. 285–307. Springer (2017)
12. Wolfram, S.: Mathematica: a system for doing mathematics by computer. Addison-Wesley (1991)