





PYDAQ: Data Acquisition and Experimental Analysis with Python

Samir Angelo Milani Martins ^{1,2} 

¹ Department of Electrical Engineering at Federal University of São João del-Rei, Brazil. ² GCoM - Modeling and Control Group at Federal University of São João del-Rei, Brazil.  Corresponding author

DOI: [10.21105/joss.05662](https://doi.org/10.21105/joss.05662)

Software

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

Editor: [Kyle Niemeyer](#)  

Reviewers:

- [@galessiorob](#)
- [@nataliakeles](#)

Submitted: 10 March 2023

Published: 14 December 2023

License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](https://creativecommons.org/licenses/by/4.0/)).

Summary

System identification is a relevant research topic that aims to find mathematical models using acquired data. One of the main contributions is the work of Ljung (1999), which was substantially developed over the years (Lacerda Junior et al., 2019; Martins & Aguirre, 2016). Among system identification tools, SysIdentPy (Junior et al., 2020) uses Python in a very straightforward way for system modeling through empirical data, while Narmax (Ayala et al., 2020) promises to obtain models using R language.

As pointed out by Ljung (1999), experimental data are necessary for obtaining black-box models, and this is exactly where PYDAQ find its place. PYDAQ is a Python tool which was primarily developed for experiments with empirical data, either sending and/or acquiring data using simple Graphical User Interfaces or command line, with few (or no) lines of code required, using different solutions provided by the market (NIDAQ and Arduino). Even a researcher with no programming skills is able to use PYDAQ easily and quickly for data acquisition.

In what follows it will be shown how PYDAQ can be use by any scientist, its advantages and features for quickly and effective data acquisition experiments, even if the scientist has no programming skills.

Statement of need

Any scientist or student who needs to acquire data in an easy and quick way, in three line of code (LOC), are the target audience of this manuscript/package. PYDAQ is a solution that aims to allow the user to perform experiments from data acquisition to signals generation using a Graphical User Interface.

Despite this, for advanced users, it is also possible to use PYDAQ through command line, as shown in the [documentation](#), allowing PYDAQ to be integrated in real time with well-known modeling tools.

To contextualize the importance of PYDAQ, there are full papers dealing only with data acquisition (Koerner et al., 2020; Yang, 2019). It should be clear at this point that data acquisition is commonly only the first step in an empirical scientific procedure, such as in the work of Ostrovskii et al. (2020). Therefore, this step should not take too much energy of a researcher, since he/she needs to be full of energy to continue the research project.

Further, PYDAQ deals from different type of data acquisition cards, from simple and cheaper Arduino boards up to NIDAQ devices, allowing the execution from simple to complex experiments. Also, since PYDAQ can be also used as a command line tool, it can be easily incorporated to be used in production along with any available mathematical tool, such as SysIdentPy (Junior et al., 2020) or SciKitLearn (Pedregosa et al., 2011).

In literature there are packages that deals with NIDAQ devices, such as the NSLS-II tools (Koerner et al., 2020). However, they need several lines of codes in order to make a single data acquisition, and works only with expensive and proprietary boards. Besides, as far as I know, there is no Graphical User Interface open software that allows instantly and easily data acquisition with Python, this being another feature of PYDAQ.

Because of the above-mentioned facts, PYDAQ can be used also to introduce new scientist in the System Identification research area. Also, PYDAQ can be used in teaching, during engineering courses and in low-cost laboratories' implementation, once Arduino boards are quite cheap and easy to find. Graphical User Interfaces also allows the user to be directly connected with the subject, as explicitly said by Silva et al. (2018).

In what follows examples of how to use PYDAQ will be presented, as well as future research topics. Further details can also be found in the [documentation](#).

Examples

The fastest way to install PYDAQ is using pip:

```
pip install pydaq
```

[Figure 1](#) and [Figure 2](#) depict the Graphical User Interface developed for Data Acquisition using Arduino or any NIDAQ board.

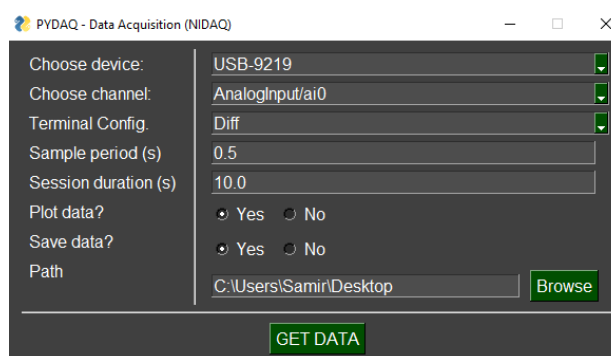


Figure 1: Data Acquisition through NIDAQ.

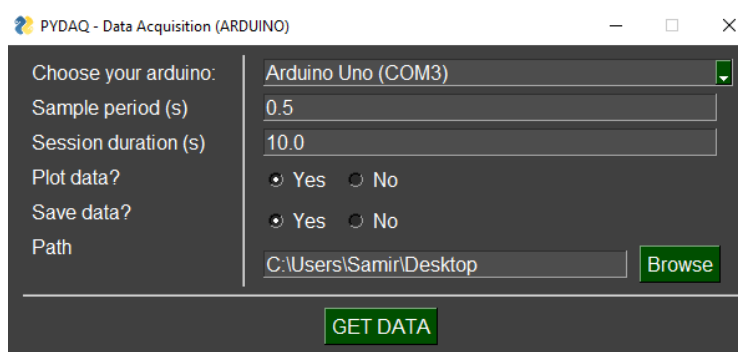


Figure 2: Data Acquisition through Arduino.

To start them, only three line of codes (LOC) are necessary, including one for importing PYDAQ:

```

from pydaq.get_data import Get_data
# Class Get_data
g = Get_data()

# Arduino or NIDAQ - Use ONE of the following lines
g.get_data_nidaq_gui() # For NIDAQ devices
g.get_data_arduino_gui() # For arduino boards

```

Similarly, to send data, only three LOC are required, as showed up in what follow:

```

from pydaq.send_data import Send_data

# Class Send_data
s = Send_data()

# Arduino or NIDAQ - Use ONE of the following lines
s.send_data_nidaq_gui()
s.send_data_arduino_gui()

```

If the user decides to save data, it will be saved in .dat format, located at the path defined in the GUI (Desktop is the default path). Figure 3 shows an example of how data will be saved: i) one file (time.dat) with the timestamp, in seconds, when each sample was acquired; ii) file data.dat contains acquired values.

| Sample | Time (s) | Value |
|--------|----------|----------------------|
| 1 | 0.0 | -0.0748121783025371 |
| 2 | 0.5 | 0.04000962019024015 |
| 3 | 1.0 | 0.014939905103439398 |
| 4 | 1.5 | 0.13772428856636815 |
| 5 | 2.0 | 0.1346308073181395 |
| 6 | 2.5 | 0.1493221610380507 |
| 7 | 3.0 | 0.16034067632798413 |
| 8 | 3.5 | 0.2726394100570327 |
| 9 | 4.0 | 0.2921104605263746 |

Figure 3: Example of acquired data.

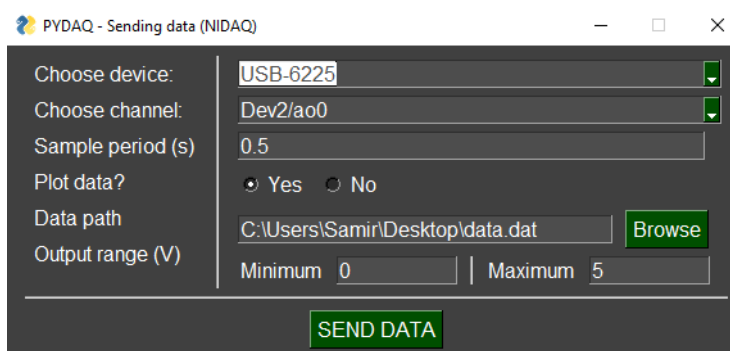


Figure 4: GUI for sending data - Arduino.

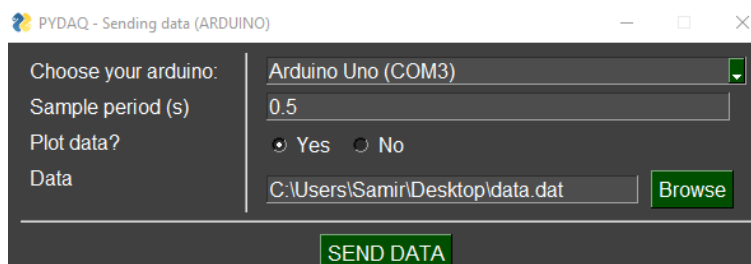


Figure 5: GUI for sending data - NIDAQ.

It should be emphasized that once this code is executed, a Graphical User Interface will manifest on the screen, according to the board selected by the user, as shown in [Figure 4](#) and [Figure 5](#).

Options are straight-forward and ease to understand. For further details and to check how to use the same functionality using a command line the reader are invited to check the [documentation](#).

It is noteworthy that any signal can be generated and applied to a physical system using the presented GUI, being the used board the main constraint. Data can be either generated manually or using a library (e.g., NumPy) to create signals as sine waves, PRBS (Pseudo-Random Binary Signal) or other signal required to be a persistently exciting input, as necessary for system identification ([Billings, 2013](#); [Ljung, 1999](#)).

Step-response is a common way to test a system and acquire data, in order to find a model, as well as system time constant and gain. To facilitate this procedure, a step-response GUI was also created and can be seen in [Figure 6](#) and [Figure 7](#). To use them, user should type the command:

```
from pydaq.step_response import Step_response

# Class Step_Response
s = Step_response()

# Arduino or NIDAQ - Use ONE of the following lines
s.step_response_nidaq_gui()
s.step_response_arduino_gui()
```

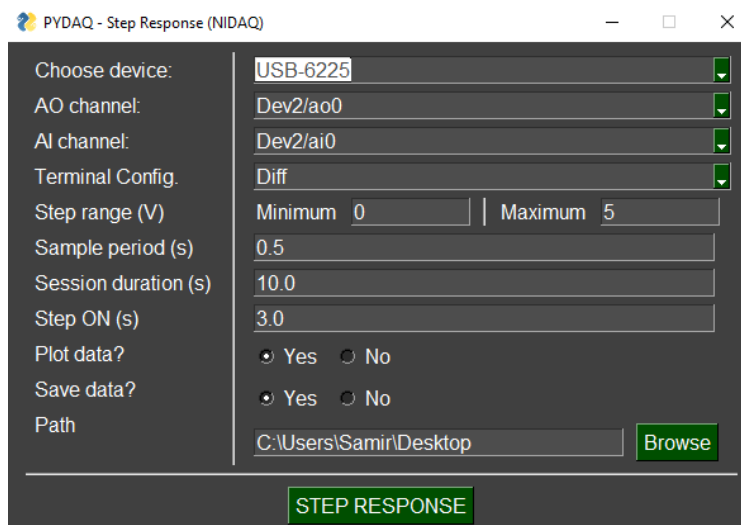


Figure 6: Step Response GUI - NIDAQ.

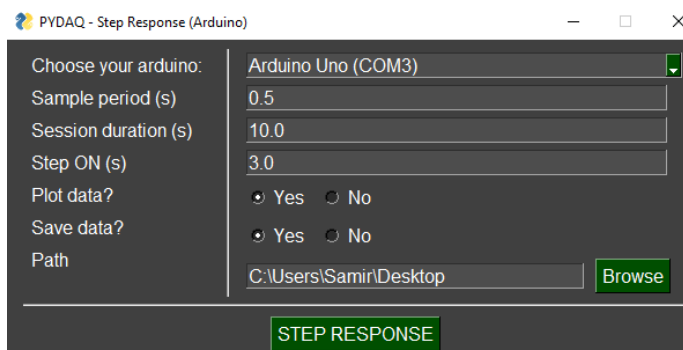


Figure 7: Step Response GUI - Arduino.

Here the user can define when the step will be applied, as well as where data will be saved. Figure 8 and Figure 9 show data that were empirically-acquired with PYDAQ. In the figures the user will find labels, functionality (Sending Data/Data Acquisition/Step Response), device/channel (for NIDAQ boards) or COM port used (for Arduino devices).

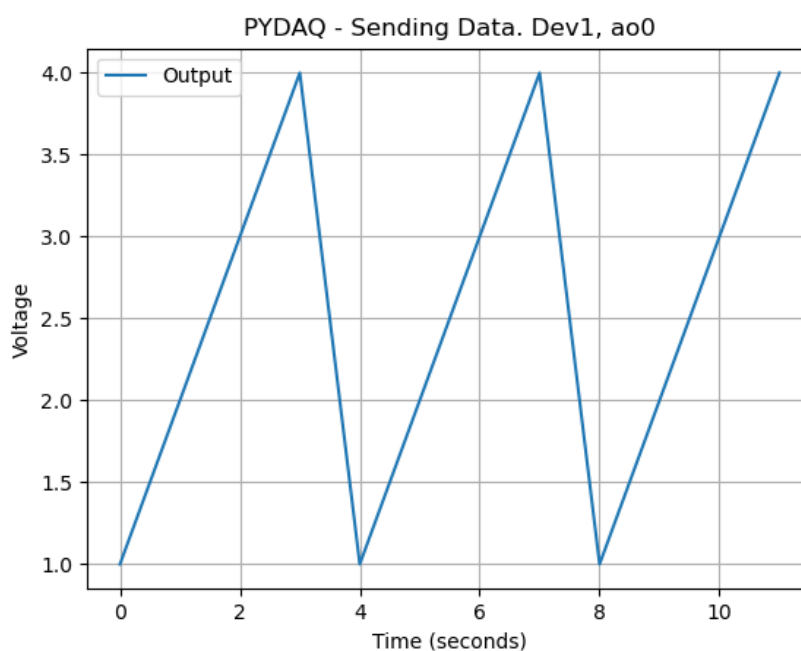


Figure 8: Data acquired using a NIDAQ board.

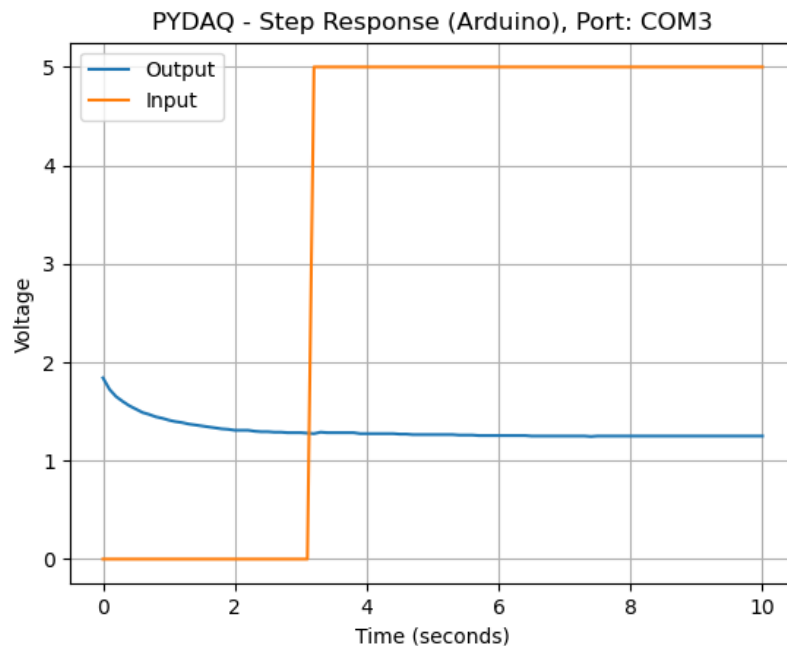


Figure 9: Data generated by a step-response experiment.

Examples showed above shed light in some functionalities of PYDAQ. For further details and for command line use, the reader is welcome to consult the full [documentation](#).

Future Work

Future releases will include real-time and data-driven system identification using linear and nonlinear approaches. Also, real-time model based controllers will be implemented through PYDAQ. Saving data in an SQL server is a future feature, as well.

References

- Ayala, H. V. H., Gritti, M. C., & Santos Coelho, L. dos. (2020). An R library for nonlinear black-box system identification. *SoftwareX*, 11, 100495. <https://doi.org/10.1016/j.softx.2020.100495>
- Billings, S. A. (2013). *Nonlinear system identification: NARMAX methods in the time, frequency, and spatio-temporal domains* (p. 574). John Wiley & Sons. <https://doi.org/10.1002/9781118535561>
- Junior, W. R. L., Andrade, L. P. C. da, Oliveira, S. C. P., & Martins, S. A. M. (2020). SysIdentPy: A Python package for system identification using NARMAX models. *Journal of Open Source Software*, 5(54), 2384. <https://doi.org/10.21105/joss.02384>
- Koerner, L. J., Caswell, T. A., Allan, D. B., & Campbell, S. I. (2020). A Python instrument control and data acquisition suite for reproducible research. *IEEE Transactions on Instrumentation and Measurement*, 69(4), 1698–1707. <https://doi.org/10.1109/TIM.2019.2914711>
- Lacerda Junior, W. R., Martins, S. A. M., Nepomuceno, E. G., & Lacerda, M. J. (2019). Control of hysteretic systems through an analytical inverse compensation based on a NARX model. *IEEE Access*, 7, 98228–98237. <https://doi.org/10.1109/access.2019.2926057>

- Ljung, L. (1999). *System identification: Theory for the user* (2nd ed.). Prentice-Hall. ISBN: 0-13-656695-2
- Martins, S. A. M., & Aguirre, L. A. (2016). Sufficient conditions for rate-independent hysteresis in autoregressive identified models. *Mechanical Systems and Signal Processing*, 75, 607–617. <https://doi.org/10.1016/j.ymssp.2015.12.031>
- Ostrovskii, V. Y., Nazare, T. E., Martins, S. A. M., & Nepomuceno, E. G. (2020). Temperature as a chaotic circuit bifurcation parameter. *2020 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, 154–157. <https://doi.org/10.1109/EIConRus49466.2020.9038964>
- Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. <https://doi.org/10.48550/arXiv.1201.0490>
- Silva, P. H. O., Nardo, L. G., Martins, S. A. M., Nepomuceno, E. G., & Perc, M. (2018). Graphical interface as a teaching aid for nonlinear dynamical systems. *European Journal of Physics*, 39(6), 065105. <https://doi.org/10.1088/1361-6404/aae35c>
- Yang, H. (2019). Design and implementation of data acquisition system based on Scrapy technology. *2019 2nd International Conference on Safety Produce Informatization (IICSPI)*, 417–420. <https://doi.org/10.1109/IICSPI48186.2019.9096044>