ing.grid

RESEARCH ARTICLE

plotID - a toolkit for connecting research data and visualization

Martin Hock [1]
Hannes Mayr [1]
Manuela Richter [1]
Jan Lemmer
Peter F. Pelz [1]

1. Chair of Fluid Systems, Technische Universität Darmstadt, Darmstadt.

Date Published:

Reviewer:

Licenses:
This article is licensed under: 🅭🅯

Keywords:
research data management, visualization, figure, plot, mapping, referencing, ID

Data availability:
Data can be found here:
example.py

Software availability:
Software can be found here:
`git.rwth-aachen.de/plotid/plotid_python`

Abstract. The highest amount of published information on paper is contained in visualizations such as 2D and or 3D plots. Supporting a generic research workflow, plotID provides tools that can a) create and anchor a reference (ID code, URL,...) for and b) package figures, data, code and parameters used to create the figure. The code is provided as tools with small impact, that need to be used consciously by the researcher and does not aim to relieve the researcher of his duty to keep his digital working environment organized. The exported packages help immensely to make results reusable and repeatable. The initial implementation was created in Matlab and used internally before rewriting the tool in the Python programming language, for easier distribution and adaption to diverse environments.

1 Statement of need

In a typical research workflow, the researcher collects data by performing experiments, simulations, evaluations of existing data and other sources. While assessing the available data, visualizations are created to make data easier to interpret. Some of the figures created at this early stage are still useful at later stages up to the publication of results, but the rough scripts and data used to create them have been lost or changed. Other figures are created to illustrate context, connections or support claims made in a paper.

To reduce the effort of organizing figures along with all necessary data and metadata for later review and reuse, plotID was developed. The authors could not find any other software that aimed at this specific task, although some frameworks aiming to organize the full workflow achieve similar snapshots by version controlling all software and data in repositories. (See for example DataLad[6] Labelling the figure with a corresponding ID, however, is unique to plotID.

2 Design

The tool has been designed to be integrated seamlessly into existing scripts. To keep the usage simple, no GUI was created. The first steps need to be easy. By copying example code into

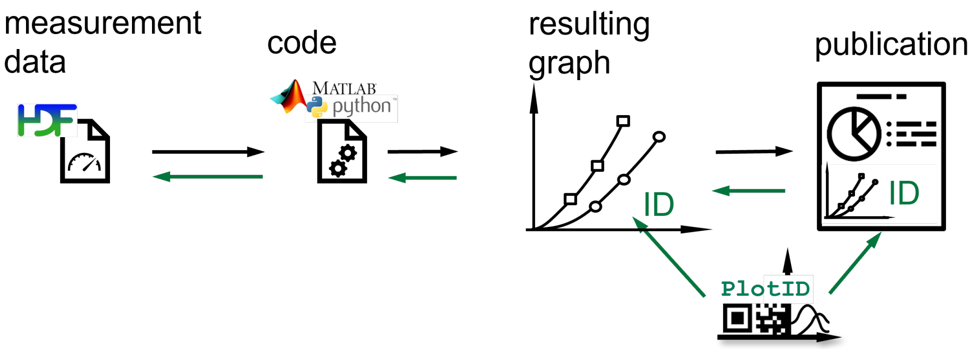

Figure 1: Research workflow from left to right; Afterwards following the chain of references from right to left
'plotID-workflow' by Martin Hock, licensed under CC-BY-SA 4.0

the users' own scripts and executing this successfully, users stay encouraged to continue using plotID and explore its advanced functionalities.

plotID aims to help during the early research process to decrease the work of making publications reproducible later on. It provides two importable modules and can be integrated into the user's scripts with one line of code each.

The first module creates a (unique) ID and stamps this ID onto an object containing a visualization, while the second module helps organize all relevant code, software, and data that went into creating this graphic, into one complete package.

If this specific visualization is later chosen to be included in a publication, the ID can be replaced by a permanent identifier like a DOI and the package of code, software and data can be published at the location referenced by the DOI. The ID on in the published paper will then directly reference the data, software and code used to create it, hence curating reproducibility.

3 Implementation

The first version of plotID was implemented in Matlab since this is the most widely used programming language in the local working environment and the language the authors had the highest familiarity with. After reaching a usable state, the focus shifted to rewriting the tool in Python, the second most used programming language (locally). In addition to being widely used in the engineering and research community, Python is non-proprietary, open source, easy to install or even shipped along many operating systems. Python also offers a package index (*PyPI*) and installer (*pip*) for easy distribution of software packages.

Currently, to run plotID a Python version ≥3.10 is required. The code is open source under the Apache-v2.0 license. The current release version is v0.1.2 showing a pre-alpha state.

4 Example script

The following script shows how plotID is used.

```python
# %% Import modules
import numpy as np
import matplotlib.pyplot as plt
from plotid.tagplot import tagplot
from plotid.publish import publish

# %% Set Project ID
PROJECT_ID = "MR04_"

# %% Create sample data
x = np.linspace(0, 10, 100)
y = np.random.rand(100) + 2
y_2 = np.sin(x) + 2

# %% Create sample figures

# 1. figure
IMG1 = 'image1.png'
FIG1 = plt.figure()
plt.plot(x, y, color='black')
plt.plot(x, y_2, color='yellow')
plt.savefig(IMG1)

# 2. figure
IMG2 = 'image2.png'
FIG2 = plt.figure()
plt.plot(x, y, color='blue')
plt.plot(x, y_2, color='red')
plt.savefig(IMG2)
```

In this part, the plotID modules and those necessary to create figures and images are imported. The variable *PROJECT_ID* is set to provide a static part of the ID. Random data is used to create two figures with matplotlib as well as image files.

```python
# %% TagPlot

# If multiple figures should be tagged, they must be provided as list.
FIGS_AS_LIST = [FIG1, FIG2]
IMGS_AS_LIST = [IMG1, IMG2]

# Example for how to use tagplot with matplotlib figures
[TAGGED_FIGS, ID] = tagplot(FIGS_AS_LIST, 'matplotlib',
        prefix=PROJECT_ID, id_method='time', location='west')

```

```
# Example for how to use tagplot with image files
# [TAGGED_FIGS, ID] = tagplot(IMGS_AS_LIST, 'image', prefix=
    PROJECT_ID,
#                            id_method='random', location='west')
```

Both matplotlib objects are tagged with a generated ID each, all in one line of code. Tagging the image files has been commented out in this case.

```
# %% Publish
```

Files (README.md and LICENSE) and a folder from the code repository are used in place of research data files. The folder ending with ’-exports’ is the destination, and ’testimage’ is a freely chosen name for the exported image files.

This also shows that the workflow does not depend on any kind of file format or pre-organized structures. Any kind of data can be used, and even if the library creating the visualization is not (yet) supported, the resulting image file can still be tagged.

5 Core functions

The core functions of plotID are *tagplot()* and *publish()*. *tagplot()* generates an ID and adds this ID to the figure object. *publish()* saves the figure object and image file, along with the script file, plotID was called from – everything necessary to recreate the visualization from scratch. A third step to replace an existing ID with a previously registered PID (DOI, hdl, ...) for permanent publication is planned.

5.1 tagplot()

The *tagplot()* function creates an ID and tags the figure object with this ID.

5.1.1 ID

tagplot() creates a unique ID (unique in a restricted system), that consists of a static part and a generated part. The static part is handed over as a parameter and is meant to be used to identify a project or organizational unit to which the figure is assigned. The generated part is by default created from the UNIX-Time stamp in hexadecimal form. As an alternative option, a random number generator can be used. The implementation of the ID is modular, enabling easy implementation of individual needs or sources for IDs.

5.1.2 tagging

In Python, there are multiple available packages that can produce visualizations from data. Adding an ID needs to be implemented for many of these engines separately. For now plotID supports figures created with *matplotlib* and raw image files. The ID is added as an attribute to the object and the graphical, visible item.

5.1.3 arguments

Necessary input arguments for *tagplot(figs, engine[, prefix, id_method, location])*:

- *figs*: the figure object or a list of objects, that is to be tagged
- *engine*: the plot/image engine to be used (currently only 'matplotlib' and 'image' (for plain image files) are supported)

Optional input arguments are:

- *prefix*: to define a static part of each created ID (prefix='Ing.grid-'). Type of string.
- *id_method*: to define how the unique part of the ID is created ('random', 'time'). Type of string.
- *location*: to define the position the ID is displayed in, relative to the full graphical object (cardinal directions like 'west', custom inputs for rotation and position are to be implemented). Type of string.

Output arguments are the tagged object and ID, if a list of objects was input, then the output is a list as well.

At this point the figure object can still be modified, for example, to adjust colours or positioning or recreate the full plot before exporting a final version.

5.2 publish()

This function starts the export process. The source files of the processed data, the visualization (including the tagged ID), and the script hosting the call to the publish function are copied together into a destination folder.

5.2.1 script

A function in Python has access to the file path of the script which it was called from. With this, the code for calculations can easily be gathered. For this reason, *publish()* cannot be called from the command line or from within a script that has been started with the 'python -m' flag. For required packages, the 'import' lines of the script can be parsed into a requirements.txt, which can easily be installed with the Python package installer *pip*. This has not yet been implemented. Furthermore, the user has to take care of including additional function files as data paths, that have not been imported but are still accessed by the executed script.

5.2.2 data files

Data files are handed over as a list of file or folder paths. Ideally, the script already manages a list of all files that are read during the execution of the script. It is up to the user to control this. By default, the data files are copied to each exported package. For large data files, the *centralized* flag is intended. The data files are copied to a central folder, relative to the export packages. For further exports, the data files are compared to the ones already copied and only copied if new data files are present. With this, a publication on a data repository could encompass the data files in addition to multiple ”satellite” folders containing the specific script, parameters and graphics.

For HDF5 files, each package can contain an empty HDF5 file that only contains a link to the ”real” central data file. While this has proven to be useful in the Matlab implementation, the Python version aims to include the ’centralized’ option in a future release.

5.2.3 arguments

Necessary input arguments for *publish(src_datapath, dst_path, figure, plot_name[, ...])* are:

- *src_datapath*: This can be a single or a list of file or folder paths for source data and additional function files. The type is a string or a list of strings.
- *dst_path*: This is the destination folder path. If it does not exist, the folder will be created. The type is a string.
- *figure*: This is a figure object, the exact class depends on the plot engine used. This object will be turned into an image file.
- *plot_name*: This is the name for the graphics objects. The type is a string or list of strings. If a single name is passed for multiple objects, a raising number will be added.

Optional input arguments:

- *data_storage*: Currently only ’individual’ and ’centralized’ are available. ’Individual’ will store all data in each exported package, while ’centralized’ stores the data files in a central folder separate from the packages containing script and image files.

6 Distribution

Providing easy ways to acquire and use the software is important for adoption. Currently, the following methods are available and described in the repository’s[5] README file.

6.1 Source Code

The plain source code is publicly available on a GitLab repository located under git.rwth-aachen.de/plotID/plotID_python/[5] and can be directly downloaded or cloned with git.

```
git clone https://git.rwth-aachen.de/plotid/plotid_python.git
cd plotid_python
pip install -r requirements.txt
pip install .
```

6.2 Python Package

During the current development state, the PyPI test instance[10] is used. With the release of a v.1.0, the package will be listed in the official Python Package Index[9]. The installation is done with the following command:

```
pip install plotid --extra-index-url=https://test.pypi.org/simple/
```

6.3 Debian Package

A debian package (dpkg) is planned to be provided in the repository git.rwth-aachen.de/plotid/-plotid_debian[8]. A .deb file can be installed via *dpkg* or *apt-get* on compatible operating systems.

7 Unit tests

Python offers various libraries for unit testing. plotID is using the *unittest* module[11], which is delivered with Python by default. Tests for each function are defined in the *tests* folder, along with the *runner_test.py* script which organizes the execution of the tests, by discovering the test files based on their location. The *unittest* module also measured the code covered by the tests, and total coverage of less than 95% counts as failed. The tests are executed by a GitLab CI/CD pipeline[3] with every commit and merge request along with Pylint[2] and Flake8[1] to check against coding style, programming errors and cyclomatic complexity. Commits that fail the pipeline tests cannot be merged into the main branch and will not make it into a release version.

8 Documentation

To ensure easy access and understanding of the code, Python docstrings[7] have been implemented in the source code from the beginning. The docstrings are compiled into HTML using the Sphinx[12] python package and GitLab CI-CD[3] creating an automatically generated API reference. The documents are hosted using GitLab Pages[4]. This documentation[13] will be improved by adding the readme, example code, example use cases and an introductory text until version 1.0.

9 Conclusion

The idea of plotID is simple yet. As with most research data management operations, the benefit for the additional work presents at a later time – although in this case, it presents for the creator of data or visualization and not only for later reuse. The code and open source implementation is still work-in-progress, but the core functionality is present. Bug reports, merge requests with code, ideas for features and all feedback are welcome and best voiced in the GitLab repository.

10 Acknowledgements

We acknowledge the help from Jan Stifter and Benjamin Hermann with testing the software and feedback on the user interface.

The authors would like to thank the Federal Government and the Heads of Government of the Länder, as well as the Joint Science Conference (GWK), for their funding and support within the framework of the NFDI4Ing consortium. Funded by the German Research Foundation (DFG) - project number 442146713.

11 Roles and contributions

Martin Hock: Conceptualization, Methodology, Coding, Tests, Writing – original draft

Hannes Mayr: Coding, Tests, Methodology

Manuela Richter: Conceptualization, Methodology, Coding

Jan Lemmer: Conceptualization, Methodology

Peter F. Pelz: Project administration, Supervision, Funding Acquisition

References

[1] GitHub. *flake8/index.rst at main · PyCQA/flake8*. 2022. URL: https://github.com/PyCQA/flake8 (visited on 08/29/2022).

[2] GitHub. *PyCQA/pylint: It's not just a linter that annoys you!* 2022. URL: https://github.com/PyCQA/pylint (visited on 08/29/2022).

[3] *GitLab CI/CD | GitLab*. 2022. URL: https://docs.gitlab.com/ee/ci/ (visited on 08/19/2022).

[4] *GitLab Pages | GitLab*. 2022. URL: https://docs.gitlab.com/ee/user/project/pages/ (visited on 08/29/2022).

[5] GitLab RWTH Aachen. *PlotID / plotID_python · GitLab*. 2022. URL: https://git.rwth-aachen.de/plotid/plotid_python (visited on 08/19/2022).

[6] Yaroslav O. Halchenko et al. “DataLad: distributed system for joint management of code, data, and their relationship”. In: *Journal of Open Source Software* 6.63 (2021), p. 3262. DOI: 10.21105/joss.03262. URL: https://doi.org/10.21105/joss.03262.

[7] *PEP 257 – Docstring Conventions | peps.python.org*. 2022. URL: https://peps.python.org/pep-0257/ (visited on 08/29/2022).

[8] *PlotID / plotID_debian · GitLab*. 2022. URL: https://git.rwth-aachen.de/plotid/plotid_debian (visited on 08/19/2022).

[9] PyPI. *PyPI · The Python Package Index*. 2022. URL: https://pypi.org/ (visited on 08/19/2022).

[10] PyPI. *TestPyPI · The Python Package Index*. 2022. URL: https://test.pypi.org/ (visited on 08/19/2022).

[11] *unittest — Unit testing framework — Python 3.10.6 documentation*. 2022. URL: https://docs.python.org/3/library/unittest.html (visited on 08/29/2022).

[12] *Welcome — Sphinx documentation*. 2022. URL: https://www.sphinx-doc.org/en/master/ (visited on 08/29/2022).

[13] *Welcome to PlotID's documentation! — plotID 0.1.2 documentation*. 2022. URL: https://plotid.pages.rwth-aachen.de/plotid_python/ (visited on 08/29/2022).