Geometric Calibration Documentation

Release 0.2.0

Matteo Rossi

Feb 22, 2021

Contents:

CHAPTER 1

Geometric Calibration

A utility to perform Geometric Calibration of a C-Arm Structure mounted on a robotic system

- Free software: MIT license
- Documentation: https://geometric-calibration.readthedocs.io.

1.1 Features

- C-arm calibration in two supported modality: 3D (CBCT), planar (AP/RL)
- Support for reading projection in .raw or .hnc (Varian) format
- Plot calibration results
- Save LUT in .txt file

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install Geometric Calibration, run this command in your terminal:

```
$ pip install geometric_calibration
```

This is the preferred method to install Geometric Calibration, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for Geometric Calibration can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/mrossi93/geometric_calibration
```

Or download the tarball:

```
$ curl -OJL https://github.com/mrossi93/geometric_calibration/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use Geometric Calibration from command line as a script:

- Open file gCal_config.ini and change the parameter for calibration
- Then, from your command line type:

```
geometric_calibration --config gCal_config.ini
```

You can also see an helper typing:

```
geometric_calibration -h
```

To use Geometric Calibration in a project:

```
import geometric_calibration
```

CHAPTER 4

API References

4.1 geometric_calibration package

Top-level package for Geometric Calibration.

4.1.1 Submodules

geometric_calibration.cli module

geometric_calibration.geometric_calibration module

geometric_calibration.reader module

This module contains some function for I/O purposes.

geometric_calibration.reader.**read_bbs_ref_file**(*filename*)
Read phantom reference file with bbs coordinates

Parameters **filename** (*str*) – path to file

Returns Array containing bbs coordinates [x,y,z]

Return type numpy.array

geometric_calibration.reader.**read_img_label_file**(*filename*)
Read imgLabels.txt file contained in .raw projection's directory. This File contains information about path and the gantry angle of every .raw projection.

Parameters **filename** (*str*) – path to file

Returns list with path and list with angles for every row in imgLabels.txt file

Return type list

geometric_calibration.reader.**read_projection_hnc**(*filename*, *dim*)
Read .hnc file and load it into a Numpy array.

Parameters

- **filename** (*str*) – path to file
- **dim** (*list*) – Dimension of image

Returns array containing loaded .raw image

Return type numpy.array

geometric_calibration.reader.**read_projection_raw**(*filename*, *dim*)
Read .raw file and load it into a Numpy array.

Parameters

- **filename** (*str*) – path to file
- **dim** (*list*) – Dimension of image

Returns array containing loaded .raw image

Return type numpy.array

geometric_calibration.dlt module

geometric_calibration.dlt.**DLTcalib**(*nd*, *xyz*, *uv*, *uv_ref=None*)
Camera calibration by DLT using known object points and their corresponding image points. The coordinates (x,y,z and u,v) are given as columns and the different points as rows. There must be at least 6 calibration points for the 3D DLT.

Parameters

- **nd** (*int*) – dimensions of the object space, typically 3
- **xyz** (*numpy.array*) – coordinates in the object 3D space
- **uv** (*numpy.array*) – coordinates in the image 2D space
- **uv_ref** (*numpy.array, optional*) – [description]. Defaults to None.

Raises

- **ValueError** – Dimension not supported
- **ValueError** – xyz and uv have different number of points
- **ValueError** – Wrong dimension for coordinates
- **ValueError** – Insufficient number of points

Returns

array of 11 parameters of the calibration matrix, followed by error of the DLT (mean residual of the DLT transformation in units of camera coordinates).

Return type numpy.array, float

geometric_calibration.dlt.**FixAngles**(*rm*)

geometric_calibration.dlt.**Normalization**(*nd*, *x*)
Normalization of coordinates (centroid to the origin and mean distance of sqrt(2 or 3).

Parameters

- **nd** (*int*) – number of dimensions, typically 3

- **x** (*numpy.array*) – the data to be normalized (directions at different columns and points at rows)

Returns

the transformation matrix (translation plus scaling), the transformed data

Return type numpy.array, numpy.array

geometric_calibration.dlt.**VerifyAngles**(*outOfPlaneAngleRAD*, *gantryAngleRAD*, *inPlaneAngleRAD*, *referenceMatrix*)

geometric_calibration.dlt.**decompose_camera_matrix**(*L*, *image_size*, *pixel_spacing*)

geometric_calibration.dlt.**extract_param_from_matrix_Rit**(*camera_matrix*)

geometric_calibration.utils module

This module contains some utility function for image manipulation.

class geometric_calibration.utils.**DraggablePoints**(*artists*, *tolerance=15*)

Bases: object

Draggable points on a matplotlib figure.

Returns DraggablePoints object

Return type *DraggablePoints*

__init__(*artists*, *tolerance=15*)

Initialize an instance of DraggablePoints object and superimpose it on the current matplotlib Figure.

Parameters

- **artists** (*list*) – list of matplotlib Circles with coordinates (x,y) in pixel coordinates.
- **tolerance** (*int, optional*) – Tolerance for mouse selection when dragging on screen. Defaults to 15.

get_coord()

Obtain current coordinates (x,y) of points.

Returns An array Nx2 containing coordinates for N point (x,y).

Return type numpy.array

on_close(*event*)

Event Handler for closure of figure.

Parameters **event** (*event*) – Event that triggers the method

on_key_pressed(*event*)

Event Handler for "Enter" key pression.

Parameters **event** (*event*) – Event that triggers the method

on_key_released(*event*)

Event Handler for any key released.

Parameters **event** (*event*) – Event that triggers the method

on_motion(*event*)

Event Handler for mouse movement during dragging of points.

Parameters **event** (*event*) – Event that triggers the method

on_press(*event*)
Event Handler for mouse button pression.

Parameters **event** (*event*) – Event that triggers the method.

on_release(*event*)
Event Handler for mouse button release.

Parameters **event** (*event*) – Event that triggers the method

geometric_calibration.utils.**create_camera_matrix**(*detector_orientation*, *sdd*, *sid*, *pixel_spacing*, *isocenter*, *proj_offset*, *source_offset*, *image_size*)
Generate projection matrix starting from extrinsic and intrinsic parameters (according to the rules of creation of a projection matrix).

Parameters

- **detector_orientation** (*numpy.array*) – Nx3 array containing rotations of the image's plane [rot_x, rot_y, rot_z]
- **sdd** (*float*) – Source to Detector distance
- **sid** (*float*) – Source to Isocenter distance
- **pixel_spacing** (*list*) – Pixel dimension in mm
- **isocenter** (*numpy.array*) – Coordinates of isocenter
- **proj_offset** (*list*) – Detector offset, expressed as [offset_x, offset_y]
- **source_offset** (*list*) – Source offset, expressed as [offset_x, offset_y]
- **image_size** (*list*) – Dimension of image

Returns 3x4 camera matrix

Return type numpy.array

geometric_calibration.utils.**drag_and_drop_bbs**(*projection*, *bbs_projected*)
Drag&Drop Routines for bbs position's correction.

Parameters

- **projection** (*str*) – Path to the projection (.raw of .hnc) file
- **bbs_projected** (*numpy.array*) – Array Nx2 with N BBs yet projected on image plane

Returns Array Nx2 containing the updated coordinates for N BBs

Return type numpy.array

geometric_calibration.utils.**get_grayscale_range**(*img*)
New grayscale range for .raw or .hnc images, since original values are too bright. New range is computed between min of image and one order of magnitude less than original image. Worst case scenario [0, 6553.5] (since image is loaded as uint16).

Parameters **img** (*numpy.array*) – Array containing the loaded .raw or .hnc image

Returns Grayscale range for current projection

Return type list

geometric_calibration.utils.**project_camera_matrix**(*coord_3d*, *camera_matrix*, *image_size*)

Project 3D data (x,y,z) in world coordinate system to 2D (u,v) coordinate system using camera matrix computed with *geometric_calibration.utils.create_camera_matrix()* function.

Parameters

- **coord_3d** (*numpy.array*) – Nx3 array containing 3D coordinates of points (x,y,z) in world coordinate system.
- **camera_matrix** (*numpy.array*) – 3x4 projection matrix obtained combining both extrinsic and intrinsic parameters.
- **image_size** (*list*) – Dimension of the image

Returns Nx2 array containing 2D coordinates of points (u,v) projected on image plane (u,v)

Return type numpy.array

geometric_calibration.utils.**search_bbs_centroids**(*img*, *ref_2d*, *search_area*, *image_size*, *mode*, *debug_level=0*)

Search bbs based on projection.

Starting from the updated coordinates, define a search area around them and identify the BBs centroid as the center of a circle or an ellipse (based on mode argument). This function automatically set as (np.nan, np.nan) the coordinates of BBs outside image space, too dark or too close to another BBs.

Parameters

- **img** (*numpy.array*) – Array containing the loaded .raw or .hnc file
- **ref_2d** (*numpy.array*) – Nx2 array containing the coordinates for BBs projected on img
- **search_area** (*int*) – Size of the region in which to search for centroids. Actual dimension of the area is a square with dimension (2* search_area,2*search_area)
- **image_size** (*list*) – Dimension of img
- **mode** (*str*) – Centroid search modality. It can be “circle” or “ellipse”. Ellipse is slower but provide better results in general.
- **debug_level** (*int, optional*) – Level for debug messages, 0 means no debug messages, 1 light debug and 2 hard debug. Defaults to 0.

Returns Nx2 array containing coordinates for every centroids found (x,y)

Return type numpy.array

geometric_calibration.utils.**search_bbs_centroids_hough**(*img*, *ref_2d*, *search_area*, *image_size*, *mode*, *debug_level=0*)

Search bbs based on projection.

Starting from the updated coordinates, define a search area around them and identify the BBs centroid as the center of a circle or an ellipse (based on mode argument). This function automatically set as (np.nan, np.nan) the coordinates of BBs outside image space, too dark or too close to another BBs.

Parameters

- **img** (*numpy.array*) – Array containing the loaded .raw or .hnc file
- **ref_2d** (*numpy.array*) – Nx2 array containing the coordinates for BBs projected on img

- **search_area** (*int*) – Size of the region in which to search for centroids. Actual dimension of the area is a square with dimension (2* search_area,2*search_area)
- **image_size** (*list*) – Dimension of img
- **mode** (*str*) – Centroid search modality. It can be "circle" or "ellipse". Ellipse is slower but provide better results in general.
- **debug_level** (*int, optional*) – Level for debug messages, 0 means no debug messages, 1 light debug and 2 hard debug. Defaults to 0.

Returns Nx2 array containing coordinates for every centroids found (x,y)

Return type numpy.array

geometric_calibration.slideshow module

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/mrossi93/geometric_calibration/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

Geometric Calibration could always use more documentation, whether as part of the official Geometric Calibration docs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/mrossi93/geometric_calibration/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *geometric_calibration* for local development.

1. Fork the *geometric_calibration* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/geometric_calibration.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv geometric_calibration
$ cd geometric_calibration/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 geometric_calibration tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/mrossi93/geometric_calibration/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_geometric_calibration
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 6

Credits

6.1 Development Lead

- Matteo Rossi <rossimatteo1993@gmail.com>

6.2 Contributors

None yet. Why not be the first?

CHAPTER 7

History

7.1 0.2.0 (2020-07-22)

- Added support for planar calibration
- Added –config parameter to easily setup calibration with an .ini file
- Added support to read automatically .raw and .hnc projection files

7.2 0.1.2 (2020-04-01)

- Fixed bugs in release 0.1.1

7.3 0.1.1 (2020-04-01)

- First release on PyPI.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

Symbols

C

D

E

F

G

N

O

P

R

S

V