Geometric Calibration Documentation

Release 0.2.0

Matteo Rossi

Feb 17, 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

Main module.

geometric_calibration.geometric_calibration.**calibrate_2d**(*projection_dir*, *bbs_3d*, *sad*, *sid*)

Main 2D Calibration routines.

Parameters

- **projection_dir** (*str*) – path to directory containing .raw files
- **bbs_3d** (*numpy.array*) – array containing 3D coordinates of BBs
- **sad** (*float*) – nominal source to isocenter (A) distance
- **sid** (*float*) – nominal source to image distance

Returns dictionary with calibration results

Return type dict

geometric_calibration.geometric_calibration.**calibrate_cbct**(*projection_dir*, *bbs_3d*, *sad*, *sid*)

Main CBCT Calibration routines.

Parameters

- **projection_dir** (*str*) – path to directory containing .raw files
- **bbs_3d** (*numpy.array*) – array containing 3D coordinates of BBs
- **sad** (*float*) – nominal source to isocenter (A) distance
- **sid** (*float*) – nominal source to image distance

Returns dictionary with calibration results

Return type dict

geometric_calibration.geometric_calibration.**calibrate_projection**(*projection_file, bbs_3d, sad, sid, angle, angle_offset=0, img_dim=[1024, 768], pixel_size=[0.388, 0.388], search_area=7, resolution_factor=1, image_center=None, drag_and_drop=True*)

Calibration of a single projection.

Parameters

- **projection_file** (*str*) – path to file
- **bbs_3d** (*numpy.array*) – 3D coordinates of phantom's reference points
- **sad** (*float*) – nominal source to isocenter (A) distance
- **sid** (*float*) – nominal source to image distance
- **angle** (*float*) – gantry angle for current projection
- **angle_offset** (*int, optional*) – angle offset for panel, defaults to 0
- **img_dim** (*list, optional*) – image dimensions in pixels, defaults to [1024, 768]
- **pixel_size** (*list, optional*) – pixel dimensions in mm, defaults to [0.388, 0.388]
- **search_area** (*int, optional*) – dimension of reference point's searching area, defaults to 7
- **resolution_factor** (*int, optional :param image_center: [description], defaults to None*) – resolution factor, when mode is "cbct" this parameter equals to 1, in 2D mode is 2 (because resolution is doubled), defaults to 1
- **image_center** (*list, optional*) – center of image, defaults to None
- **drag_and_drop** (*bool, optional*) – whether or not perform Drag&Drop correction routines, typically set to True for first projection. Defaults to True

Raises **Exception** – if less than 5 BBs centroids are recognized, optimizer automatically fails since calibration can't be consider reliable

Returns dictionary with calibration results for current projection

Return type dict

geometric_calibration.geometric_calibration.**calibration_cost_function**(*param*, *bbs_3d*, *pixel_size*, *bbs_2d*, *isocenter*)

Cost Function for calibration optimizers.

Parameters

- **param** (*list*) – parameters to be optimized
- **bbs_3d** (*numpy.array*) – 3D coordinates of reference BBs
- **pixel_size** (*list*) – pixel dimensions in mm
- **bbs_2d** (*numpy.array*) – 2D coordinates of BBs projected on the current image
- **isocenter** (*numpy.array*) – coordinates of isocenter

Returns cost function value to be minimized

Return type float

geometric_calibration.geometric_calibration.**plot_calibration_results**(*calib_results*)

Plot source/panel position after calibration.

Parameters **calib_results** (*dict*) – dictionary containing results of a calibration

geometric_calibration.geometric_calibration.**save_lut**(*path*, *calib_results*, *mode*)

Save LUT file for a calibration.

Parameters

- **path** (*str*) – path to .raw file directory, where LUT will be saved
- **calib_results** (*string*) – dictionary containing results for a calibration
- **calib_results** – acquisition modality for calibration

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.utils module

Utilities module.

class geometric_calibration.utils.**DraggablePoints**(*artists*, *tolerance=10*)
Bases: object

Draggable points on matplotlibe figure.

Returns: DraggablePoints – DraggablePoints object

get_coord()
Obtain current coordinates of points.

Returns An array nx2 containing coordinates for every point [x,y]

Return type numpy.array

on_close(*event*)
Event Handler for closure of figure.

Arguments: event – Event that triggers the method

on_key_pressed(*event*)
Event Handler for "enter" key pression.

Arguments: event – Event that triggers the method

on_motion(*event*)
Event Handler for mouse movement during dragging of points.

Arguments: event – Event that triggers the method

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

Arguments: event – Event that triggers the method

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

Arguments: event – Event that triggers the method

geometric_calibration.utils.**adjust_image**(*img*, *new_grayscale_range*)

Translate image data to the appropriate lower bound of the default data range. This translation is needed to show properly .raw images.

Parameters

- **img** (*numpy.array*) – Array containing the loaded .raw image
- **new_grayscale_range** (*list*) – Grayscale range to apply to image

Returns Image corrected with new grayscale range

Return type numpy.array

geometric_calibration.utils.**angle2rotm**(*rot_x*, *rot_y*, *rot_z*)

Generate a rototranslator (only rotation) starting from Euler angles

NB: Convention is 'XYZ'

Parameters

- **rot_x** (*int or float*) – Rotation along x
- **rot_y** (*int or float*) – Rotation along y
- **rot_z** (*int or float*) – Rotation along z

Returns 4x4 Rototranslation matrix in homogeneous form

Return type numpy.array

geometric_calibration.utils.**create_camera_matrix**(*panel_orientation*, *sid*, *sad*, *pixel_size*, *isocenter*)

Generate projection matrix starting from extrinsic and intrinsic parameters.

Parameters

- **panel_orientation** (*numpy.array*) – Array nx3 containing rotations of the image's plane [rot_x, rot_y, rot_z]
- **sid** (*float*) – SID distance
- **sad** (*float*) – SAD distance
- **pixel_size** (*list*) – Pixel Dimensions in mm
- **isocenter** (*numpy.array*) – Coordinates of isocenter

Returns 3x4 Camera Matrix

Return type numpy.array

geometric_calibration.utils.**deg2rad**(*angle_deg*)

Convert angles from degrees to radians.

Parameters **angle_deg** (*int or float*) – Angle to convert

Returns Angle converted in radians

Return type float

geometric_calibration.utils.**drag_and_drop_bbs**(*projection_path*, *bbs_projected*, *grayscale_range*)

Drag&Drop Routines for bbs position's correction.

Parameters

- **projection_path** (*str*) – Path to the projection .raw file

- **bbs_projected** (*numpy.array*) – Array nx2 with bbs yet projected
- **grayscale_range** (*list*) – Grayscale range for current projection

Returns Array nx2 containing the updated coordinates for bbs

Return type numpy.array

geometric_calibration.utils.**get_grayscale_range**(*img*)

New grayscale range for .raw 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 im is loaded as uint16)

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

Returns Grayscale range for current projection

Return type list

geometric_calibration.utils.**project_camera_matrix**(*r3d*, *image_center*, *camera_matrix*, *resolution_scale=1*)

Project 3D data starting from camera matrix based on intrinsic and extrinsic parameters

Parameters

- **r3d** (*numpy.array*) – Array nx3 containing 3d coordinates of points [x,y,z]
- **image_center** (*list*) – Center of the image
- **camera_matrix** (*numpy.array*) – Projection matrix obtained combining extrinsic and intrinsic parameters
- **resolution_scale** – resolution factor, when mode is "cbct" this

parameter equals to 1, in 2D mode is 2 (because resolution is doubled), defaults to 1

Returns Array nx2 containing 2D coordinates of points projected on image plane [x,y]

Return type numpy.array

geometric_calibration.utils.**search_bbs_centroids**(*img*, *ref_2d*, *search_area*, *dim_img*, *grayscale_range*)

Search bbs based on projection.

Starting from the updated coordinates, define a search area around them and identify the bbs as black pixels inside these areas (brandis are used as probes). Search for the bbs in the image (basically very low intensity surrounding by higher intensity pixel. Centroids coordinates are the mean pixels that have an intensity that is lower than the lowest nominal intensity plus a tollerance.

Parameters

- **img** (*numpy.array*) – Array containing the loaded .raw file
- **ref_2d** (*numpy.array*) – Array nx2 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)
- **dim_img** (*list*) – Dimension of img
- **grayscale_range** (*list*) – Grayscale range for current projection

Raises **Exception** – if the function does not find any centroid, an exception is thrown

Returns Array nx2 containing coordinates for every centroids found [x,y]

Return type numpy.array

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