# scikit-surgerycalibration Documentation

Stephen Thompson

Feb 25, 2023

# Contents

CHAPTER 1

# Features

- Pivot Calibration for pivot calibration.
- Calibration of mono or stereo tracked video data, calculating camera intrinsics and handeye transformation.

Source code is avaialble on GitHub.

## 1.1 scikit-surgerycalibration

Author(s): Stephen Thompson; Contributor(s): Matt Clarkson, Thomas Dowrick and Miguel Xochicale

scikit-surgerycalibration is part of the SciKit-Surgery software project, developed at the Wellcome EPSRC Centre for Interventional and Surgical Sciences, part of University College London (UCL).

scikit-surgerycalibration is tested on Python 3.7.

scikit-surgerycalibration contains algorithms to perform calibrations useful during surgery, for example pointer calibration, ultrasound calibration, and camera calibration.

Please explore the project structure, and request or implement your desired functionality.

### 1.1.1 Features

- Pivot Calibration for pivot calibration.
- Calibration of mono or stereo tracked video data, calculating camera intrinsics and handeye transformation.

### 1.1.2 Cloning

You can clone the repository using the following command:

```
git clone https://github.com/SciKit-Surgery/scikit-surgerycalibration
git clone git@github.com:SciKit-Surgery/scikit-surgerycalibration.git # Alternatively,
↪ use password-protected SSH key.
```

### 1.1.3 Developing

We recommend using anaconda or miniconda to create a python 3.7 environment, then using tox to install all dependencies inside a dedicated venv. We then use github actions to run a matrix of builds for Windows, Linux and Mac and various python versions.

All library dependencies are specified via `requirements-dev.txt` which refers to `requirements.txt`.

So, assuming either anaconda or miniconda is installed, and your current working directory is the root directory of this project:

```
conda create --name scikit-surgery python=3.7
conda activate scikit-surgery
pip install tox
tox
```

As the tox command runs, it will install all dependencies in a sub-directory `.tox/py37` (Linux/Mac) or `.tox\py37` (Windows). tox will also run pytest and linting for you.

To run commands inside the same environment as tox, you should:

```
source .tox/py37/bin/activate
```

on Linux/Mac, or if you are Windows user:

```
.tox\py37\Scripts\activate
```

Then you can run pytest, linting, or directly run python scripts, and know that the environment was created correctly by tox.

#### Generating documentation

The simplest way is again using tox.

```
tox -e docs
```

then open `docs/build/html/index.html` in your browser.

#### Running tests

Pytest is used for running unit tests:

```
python -m pytest
pytest -v -s tests/algorithms/test_triangulate.py #example for individual tests
```

#### Linting

This code conforms to the PEP8 standard. Pylint can be used to analyse the code:

```
pylint --rcfile=tests/pylintrc sksurgerycalibration
```

### 1.1.4 Installing

You can pip install directly from the repository as follows:

```
pip install git+https://github.com/SciKit-Surgery/scikit-surgerycalibration
```

### 1.1.5 Contributing

Please see the contributing guidelines.

### 1.1.6 Useful links

- Source code repository
- Documentation

### 1.1.7 Licensing and copyright

Copyright 2020 University College London. scikit-surgerycalibration is released under the BSD-3 license. Please see the license file for details.

### 1.1.8 Acknowledgements

Supported by Wellcome and EPSRC.

## 1.2 Pivot Calibration

Functions for pivot calibration.

sksurgerycalibration.algorithms.pivot.**pivot_calibration**(*tracking_matrices*, *configuration=None*)

Performs pivot calibration on an array of tracking matrices

**Parameters**

- **tracking_matrices** – an Nx4x4 array of tracking matrices
- **configuration** – an optional configuration dictionary, if not the algorithm defaults to Algebraic One Step. Other options include ransac, and sphere_fitting

**Returns** tuple containing; 'pointer_offset' The coordinate of the pointer tip relative to the tracking centre 'pivot_point' The location of the pivot point in world coordinates 'residual_error' The RMS pointer tip error, errors in each direction are treated as independent variables, so for a calibration with n matrices, RMS error is calculated using nx3 measurements.

**Raises** TypeError, ValueError

sksurgerycalibration.algorithms.pivot.**pivot_calibration_aos**(*tracking_matrices*)

Performs Pivot Calibration, using Algebraic One Step method, and returns Residual Error.

See Yaniv 2015.

**Parameters** **tracking_matrices** – N x 4 x 4 ndarray, of tracking matrices.

**Returns** pointer offset, pivot point and RMS Error about centroid of pivot.

**Raises** ValueError if rank less than 6

sksurgerycalibration.algorithms.pivot.**pivot_calibration_sphere_fit**(*tracking_matrices*, *init_parameters=None*)

Performs Pivot Calibration, using sphere fitting, based on

See Yaniv 2015.

**Parameters**

- **tracking_matrices** – N x 4 x 4 ndarray, of tracking matrices.
- **init_parameters** – 1X4 array of initial parameter for finding the pivot point in world coords and pivot radius. Default is to set to the mean x,y,z values and radius = 0.

**Returns** pointer offset, pivot point and RMS Error about centroid of pivot.

sksurgerycalibration.algorithms.pivot.**pivot_calibration_with_ransac**(*tracking_matrices*, *number_iterations*, *error_threshold*, *consensus_threshold*, *early_exit=False*)

Written as an exercise for implementing RANSAC.

**Parameters**

- **tracking_matrices** – N x 4 x 4 ndarray, of tracking matrices.
- **number_iterations** – the number of iterations to attempt.
- **error_threshold** – distance in millimetres from pointer position

- **concensus_threshold** – the minimum percentage of inliers to finish
- **early_exit** – If True, returns model as soon as thresholds are met

**Returns** pointer offset, pivot point and RMS Error about centroid of pivot.

**Raises** TypeError, ValueError

Functions used by calibration the calibration routines

sksurgerycalibration.algorithms.sphere_fitting.**fit_sphere_least_squares**(*coordinates*, *initial_parameters*, *bounds=((-inf*, *-inf*, *-inf*, *-inf)*, *(inf*, *inf*, *inf*, *inf)))*

Uses scipy's least squares optimisor to fit a sphere to a set of 3D Points

**Parameters**

- **coordinates** – (x,y,z) n x 3 array of point coordinates
- **parameters** (*initial*) – 1 x 4 array containing four initial values (centre, and radius)

**Returns** x: an array containing the four fitted parameters

**Returns** ier: int An integer flag. If it is equal to 1, 2, 3 or 4, the solution was found.

## 1.3 Video Calibration

### 1.3.1 Mono

Class to do stateful video calibration of a mono camera.

**class** sksurgerycalibration.video.video_calibration_driver_mono.**MonoVideoCalibrationDriver**(*p*
*s*
*i*
*c*
*r*
*i*
*r*
*i*

Bases: sksurgerycalibration.video.video_calibration_driver_base.BaseVideoCalibrationDriver

Class to do stateful video calibration of a mono camera.

**calibrate**(*flags=0*)

Do the video calibration, returning RMS re-projection error.

**Parameters** **flags** – OpenCV calibration flags, eg. cv2.CALIB_FIX_ASPECT_RATIO

**Returns** RMS projection

**grab_data**(*image*, *device_tracking=None*, *calibration_object_tracking=None*)
Extracts points, by passing it to the PointDetector.

This will throw various exceptions if the input data is invalid, but will return empty arrays if no points were detected. So, no points is not an error. Its an expected condition.

**Parameters**

- **image** – RGB image.
- **device_tracking** – transformation for the tracked device
- **calibration_object_tracking** – transformation of tracked

calibration object :return: The number of points grabbed.

**handeye_calibration**(*override_pattern2marker=None*, *use_opencv: bool = True*, *do_bundle_adjust: bool = False*)
Do handeye calibration, returning RMS re-projection error.

Note: This handeye_calibration on this class assumes you are tracking both the calibration pattern (e.g. chessboard) and the device (e.g. laparoscope). So, the calibration routines calibrate for hand2eye and pattern2marker. If you want something more customised, work with video_calibration_hand_eye.py.

**Parameters** **override_pattern2marker** – If provided a 4x4 pattern2marker

that is taken as constant. :param use_opencv: If True we use OpenCV based methods, if false, Guofang Xiao's method. :param do_bundle_adjust: If True we do an additional bundle adjustment at the end.

**Returns** RMS reprojection error

**Return type** float

**iterative_calibration**(*number_of_iterations: int*, *reference_ids*, *reference_image_points*, *reference_image_size*, *flags: int = 0*)
Does iterative calibration, like Datta 2009, returning RMS re-projection error. :return: RMS projection

### 1.3.2 Stereo

Class to do stateful video calibration of a stereo camera.

**class** sksurgerycalibration.video.video_calibration_driver_stereo.**StereoVideoCalibrationDriv**

Bases: sksurgerycalibration.video.video_calibration_driver_base.BaseVideoCalibrationDriver

Class to do stateful video calibration of a stereo camera.

**calibrate**(*flags=1*, *override_left_intrinsics=None*, *override_left_distortion=None*, *override_right_intrinsics=None*, *override_right_distortion=None*, *override_l2r_rmat=None*, *override_l2r_tvec=None*)

Do the stereo video calibration, returning reprojection and reconstruction error.

This returns RMS projection error, which is a common metric, but also, the reconstruction / triangulation error.

**Parameters**

- **flags** – OpenCV flags, eg. cv2.CALIB_FIX_INTRINSIC
- **override_left_intrinsics** –
- **override_left_distortion** –
- **override_right_intrinsics** –
- **override_right_distortion** –
- **override_l2r_rmat** –
- **override_l2r_tvec** –

**Returns** projection, reconstruction error.

**Return type** float, float

**grab_data**(*left_image*, *right_image*, *device_tracking=None*, *calibration_object_tracking=None*)

Extracts points, by passing it to the PointDetector.

This will throw various exceptions if the input data is invalid, but will return empty arrays if no points were detected. So, no points is not an error. Its an expected condition.

**Parameters**

- **left_image** – BGR image.
- **right_image** – BGR image.
- **device_tracking** – transformation for the tracked device
- **calibration_object_tracking** – transformation of tracked

calibration object :return: The number of points grabbed.

**handeye_calibration**(*override_pattern2marker=None*, *use_opencv: bool = True*, *do_bundle_adjust: bool = False*)

Do handeye calibration, returning reprojection and reconstruction error.

Note: This handeye_calibration on this class assumes you are tracking both the calibration pattern (e.g. chessboard) and the device (e.g. laparoscope). So, the calibration routines calibrate for hand2eye and pattern2marker. If you want something more customised, work with video_calibration_hand_eye.py.

**Parameters** **override_pattern2marker** – If provided a 4x4 pattern2marker

that is taken as constant. :param use_opencv: If True we use OpenCV based methods, if false, Guofang Xiao's method. :param do_bundle_adjust: If True we do an additional bundle adjustment at the end.

**Returns** reprojection, reconstruction error, camera parameters

**Return type** float, float, object

**iterative_calibration**(*number_of_iterations: int*, *reference_ids*, *reference_image_points*, *reference_image_size*, *flags: int = 1*)

Does iterative calibration, like Datta 2009, returning reprojection and reconstruction error.

**Returns** projection, reconstruction error.

**Return type** float, float

### 1.3.3 Video Calibration Data

Containers for video calibration data.

**class** sksurgerycalibration.video.video_calibration_data.**BaseVideoCalibrationData**
Bases: object

Constructor, no member variables, so just a pure virtual interface.

Not really necessary if you rely on duck-typing, but at least it shows the intention of what derived classes should implement, and means we can use this base class to type check against.

**get_number_of_views**()
Returns the number of views of data.

**load_data**(*dir_name: str*, *file_prefix: str*)
Loads all contained data from disk.

**pop**()
Remove the last view of data.

**reinit**()
Used to clear, re-initialise all member variables.

**save_data**(*dir_name: str*, *file_prefix: str*)
Writes all contained data to disk.

**class** sksurgerycalibration.video.video_calibration_data.**MonoVideoData**
Bases: sksurgerycalibration.video.video_calibration_data.BaseVideoCalibrationData

Stores data extracted from each video view of a mono calibration.

**get_number_of_views**()
Returns the number of views.

**load_data**(*dir_name: str*, *file_prefix: str*)
Loads the calibration data.

**Parameters**

- **dir_name** – directory to load from
- **file_prefix** – prefix for all files

**pop**()
Removes the last (most recent) view of data.

**push**(*image*, *ids*, *object_points*, *image_points*)
Stores another view of data. Copies data.

**reinit**()
Deletes all data.

**save_annotated_images**(*dir_name: str*, *file_prefix: str*)
Saves video images, annotated with the ID of each 2D point detected.

**save_data**(*dir_name: str*, *file_prefix: str*)
Saves the calibration data to lots of different files.

**Parameters**

- **dir_name** – directory to save to
- **file_prefix** – prefix for all files

**class** sksurgerycalibration.video.video_calibration_data.**StereoVideoData**
Bases: sksurgerycalibration.video.video_calibration_data.BaseVideoCalibrationData

Stores data extracted from each view of a stereo calibration.

**get_number_of_views**()
Returns the number of views.

**load_data**(*dir_name: str*, *file_prefix: str*)
Loads the calibration data.

**Parameters**

- **dir_name** – directory to load from
- **file_prefix** – prefix for all files

**pop**()
Removes the last (most recent) view of data.

**push**(*left_image*, *left_ids*, *left_object_points*, *left_image_points*, *right_image*, *right_ids*, *right_object_points*, *right_image_points*)
Stores another view of data. Copies data.

**reinit**()
Deletes all data.

**save_annotated_images**(*dir_name: str*, *file_prefix: str*)
Saves video images, annotated with the ID of each 2D point detected.

**save_data**(*dir_name: str*, *file_prefix: str*)
Saves the calibration data to lots of different files.

**Parameters**

- **dir_name** – directory to save to
- **file_prefix** – prefix for all files

**class** sksurgerycalibration.video.video_calibration_data.**TrackingData**
Bases: sksurgerycalibration.video.video_calibration_data.BaseVideoCalibrationData

Class for storing tracking data.

**get_number_of_views**()
Returns the number of views of data. :return: int

**load_data**(*dir_name: str*, *file_prefix: str*)
Loads tracking data from files.

**Parameters**

- **dir_name** – directory to load from
- **file_prefix** – prefix for all files

**pop**()
Removes the last (most recent) view of data.

**push**(*device_tracking*, *calibration_tracking*)
Stores a pair of tracking data.

**Parameters**

- **device_tracking** – transformation for the thing you're tracking
- **calibration_tracking** – transformation for tracked calibration obj

**reinit**()
Deletes all data.

**save_data**(*dir_name: str*, *file_prefix: str*)
Saves the tracking data to lots of different files.

**Parameters**

- **dir_name** – directory to save to
- **file_prefix** – prefix for all files

### 1.3.4 Video Calibration Metrics

Video calibration metrics, used in cost functions for optimisation, and as measures of error generally.

sksurgerycalibration.video.video_calibration_metrics.**compute_mono_2d_err**(*object_points*, *image_points*, *rvecs*, *tvecs*, *camera_matrix*, *distortion*, *return_residuals=False*)
Function to compute mono reprojection (SSE) error, or residuals over multiple views of a mono camera.

**Parameters**

- **object_points** – Vector of Vector of 1x3 of type float32
- **image_points** – Vector of Vector of 1x2 of type float32
- **rvecs** – Vector of [3x1] ndarray, Rodrigues rotations for each camera
- **tvecs** – Vector of [3x1] ndarray, translations for each camera
- **camera_matrix** – [3x3] ndarray
- **distortion** – [1x5] ndarray
- **return_residuals** – If True returns a big array of residuals for LM.

**Returns** SSE re-reprojection error, number_samples OR residuals

sksurgerycalibration.video.video_calibration_metrics.**compute_mono_2d_err_handeye**(*model_points: List[T], image_points: List[T], camera_matrix: numpy.ndarra camera_distortion numpy.ndarra hand_tracking List[T], model_trackin List[T], handeye_matrix: numpy.ndarra pattern2marker_ numpy.ndarra*

Function to compute mono reprojection error (SSE), mapping from the calibration pattern coordinate system to the camera coordinate system, via tracking matrices and hand-eye calibration.

**Parameters**

- **model_points** (*List*) – Vector of Vector of 1x3 float32
- **image_points** (*List*) – Vector of Vector of 1x2 float32
- **camera_matrix** (*np.ndarray*) – Camera intrinsic matrix
- **camera_distortion** (*np.ndarray*) – Camera distortion coefficients
- **hand_tracking_array** –

Vector of 4x4 tracking matrices for camera (hand) :type hand_tracking_array: List :param model_tracking_array: Vector of 4x4 tracking matrices for calibration model :type model_tracking_array: List :param handeye_matrix: Handeye matrix :type handeye_matrix: np.ndarray :param pattern2marker_matrix: Pattern to marker matrix :type pattern2marker_matrix: np.ndarray :return: SSE reprojection error, number of samples :rtype: float, float

sksurgerycalibration.video.video_calibration_metrics.**compute_mono_3d_err**(*ids*, *object_points*, *image_points*, *rvecs*, *tvecs*, *camera_matrix*, *distortion*)

Function to compute mono reconstruction error (SSE) over multiple views.

Here, to triangulate, we take the i^th camera as left camera, and the i+1^th camera as the right camera, compute l2r, and triangulate.

Note: This may fail if the difference between two successive views is too large, and there are not enough common points.

**Parameters**

- **ids** – Vector of ndarray of integer point ids
- **object_points** – Vector of Vector of 1x3 of type float32
- **image_points** – Vector of Vector of 1x2 of type float32
- **rvecs** – Vector of [3x1] ndarray, Rodrigues rotations for each camera
- **tvecs** – Vector of [3x1] ndarray, translations for each camera
- **camera_matrix** – [3x3] ndarray
- **distortion** – [1x5] ndarray

**Returns** SSE re-reprojection error, number_samples

sksurgerycalibration.video.video_calibration_metrics.**compute_mono_3d_err_handeye**(*ids: List[T], model_points: List[T], image_points: List[T], camera_matrix: numpy.ndarra camera_distortion numpy.ndarra hand_tracking List[T], model_trackin List[T], hand-eye_matrix: numpy.ndarra pattern2marker_ numpy.ndarra*

Function to compute mono reconstruction error (SSE). Calculates new rvec/tvec values for pattern_to_camera based on handeye calibration and then calls compute_mono_3d_err().

**Parameters**

- **ids** (*List*) – Vector of ndarray of integer point ids
- **model_points** (*List*) – Vector of Vector of 1x3 float32
- **image_points** (*List*) – Vector of Vector of 1x2 float32
- **camera_matrix** (*np.ndarray*) – Camera intrinsic matrix
- **camera_distortion** (*np.ndarray*) – Camera distortion coefficients
- **hand_tracking_array** –

Vector of 4x4 tracking matrices for camera (hand) :type hand_tracking_array: List :param model_tracking_array: Vector of 4x4 tracking matrices for calibration model :type model_tracking_array: List

:param handeye_matrix: Handeye matrix :type handeye_matrix: np.ndarray :param pattern2marker_matrix: Pattern to marker matrix :type pattern2marker_matrix: np.ndarray :return: SSE reprojection error, number of samples :rtype: float, float

sksurgerycalibration.video.video_calibration_metrics.**compute_stereo_2d_err**(*l2r_rmat*, *l2r_tvec*, *left_object_points*, *left_image_points*, *left_camera_matrix*, *left_distortion*, *right_object_points*, *right_image_points*, *right_camera_matrix*, *right_distortion*, *left_rvecs*, *left_tvecs*, *return_residuals=False*)

Function to compute stereo re-projection error (SSE), or residuals, over multiple views.

**Parameters**

- **l2r_rmat** – [3x3] ndarray, rotation for l2r transform
- **l2r_tvec** – [3x1] ndarray, translation for l2r transform
- **left_object_points** – Vector of Vector of 1x3 of type float32
- **left_image_points** – Vector of Vector of 1x2 of type float32
- **left_camera_matrix** – [3x3] ndarray
- **left_distortion** – [1x5] ndarray
- **right_object_points** – Vector of Vector of 1x3 of type float32
- **right_image_points** – Vector of Vector of 1x2 of type float32
- **right_camera_matrix** – [3x3] ndarray
- **right_distortion** – [1x5] ndarray
- **left_rvecs** – Vector of [3x1] ndarray, Rodrigues rotations, left camera
- **left_tvecs** – Vector of [3x1] ndarray, translations, left camera
- **return_residuals** – if True returns vector of residuals for LM,

otherwise, returns SSE. :return: SSE, number_samples OR residuals

sksurgerycalibration.video.video_calibration_metrics.**compute_stereo_2d_err_handeye**(*common_o*
*List[T],*
*left_image_*
*List[T],*
*left_camer*
*numpy.nda*
*left_distort*
*numpy.nda*
*right_imag*
*List[T],*
*right_came*
*numpy.nda*
*right_disto*
*numpy.nda*
*hand_track*
*List[T],*
*model_trac*
*List[T],*
*left_handey*
*numpy.nda*
*left_pattern*
*numpy.nda*
*right_hand*
*numpy.nda*
*right_patte*
*numpy.nda*

Function to compute stereo reprojection error (SSE), taking into account handeye calibration.

**Parameters**

- **common_object_points** (*List*) – Vector of Vector of 1x3 float32
- **left_image_points** (*List*) – Vector of Vector of 1x2 float32
- **left_camera_matrix** (*np.ndarray*) – Left camera matrix
- **left_distortion** (*np.ndarray*) – Left camera distortion coefficients
- **right_image_points** (*List*) – Vector of Vector of 1x2 float32
- **right_camera_matrix** (*np.ndarray*) – Right camera matrix
- **right_distortion** (*np.ndarray*) – Right camera distortion coefficients
- **hand_tracking_array** –

Vector of 4x4 tracking matrices for camera (hand) :type hand_tracking_array: List :param model_tracking_array: Vector of 4x4 tracking matrices for calibration model :type model_tracking_array: List :param left_handeye_matrix: Left handeye transform matrix :type left_handeye_matrix: np.ndarray :param left_pattern2marker_matrix: Left pattern to marker transform matrix :type left_pattern2marker_matrix: np.ndarray :param right_handeye_matrix: Right handeye transform matrix :type right_handeye_matrix: np.ndarray :param right_pattern2marker_matrix: Right pattern to marker transform matrix :type right_pattern2marker_matrix: np.ndarray :return: SSE reprojection error, number of samples :rtype: float, float

sksurgerycalibration.video.video_calibration_metrics.**compute_stereo_3d_err_handeye**(*l2r_rmat: numpy.nda l2r_tvec: numpy.nda com- mon_objec List[T], com- mon_left_i List[T], left_camer numpy.nda left_distort numpy.nda com- mon_right_ List[T], right_came numpy.nda right_disto numpy.nda hand_track List[T], model_trac List[T], left_handey numpy.nda left_patter numpy.nda*

Function to compute stereo reconstruction error (SSE), taking into account handeye calibration.

**Parameters**

- **l2r_rmat** (*np.ndarray*) – Rotation for l2r transform
- **l2r_tvec** (*np.ndarray*) – Translation for l2r transform
- **common_object_points** (*List*) – Vector of Vector of 1x3 float32
- **common_left_image_points** (*List*) – Vector of Vector of 1x2 float32
- **left_camera_matrix** (*np.ndarray*) – Left camera matrix
- **left_distortion** (*np.ndarray*) – Left camera distortion coefficients
- **common_right_image_points** (*List*) – Vector of Vector of 1x2 float32
- **right_camera_matrix** (*np.ndarray*) – Right camera matrix
- **right_distortion** (*np.ndarray*) – Right camera distortion coefficients
- **hand_tracking_array** –

Vector of 4x4 tracking matrices for camera (hand) :type hand_tracking_array: List :param model_tracking_array: Vector of 4x4 tracking matrices for calibration model :type model_tracking_array: List :param left_handeye_matrix: Left handeye transform matrix :type left_handeye_matrix: np.ndarray :param left_pattern2marker_matrix: Left pattern to marker transform matrix :type left_pattern2marker_matrix: np.ndarray :return: SSE reconstruction error, number of samples :rtype: float, float

sksurgerycalibration.video.video_calibration_metrics.**compute_stereo_3d_error**(*l2r_rmat*, *l2r_tvec*, *common_object_points*, *common_left_image_po*, *left_camera_matrix*, *left_distortion*, *common_right_image_p*, *right_camera_matri*, *right_distortion*, *left_rvecs*, *left_tvecs*, *return_residuals=Fals*)

Function to compute stereo reconstruction error (SSE), or residuals over multiple views.

**Parameters**

- **l2r_rmat** – [3x3] ndarray, rotation for l2r transform
- **l2r_tvec** – [3x1] ndarray, translation for l2r transform
- **common_object_points** – Vector of Vector of 1x3 of type float32
- **common_left_image_points** – Vector of Vector of 1x2 of type float32
- **left_camera_matrix** – [3x3] ndarray
- **left_distortion** – [1x5] ndarray
- **common_right_image_points** – Vector of Vector of 1x2 of type float32
- **right_camera_matrix** – [3x3] ndarray
- **right_distortion** – [1x5] ndarray
- **left_rvecs** – Vector of [3x1] ndarray, Rodrigues rotations, left camera
- **left_tvecs** – Vector of [3x1] ndarray, translations, left camera
- **return_residuals** – if True returns vector of residuals for LM,

otherwise, returns SSE. :return: SSE re-reprojection error, number_samples

### 1.3.5 Video Calibration Parameters

Containers for video calibration parameters.

**class** sksurgerycalibration.video.video_calibration_params.**BaseCalibrationParams**

Bases: object

Constructor, no member variables, so just a pure virtual interface.

Not really necessary if you rely on duck-typing, but at least it shows the intention of what derived classes should implement, and means we can use this base class to type check against.

**load_data**(*dir_name: str*, *file_prefix: str*)

Loads all contained data from disk.

**reinit**()

Used to clear, re-initialise all member variables.

**save_data**(*dir_name: str*, *file_prefix: str*)
Writes all contained data to disk.

**class** sksurgerycalibration.video.video_calibration_params.**MonoCalibrationParams**
Bases: sksurgerycalibration.video.video_calibration_params.BaseCalibrationParams

Holds a set of intrinsic and extrinsic camera parameters for 1 camera.

**load_data**(*dir_name: str*, *file_prefix: str*, *halt_on_ioerror=True*)
Loads calibration parameters from a directory.

**Parameters**

- **dir_name** – directory to load from
- **file_prefix** – prefix for all files
- **halt_on_ioerror** – if false, and handeye or pattern2marker are not found they will be left as None

**reinit**()
Resets data, to identity/empty arrays etc.

**save_data**(*dir_name: str*, *file_prefix: str*)
Saves calibration parameters to a directory.

**Parameters**

- **dir_name** – directory to save to
- **file_prefix** – prefix for all files

**set_data**(*camera_matrix*, *dist_coeffs*, *rvecs*, *tvecs*)
Stores the provided parameters, by taking a copy.

**set_handeye**(*handeye_matrix*, *pattern2marker_matrix*)
Stores the provided parameters, by taking a copy.

**class** sksurgerycalibration.video.video_calibration_params.**StereoCalibrationParams**
Bases: sksurgerycalibration.video.video_calibration_params.BaseCalibrationParams

Holds a pair of MonoCalibrationParams, and the left-to-right transform.

**get_l2r_as_4x4**()
Extracts the left-to-right transform as 4x4 matrix.

**load_data**(*dir_name: str*, *file_prefix: str*)
Loads calibration parameters from a directory.

**Parameters**

- **dir_name** – directory to load from
- **file_prefix** – prefix for all files

**reinit**()
Resets data, to identity/empty arrays etc.

**save_data**(*dir_name: str*, *file_prefix: str*)
Saves calibration parameters to a directory.

**Parameters**

- **dir_name** – directory to save to

- **file_prefix** – prefix for all files

**set_data**(*left_cam_matrix*, *left_dist_coeffs*, *left_rvecs*, *left_tvecs*, *right_cam_matrix*, *right_dist_coeffs*, *right_rvecs*, *right_tvecs*, *l2r_rmat*, *l2r_tvec*, *essential*, *fundamental*)
Stores the provided parameters, by taking a copy.

**set_handeye**(*left_handeye_matrix*, *left_pattern2marker_matrix*, *right_handeye_matrix*, *right_pattern2marker_matrix*)
Call the left/right set_handeye methods.

### 1.3.6 Handeye Calibration Functions

Various routines for Hand-Eye calibration.

sksurgerycalibration.video.video_calibration_hand_eye.**calibrate_hand_eye_and_grid_to_world**

Hand-eye calibration using standard OpenCV methods. This method assumes you have a stationary untracked calibration pattern, and a tracked device (e.g. laparoscope)

**Parameters** **camera_rvecs** – list of rvecs that we get from OpenCV camera

extrinsics, pattern_to_camera. :param camera_tvecs: list of tvecs that we get from OpenCV camera extrinsics, pattern_to_camera. :param device_tracking_matrices: list of tracking matrices for the tracked device, e.g. laparoscope, marker_to_tracker. :param method: Choice of OpenCV RobotWorldHandEye method. :return hand-eye, grid-to-world transforms as 4x4 matrices

sksurgerycalibration.video.video_calibration_hand_eye.**calibrate_hand_eye_and_pattern_to_ma**

Hand-eye calibration using standard OpenCV methods. This method assumes you are tracking both the device that needs hand-eye calibration, and the calibration pattern.

**Parameters** **camera_rvecs** – list of rvecs that we get from OpenCV camera

extrinsics, pattern_to_camera. :param camera_tvecs: list of tvecs that we get from OpenCV camera extrinsics, pattern_to_camera. :param device_tracking_matrices: list of tracking matrices for the tracked device, e.g. laparoscope, marker_to_tracker. :param pattern_tracking_matrices: list of tracking matrices for the calibration object, marker_to_tracker :param method: Choice of OpenCV RobotWorldHandEye method. :return hand-eye, pattern-to-marker transforms as 4x4 matrices

sksurgerycalibration.video.video_calibration_hand_eye.**calibrate_hand_eye_using_stationary_p**

Hand-eye calibration using standard OpenCV methods. This method assumes a single set of tracking data, so it is useful for a stationary, untracked calibration pattern, and a tracked video device, e.g. laparoscope.

**Parameters** **camera_rvecs** – list of rvecs that we get from OpenCV camera

extrinsics, pattern_to_camera. :param camera_tvecs: list of tvecs that we get from OpenCV camera extrinsics, pattern_to_camera. :param tracking_matrices: list of tracking matrices for the tracked device, marker_to_tracker. :param method: Choice of OpenCV Hand-Eye method. :param invert_camera: if True, we invert camera matrices before hand-eye calibration. :return hand-eye transform as 4x4 matrix.

sksurgerycalibration.video.video_calibration_hand_eye.**calibrate_hand_eye_using_tracked_patt**

Hand-eye calibration using standard OpenCV methods. This method assumes two sets of tracking data, so it is useful for a tracked device (e.g. laparoscope) and a tracked calibration object (e.g. chessboard).

**Parameters** **camera_rvecs** – list of rvecs that we get from OpenCV camera

extrinsics, pattern_to_camera. :param camera_tvecs: list of tvecs that we get from OpenCV camera extrinsics, pattern_to_camera. :param device_tracking_matrices: list of tracking matrices for the tracked device, marker_to_tracker. :param pattern_tracking_matrices: list of tracking matrices for the tracked calibration object, marker_to_tracker. :param method: Choice of OpenCV Hand-Eye method. :return hand-eye transform as 4x4 matrix.

sksurgerycalibration.video.video_calibration_hand_eye.**calibrate_pattern_to_tracking_marker**

In some calibration problems, people use a pattern (e.g. chessboard) that is tracked. If you manufacture this well, you should know the pattern_2_marker transform by DESIGN. However, if you assume a STATIONARY video camera, and a moving pattern, this method allows you to use standard hand-eye techniques to estimate the pattern_2_marker transform from at least 2 motions (3 views) around different axes of rotation.

**Parameters** **camera_rvecs** – list of rvecs that we get from OpenCV camera

extrinsics, pattern_to_camera. :param camera_tvecs: list of tvecs that we get from OpenCV camera extrinsics, pattern_to_camera. :param tracking_matrices: list of tracking matrices for the tracked calibration pattern, marker_to_tracker. :param method: Choice of OpenCV Hand-Eye method. :return pattern_to_marker

sksurgerycalibration.video.video_calibration_hand_eye.**guofang_xiao_handeye_calibration**(*rvecs List[ tvecs List[ quat_ nump trans nump* → Tuple[n nump

Guofang Xiao's method. Solve for the hand-eye transformation, as well as the transformation from the pattern to the tracking markers on the calibration object.

This method, developed for the SmartLiver project, assumes both device (laparoscope), and the calibration object are tracked. We also, keep the device (laparoscope) stationary while moving the calibration object.

**Parameters** **rvecs** – Array of rotation vectors, from OpenCV, camera extrinsics

(pattern to camera transform) :type rvecs: List[np.ndarray] :param tvecs: Array of translation vectors, from OpenCV, camera extrinsics (pattern to camera transform) :type tvecs: List[np.ndarray] :param quat_model2hand_array: Array of quaternions representing rotational part of marker-to-hand. :type quat_model2hand_array: np.ndarray :param trans_model2hand_array: Array of quaternions representing translational part of marker-to-hand. :type trans_model2hand_array: np.ndarray :return: two 4x4 matrices as np.ndarray, representing handeye_matrix, pattern2marker_matrix :rtype: np.ndarray, np.ndarray

sksurgerycalibration.video.video_calibration_hand_eye.**set_model2hand_arrays**(*calibration_tracking_ List[T], device_tracking_array: List[T], use_quaternions=Fal* → Tuple[numpy.ndarray, numpy.ndarray]

Guofang Xiao's method. Set the model-to-hand quaternion and translation arrays from tracking data.

**Parameters** **calibration_tracking_array** – Array of tracking data for

calibration target :type calibration_tracking_array: List of tracking data :param device_tracking_array: Array of tracking data for device (e.g. camera) :type device_tracking_array: List of tracking data :param use_quaternions: If True input should be quaternions :type device_tracking_array: bool :return: quaternion model to hand array and translation model to hand array :rtype: np.ndarray, np.ndarray

### 1.3.7 Helper Classes/Functions

Base class for our mono and stereo video camera calibration drivers.

**class** sksurgerycalibration.video.video_calibration_driver_base.**BaseVideoCalibrationDriver**(*i*

Bases: object

Base class for video calibration drivers.

**calibrate**(*flags=0*)
Do the video calibration. Derived classes must implement this.

**get_number_of_views**()
Returns the current number of stored views.

**Returns** number of views

**get_params**()
Copies and returns the parameters.

**get_tracking_data**()
Copies and returns the tracking data.

**get_video_data**()
Copies and returns the video data.

**is_calibration_target_tracked**()
Returns True if we have tracking data for the calibration target.

**is_device_tracked**()
Returns True if we have tracking data for the device.

**load_data**(*dir_name: str*, *file_prefix: str*)
Loads the data from dir_name, and populates this object.

**load_params**(*dir_name: str*, *file_prefix: str*)
Loads the calibration params from dir_name, using file_prefix.

**pop**()
Removes the last grabbed view of data.

**reinit**()
Resets this object, which means, removes stored calibration data and reset the calibration parameters to identity/zero.

**save_data**(*dir_name: str*, *file_prefix: str*)
Saves the data to the given dir_name, with file_prefix.

**save_params**(*dir_name: str*, *file_prefix: str*)
Saves the calibration parameters to dir_name, with file_prefix.

Various functions to help with IO. Not intended for 3rd party clients.

sksurgerycalibration.video.video_calibration_io.**get_annotated_images_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_calib_prefix**(*file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_calibration_tracking_file_name**(*dir_name: str*, *file_prefix: str*, *view_numb int*)

sksurgerycalibration.video.video_calibration_io.**get_device_tracking_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_distortion_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_enumerated_file_glob**(*dir_name: str*, *file_prefix: str*, *type_prefix: str*, *extension_wth_dot: str*)

sksurgerycalibration.video.video_calibration_io.**get_enumerated_file_name**(*dir_name: str*, *file_prefix: str*, *type_prefix: str*, *view_number: str*, *extension_wth_dot: str*)

sksurgerycalibration.video.video_calibration_io.**get_essential_matrix_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_extrinsic_file_names**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_extrinsics_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_filenames_by_glob_expr**(*dir_name: str*, *file_prefix: str*, *type_prefix: str*, *extension_with_dot: str*)

sksurgerycalibration.video.video_calibration_io.**get_fundamental_matrix_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_handeye_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_ids_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_imagepoints_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_images_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_intrinsics_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_l2r_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_left_prefix**(*file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_objectpoints_file_name**(*dir_name: str*, *file_prefix: str*, *view_number: int*)

sksurgerycalibration.video.video_calibration_io.**get_pattern2marker_file_name**(*dir_name: str*, *file_prefix: str*)

sksurgerycalibration.video.video_calibration_io.**get_right_prefix**(*file_prefix: str*)

Various utilities, converters etc., to help video calibration.

sksurgerycalibration.video.video_calibration_utils.**array_contains_tracking_data**(*array_to_check*
Returns True if the array contains some tracking data.

sksurgerycalibration.video.video_calibration_utils.**convert_numpy2d_to_opencv**(*image_points*)
Converts numpy array to Vector of 1x2 vectors containing float32.

**Parameters** **image_points** – numpy [Mx2] array.

**Returns** vector (length M), of 1x2 vectors of float32.

sksurgerycalibration.video.video_calibration_utils.**convert_numpy3d_to_opencv**(*object_points*)
Converts numpy array to Vector of 1x3 vectors containing float32.

**Parameters** **object_points** – numpy [Mx3] array.

**Returns** vector (length M), of 1x3 vectors of float32.

sksurgerycalibration.video.video_calibration_utils.**convert_pd_to_opencv**(*ids*, *object_points*, *image_points*)
The PointDetectors from scikit-surgeryimage aren't quite compatible with OpenCV.

sksurgerycalibration.video.video_calibration_utils.**detect_points_in_canonical_space**(*point_det*
*minimum_poi*
*video_da*
*images*,
*camera_matr*
*distortion_coef*
*reference_ids*,
*reference_ima*
*reference_ima*

Method that does the bulk of the heavy lifting in Datta 2009.

**Parameters**

- **point_detector** –
- **minimum_points_per_frame** –

- **video_data** –
- **images** –
- **camera_matrix** –
- **distortion_coefficients** –
- **reference_ids** –
- **reference_image_points** –
- **reference_image_size** –

**Returns**

sksurgerycalibration.video.video_calibration_utils.**detect_points_in_stereo_canonical_space**

Method that does the bulk of the heavy lifting in Datta 2009.

The reason we need a combined stereo one, instead of calling the mono one twice, is because at any point, if either left or right channel fails feature detection, we need to drop that image from BOTH channels.

**Parameters**

- **left_point_detector** –
- **right_point_detector** –
- **minimum_points_per_frame** –
- **left_video_data** –
- **left_images** –
- **left_camera_matrix** –
- **left_distortion_coeffs** –
- **right_video_data** –
- **right_images** –
- **right_camera_matrix** –

- **right_distortion_coeffs** –
- **reference_ids** –
- **reference_image_points** –
- **reference_image_size** –

**Returns**

sksurgerycalibration.video.video_calibration_utils.**distort_points**(*image_points*, *camera_matrix*, *distortion_coeffs*)

Distorts image points, reversing the effects of cv2.undistortPoints.

Slow, but should do for now, for offline calibration at least.

**Parameters**

- **image_points** – undistorted image points.
- **camera_matrix** – [3x3] camera matrix
- **distortion_coeffs** – [1x5] distortion coefficients

**Returns** distorted points

sksurgerycalibration.video.video_calibration_utils.**extrinsic_matrix_to_vecs**(*transformation_matri*

Method to convert a [4x4] rigid body matrix to an rvec and tvec.

**Parameters** **transformation_matrix** – [4x4] rigid body matrix.

:return [3x1] Rodrigues rotation vec, [3x1] translation vec

sksurgerycalibration.video.video_calibration_utils.**extrinsic_vecs_to_matrix**(*rvec*, *tvec*)

Method to convert rvec and tvec to a 4x4 matrix.

**Parameters**

- **rvec** – [3x1] ndarray, Rodrigues rotation params
- **rvec** – [3x1] ndarray, translation params

**Returns** [3x3] ndarray, Rotation Matrix

sksurgerycalibration.video.video_calibration_utils.**filter_common_points_all_images**(*left_ids*, *left_object_*, *left_image_*, *right_ids*, *right_imag*, *minimum_point*

Loops over each images's data, filtering per image. See: filter_common_points_per_image :return: Vectors of outputs from filter_common_points_per_image

sksurgerycalibration.video.video_calibration_utils.**filter_common_points_per_image**(*left_ids*, *left_object_p*, *left_image_p*, *right_ids*, *right_image_*, *minimum_points*)

For stereo calibration, we need common points in left and right. Remember that a point detector, may provide different numbers of points for left and right, and they may not be sorted.

**Parameters**

- **left_ids** – ndarray of integer point ids
- **left_object_points** – Vector of Vector of 1x3 float 32
- **left_image_points** – Vector of Vector of 1x2 float 32
- **right_ids** – ndarray of integer point ids
- **right_image_points** – Vector of Vector of 1x2 float 32
- **minimum_points** – the number of minimum common points to accept

**Returns** common ids, object_points, left_image_points, right_image_points

sksurgerycalibration.video.video_calibration_utils.**map_points_from_canonical_to_original**(*im*, *image_ag*, *video_id*, *object_image_ho*, *mapping_ra*, *photo_ca*, *error_di*, *tolerance_ti*)

Utility method to map image points, detected in a canonical face on image, back to the original image space.

**Parameters**

- **images_array** –
- **image_index** –
- **video_data** –
- **ids** –
- **object_points** –
- **image_points** –
- **homography** –

- **camera_matrix** –
- **distortion_coeffs** –

**Returns**

sksurgerycalibration.video.video_calibration_utils.**match_points_by_id**(*ids_1*, *points_1*, *ids_2*, *points_2*)

Returns an ndarray of matched points, matching by their identifier.

**Parameters**

- **ids_1** – ndarray [Mx1] list of ids for points_1
- **points_1** – ndarray [Mx2 or 3] of 2D or 3D points
- **ids_2** – ndarray [Nx1] list of ids for points_2
- **points_2** – ndarray [Nx2 or 3] of 2D or 3D points

**Returns** ndarray. Number of rows is the number of common points by ids.

Video Calibration functions, that wrap OpenCV functions mainly.

sksurgerycalibration.video.video_calibration_wrapper.**mono_handeye_calibration**(*object_points: List[T], image_points: List[T], camera_matrix: numpy.ndarray, camera_distortion: numpy.ndarray, device_tracking_arra List[T], pattern_tracking_arra List[T], rvecs: List[numpy.ndarra tvecs: List[numpy.ndarra override_pattern2mark numpy.ndarray = None, use_opencv: bool = True, do_bundle_adjust: bool = False*)

Wrapper around handeye calibration functions and reprojection / reconstruction error metrics.

**Parameters**

- **object_points** (*List*) – Vector of Vectors of 1x3 object points, float32
- **image_points** (*List*) – Vector of Vectors of 1x2 object points, float32
- **ids** (*List*) – Vector of ndarrays containing integer point ids.
- **camera_matrix** (*np.ndarray*) – Camera intrinsic matrix
- **camera_distortion** (*np.ndarray*) – Camera distortion coefficients
- **device_tracking_array** (*List*) – Tracking data for camera (hand)
- **pattern_tracking_array** (*List*) – Tracking data for calibration target
- **rvecs** (*List[np.ndarray]*) – Vector of 3x1 ndarray, Rodrigues rotations for each camera
- **tvecs** (*List[np.ndarray]*) – Vector of [3x1] ndarray, translations for each camera
- **override_pattern2marker** – If provided a 4x4 pattern2marker that

is taken as constant. :param use_opencv: If True we use OpenCV based methods, if false, Guofang Xiao's method. :param do_bundle_adjust: If True we do an additional bundle adjustment at the end. Needs pattern tracking too. :return: Reprojection error, handeye matrix, patter to marker matrix :rtype: float, float, np.ndarray, np.ndarray

sksurgerycalibration.video.video_calibration_wrapper.**mono_video_calibration**(*object_points*, *image_points*, *image_size*, *flags=0*)

Calibrates a video camera using Zhang's 2000 method, as implemented in OpenCV. We wrap it here, so we have a place to add extra validation code, and a space for documentation. The aim is to check everything before we pass it to OpenCV, and raise Exceptions consistently for any error we can detect before we pass it to OpenCV, as OpenCV just dies without throwing exceptions.

- N = number of images
- M = number of points for that image
- rvecs = list of 1x3 Rodrigues rotation parameters
- tvecs = list of 3x1 translation vectors
- camera_matrix = [3x3] ndarray containing fx, fy, cx, cy
- dist_coeffs = [1x5] ndarray, containing distortion coefficients

**Parameters**

- **object_points** – Vector (N) of Vector (M) of 1x3 points of type float
- **image_points** – Vector (N) of Vector (M) of 1x2 points of type float
- **image_size** – (x, y) tuple, size in pixels, e.g. (1920, 1080)
- **flags** – OpenCV flags to pass to calibrateCamera().

**Returns** RMS projection error, camera_matrix, dist_coeffs, rvecs, tvecs

sksurgerycalibration.video.video_calibration_wrapper.**stereo_calibration_extrinsics**(*common_o*
*com-*
*mon_left_i*
*com-*
*mon_right_*
*l_rvecs*,
*l_tvecs*,
*over-*
*ride_left_in*
*over-*
*ride_left_d*
*over-*
*ride_right_*
*over-*
*ride_right_*
*over-*
*ride_l2r_rv*
*over-*
*ride_l2r_tv*

Simply re-optimises the extrinsic parameters. :return: error, l_rvecs, l_tvecs

sksurgerycalibration.video.video_calibration_wrapper.**stereo_handeye_calibration**(*l2r_rmat: numpy.ndarray, l2r_tvec: numpy.ndarray, left_ids: List[T], left_object_poi List[T], left_image_poi List[T], right_ids: List[T], right_image_po List[T], left_camera_ma numpy.ndarray, left_camera_di numpy.ndarray, right_camera_r numpy.ndarray, right_camera_d numpy.ndarray, device_tracking_a List[T], calibration_tracking_a List[T], left_rvecs: List[numpy.nda left_tvecs: List[numpy.nda override_pattern2m use_opencv: bool = True, do_bundle_adju bool = False*)

Wrapper around handeye calibration functions and reprojection / reconstruction error metrics.

**Parameters**

- **l2r_rmat** (*np.ndarray*) – [3x3] ndarray, rotation for l2r transform
- **l2r_tvec** (*np.ndarray*) – [3x1] ndarray, translation for l2r transform
- **left_ids** (*List*) – Vector of ndarrays containing integer point ids.
- **left_object_points** (*List*) – Vector of Vector of 1x3 of type float32
- **left_image_points** (*List*) – Vector of Vector of 1x2 of type float32

- **right_ids** (*List*) – Vector of ndarrays containing integer point ids.
- **right_image_points** (*List*) – Vector of Vector of 1x3 of type float32
- **left_camera_matrix** (*np.ndarray*) – Camera intrinsic matrix
- **left_camera_distortion** (*np.ndarray*) – Camera distortion coefficients
- **right_camera_matrix** (*np.ndarray*) – Camera intrinsic matrix
- **right_camera_distortion** (*np.ndarray*) – Camera distortion coefficients
- **device_tracking_array** (*List*) – Tracking data for camera (hand)
- **calibration_tracking_array** (*List*) – Tracking data for calibration target
- **left_rvecs** – Vector of 3x1 ndarray, Rodrigues rotations for each

camera :type left_rvecs: List[np.ndarray] :param left_tvecs: Vector of [3x1] ndarray, translations for each camera :type left_tvecs: List[np.ndarray] :param right_rvecs: Vector of 3x1 ndarray, Rodrigues rotations for each camera :type right_rvecs: List[np.ndarray] :param right_tvecs: Vector of [3x1] ndarray, translations for each camera :type right_tvecs: List[np.ndarray] :param override_pattern2marker: If provided a 4x4 pattern2marker that is taken as constant. :param use_opencv: If True we use OpenCV based methods, if false, Guofang Xiao's method. :param do_bundle_adjust: If True we do an additional bundle adjustment at the end. :return: Reprojection error, reconstruction error, left handeye matrix, left pattern to marker matrix, right handeye, right pattern to marker :rtype: float, float, np.ndarray, np.ndarray, np.ndarray, np.ndarray

sksurgerycalibration.video.video_calibration_wrapper.**stereo_video_calibration**(*left_ids*, *left_object_points*, *left_image_points*, *right_ids*, *right_object_point*
*right_image_point*
*im-*
*age_size*, *flags=1*, *over-*
*ride_left_intrinsics*
*over-*
*ride_left_distortio*
*over-*
*ride_right_intrinsi*
*over-*
*ride_right_distorti*
*over-*
*ride_l2r_rmat=No*
*over-*
*ride_l2r_tvec=Nor*

Default stereo calibration, using OpenCV methods.

We wrap it here, so we have a place to add extra validation code, and a space for documentation. The aim is to check everything before we pass it to OpenCV, and raise Exceptions consistently for any error we can detect before we pass it to OpenCV.

**Parameters**

- **left_ids** – Vector of ndarrays containing integer point ids.
- **left_object_points** – Vector of Vectors of 1x3 object points, float32
- **left_image_points** – Vector of Vectors of 1x2 object points, float32

- **right_ids** – Vector of ndarrays containing integer point ids.
- **right_object_points** – Vector of Vectors of 1x3 object points, float32
- **right_image_points** – Vector of Vectors of 1x2 object points, float32
- **image_size** – (x, y) tuple, size in pixels, e.g. (1920, 1080)
- **flags** – OpenCV flags to pass to calibrateCamera().

**Returns**

## 1.4 Camera Calibration

scikit-surgerycalibration provides a range of functions/classes mono, stereo and hand-eye calibration.

The Jupyter notebook for this page can be found in the docs/tutorials folder of the code repository.

- Setting up paths and virtual environment

```
cd $HOME/repositories/Scikit-Surgery/scikit-surgerycalibration
export PYTHONPATH=$HOME/repositories/Scikit-Surgery/scikit-surgerycalibration
conda activate scikit-surgerycalibrationVE
```

### 1.4.1 Required modules

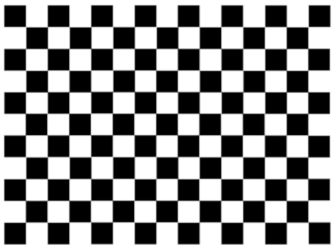
This tutorial requires numpy, opencv-contib-python, scikit-surgeryimage, scikit-surgerycalibration, and matplotlib (for inline image plotting).

[1]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sksurgeryimage.calibration.chessboard_point_detector as cpd
import sksurgerycalibration.video.video_calibration_driver_mono as mc
```

### 1.4.2 Chessboard Calibration

The calibrator can be used with any of the point detectors available in scikit-surgeryimage.

In this example, we will use the chessboard detector, see the Point Detectors Tutorial for some examples of other options.



We will use this 14x10 chessboard pattern for calibration.

You can either print it out, or display it on a mobile phone. If printed, the square size is 6mm, you will have to measure/estimate the size if displayed on a phone screen.

### Initial setup

[2]:
```
chessboard_corners = (14, 10)
min_points_to_detect = chessboard_corners[0] * chessboard_corners[1]
square_size_mm = 6 # Change as appropriate if displayed on phone

video_source = cv2.VideoCapture(0)

detector = cpd.ChessboardPointDetector(chessboard_corners, square_size_mm)
calibrator = mc.MonoVideoCalibrationDriver(detector, min_points_to_detect)
```

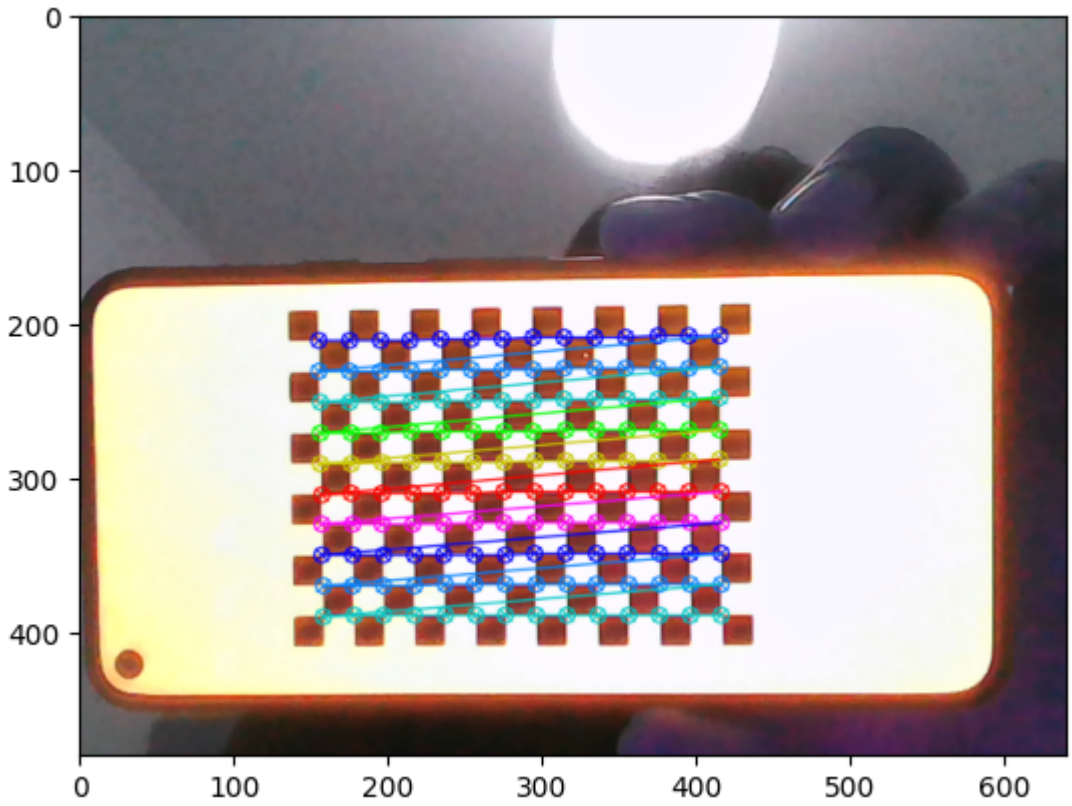
### Capture some frames

If you hold the chessboard in view of your camera, and run the below cell repeatedly, it will grab some frames. At least 2 frames are needed for a calibration, but in general the more frames, the better the calibration.

[5]:
```
# Read a frame and detect points
_, frame = video_source.read()
number_of_points = calibrator.grab_data(frame)

# Draw chessboard points on frame
if number_of_points > 0:
    img_pts = calibrator.video_data.image_points_arrays[-1]
    ###print(img_pts.shape)##(140, 1, 2)
    frame = cv2.drawChessboardCorners(frame,
                                      chessboard_corners,
                                      img_pts,
                                      number_of_points)
plt.imshow(frame)
plt.show()

# Print some details
number_of_views = calibrator.get_number_of_views()
print(f"Detected {number_of_points} corners")
print(f"{number_of_views} frames captured")
```




```
Detected 140 corners
3 frames captured
```

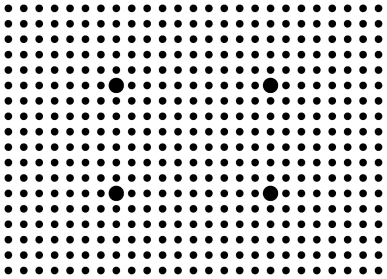
### Perform calibration

[6]:
```
if number_of_views > 1:
    proj_err, params = calibrator.calibrate()
    #proj_err, recon_err, params = calibrator.calibrate()
    print(f'Reprojection (2D) error is: \n {proj_err}')
    #print(f'Reprojection (3D) error is: \n {recon_err}')
    print(f'Intrinsics are: \n  {params.camera_matrix}')
    print(f'Distortion matrix is:  \n {params.dist_coeffs}')
```

```
Reprojection (2D) error is:
 0.19281401614557028
Intrinsics are:
  [[809.69585633   0.         432.93998965]
 [  0.         802.69795671 291.847329  ]
 [  0.           0.           1.        ]]
Distortion matrix is:
 [[-0.06609416 -0.10402796  0.00859668  0.00367779  2.34223261]]
```

[7]:
```
video_source.release()
```

### 1.4.3 Dot Calibration



We will use this dot pattern.

Due to the relatively small size of the dots, performance when displayed on a phone screen may vary.

[8]:
```
import cv2
import numpy as np
import matplotlib.pyplot as plt
import sksurgeryimage.calibration.dotty_grid_point_detector as dpd
import sksurgerycalibration.video.video_calibration_driver_mono as mc
```

#### Initial setup for dot detector

[9]:
```
number_of_dots = [18, 25]
pixels_per_mm = 80
dot_separation = 5

# Don't need to capture all dots for calibration to work
min_points_to_detect = (number_of_dots[0] / number_of_dots[1]) / 2
model_points = dpd.get_model_points(number_of_dots,
                                    pixels_per_mm,
                                    dot_separation)

# Location of the large dots in the pattern
fiducial_indexes = [132, 142, 307, 317]

# Image size
reference_image_size = [1900, 2600]

# data in scikit-surgerycalibration/tests/data/dot_calib
left_intrinsic_matrix = np.loadtxt("../../tests/data/dot_calib/calib.left.intrinsics.
↪txt")
left_distortion_matrix = np.loadtxt("../../tests/data/dot_calib/calib.left.distortion.
↪txt")

video_source = cv2.VideoCapture(0)


detector = \
    dpd.DottyGridPointDetector(
        model_points,
        fiducial_indexes,
        left_intrinsic_matrix,
        left_distortion_matrix,
        reference_image_size=(reference_image_size[1],
                              reference_image_size[0])
```

(continues on next page)

(continued from previous page)

```
        )

calibrator = mc.MonoVideoCalibrationDriver(detector, min_points_to_detect)
```

### Capture some frames

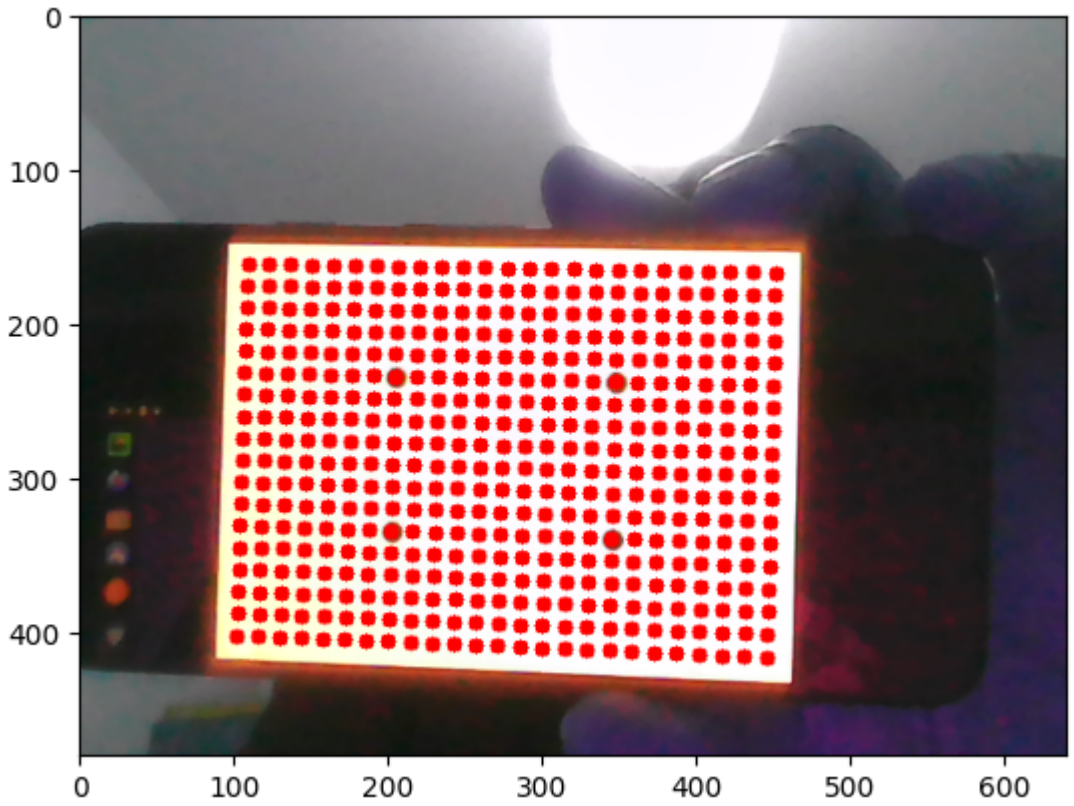
If you hold the chessboard in view of your camera, and run the below cell repeatedly, it will grab some frames. At least 2 frames are needed for a calibration, but in general the more frames, the better the calibration.

[12]:
```
# Read a frame and detect points
_, frame = video_source.read()
number_of_points = calibrator.grab_data(frame)

if number_of_points > 0:
    img_pts = calibrator.video_data.image_points_arrays[-1]
    ###print(img_pts.shape)##(448, 1, 2)
    # Draw detected points on frame
    for point in img_pts:
        frame = cv2.circle(frame, ( int(point[0,0]), int(point[0,1])), 5, (255, 0 ,0),
↪ -1)

plt.imshow(frame)
plt.show()

# Print some details
number_of_views = calibrator.get_number_of_views()
print(f"Detected {number_of_points} points")
print(f"{number_of_views} frames captured")
```




```
Detected 450 points
3 frames captured
```

### Release video_source

[13]:
```
video_source.release()
```

### Perform Calibration

[14]:
```
if number_of_views > 1:
    proj_err, params = calibrator.calibrate()
    #proj_err, recon_err, params = calibrator.calibrate()
    print(f'Reprojection (2D) error is: \n {proj_err}')
    #print(f'Reprojection (3D) error is: \n {recon_err}')
    print(f'Intrinsics are: \n  {params.camera_matrix}')
    print(f'Distortion matrix is:  \n {params.dist_coeffs}')
```

```
Reprojection (2D) error is:
 0.08160305101574904
Intrinsics are:
  [[546.86680246   0.         342.68154282]
 [  0.         545.55158144 232.75969187]
 [  0.           0.           1.        ]]
Distortion matrix is:
 [[-0.05469762  0.2762558   0.00113643  0.00038568 -0.35911484]]
```

### 1.4.4 Stereo Calibration

Stereo calibration works in the same way as mono calibration, except with two detectors and two images:

```
import sksurgerycalibration.video.video_calibration_driver_stereo as sc

# Define detectors

calibrator = sc.StereoVideoCalibrationDriver(left_detector,
                                             right_detector,
                                             min_points)

# Acquire images

number_of_points = calibrator.grab(left_image, right_image)
```

[ ]:

# Index

## E



## F



## G

## H



## I



## L



## M



## P

## R



## S

## T