scikit-surgery Documentation

Thomas Dowrick

Jul 28, 2022

Module Reference

SciKit-Surgery is a collection of compact libraries developed for surgical navigation. Individual libraries can be combined using Python to create clinical applications for translational research. However because each application's requirements are unique the individual SciKit-Surgery libraries are kept independent, enabling them to be maintained, modified and combined in new ways to create new clinical applications. Keeping the libraries independent enables researchers to implement novel algorithms within a small library that can be readily reused and built on by the research community.

A typical clinical application might consist of an imaging source (e.g. SciKit-SurgeryBK to stream ultrasound images), a tracking source (e.g. SciKitSurgery-NDITracker) to locate the images in space, an image processor (e.g. SciKit-SurgeryTorch) to segment anatomy from the image, and a visualisation layer (e.g. SciKit-SurgeryVTK)

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

CHAPTER 1

Application Examples

One way to get an introduction to SciKit-Surgery is to take a look at some the applications currently using SciKit-Surgery libraries;

The SmartLiver augmented reality guidance system for key hole liver surgery is built on the SciKit-Surgery libraries and within our ISO-13485 quality management system. SmartLiver is currently undergoing clinical trials at the Royal Free Hospital London. SmartLiver uses SciKit-SurgeryCore, SciKit-SurgeryBK, SciKit-SurgeryImage, SciKit-SurgeryVTK, SciKitSurgery-NDITracker, SciKit-SurgerySpeech, SciKit-SurgeryTF, SciKit-SurgerySurfaceMatch, SciKit-SurgeryTorch, and SciKit-SurgeryCalibration. The image above shows the user interface using SciKit-SurgerySpeech being tested in theatre.

SnappySonic is an ultrasound simulator developed primarily for educational purposes. SnappySonic uses SciKit-SurgeryUtils, SciKit-SurgeryNDITracker, SciKit-SurgeryArucoTracker, and SciKit-SurgeryImage.

SciKit-SurgeryBARD uses SciKit-SurgeryCalibration, SciKit-SurgeryCore, SciKit-SurgeryUtils, SciKit-SurgeryVTK, SciKit-SurgerySpeech, and SciKit-SurgeryArucoTracker to build a Basic Augmented Reality Demonstrator. SciKit-SurgeryBARD was developed for educational purposes, but by swapping SciKit-SurgeryArucoTracker for SciKitSurgery-NDITracker it can be used as a minimal system for surgical augmented reality.

SciKit-SurgeryFRED was developed for teaching and research for registration applied to image guided interventions. SciKit-Surgery provides a graphical front end to the image processing classes within SciKit-SurgeryImage and the image registration classes within SciKit-SurgeryCore.

CHAPTER 2

Tutorials

Tutorials are split into three groups, those that show how to assemble SciKit-Surgery libraries into an application, those that concentrate on the workings a single application, and those that are aimed at general education in image guided interventions using SciKit-Surgery.

General Tutorials

- Use SciKit-SurgeryUtils and SciKit-SurgeryArUcoTracker to build an AR application using your webcam.
- ROS Integration

scikit-surgeryvtk

- How To Use VTKOverlayWindow
- How To Add Text To VTKOverlayWindow
- Using The Rendering Generator
- Distance Fields & Voxelisation

scikit-surgeryimage

- Point/Chessboard detectors

scikit-surgerycalibration

* Camera Calibration

Educational Tutorials

- Use a ready made application to investigate different ways of presenting augmented reality.
- Improve your impact by creating high quality software implementations of your research.
- Camera calibration using your phone or webcam.
- Make and Calibrate a Pointer.
- Online Fiducial Registration Tutorial.
- Point Based Registration using Lego or anatomical phantoms.

- Camera Calibration of Laparoscopes

CHAPTER 3

Packages

- scikit-surgerycore - Algorithms/tools common to all scikit-surgery packages
- scikit-surgeryimage - Image processing algorithms using OpenCV
- scikit-surgeryvtk - Implements VTK functionality for IGS applications
- scikit-surgeryutils - Example applications/utilities
- scikit-surgerycalibration - Calibration algorithms (camera/pointer/ultrasound etc)
- scikit-surgerysurfacematch - Stereo reconstruction and point cloud matching
- scikit-surgerytf - IGS models implemented in TensorFlow
- scikit-surgerytorch - IGS models implemented in PyTorch
- scikit-surgerynditracker - Interface for Northern Digital (NDI) trackers. Vicra, Spectra, Vega, Aurora.
- scikit-surgeryarucotracker - Interface for OpenCV ARuCo.
- scikit-surgeryspeech - Speech/Wakeword detection

3.1 scikit-surgerycore

3.1.1 Quaternion Averaging

Quaternion averaging functions

sksurgerycore.algorithms.averagequaternions.**average_quaternions**(*quaternions*)
Calculate average quaternion

Params quaternions is a Nx4 numpy matrix and contains the quaternions to average in the rows. The quaternions are arranged as (w,x,y,z), with w being the scalar

Returns the average quaternion of the input. Note that the signs of the output quaternion can be reversed, since q and -q describe the same orientation

sksurgerycore.algorithms.averagequaternions.**weighted_average_quaternions**(*quaternions*, *weights*)

Average multiple quaternions with specific weights

Params quaternions is a Nx4 numpy matrix and contains the quaternions to average in the rows. The quaternions are arranged as (w,x,y,z), with w being the scalar

Params weights The weight vector w must be of the same length as the number of rows in the

Returns the average quaternion of the input. Note that the signs of the output quaternion can be reversed, since q and -q describe the same orientation

Raises ValueError if all weights are zero

3.1.2 Registration Error Calculation

Registration Error Calculations

sksurgerycore.algorithms.errors.**compute_fre**(*fixed*, *moving*, *rotation*, *translation*)

Computes the Fiducial Registration Error, equal to the root mean squared error between corresponding fiducials.

Parameters

- **fixed** – point set, N x 3 ndarray
- **moving** – point set, N x 3 ndarray of corresponding points
- **rotation** – 3 x 3 ndarray
- **translation** – 3 x 1 ndarray

Returns Fiducial Registration Error (FRE)

sksurgerycore.algorithms.errors.**compute_fre_from_fle**(*fiducials*, *mean_fle_squared*)

Computes an estimation of FRE from FLE and a list of fiducial locations.

See: Fitzpatrick (1998), equation 10.

Parameters

- **fiducials** – Nx3 ndarray of fiducial points
- **mean_fle_squared** – expected (mean) FLE squared

Returns mean FRE squared

sksurgerycore.algorithms.errors.**compute_tre_from_fle**(*fiducials*, *mean_fle_squared*, *target_point*)

Computes an estimation of TRE from FLE and a list of fiducial locations.

See: Fitzpatrick (1998), equation 46.

Parameters

- **fiducials** – Nx3 ndarray of fiducial points
- **mean_fle_squared** – expected (mean) FLE squared
- **target_point** – a point for which to compute TRE.

Returns mean TRE squared

sksurgerycore.algorithms.errors.**validate_procrustes_inputs**(*fixed*, *moving*)

Validates the fixed and moving set of points

1. fixed and moving must be numpy array

2. fixed and moving should have 3 columns
3. fixed and moving should have at least 3 rows
4. fixed and moving should have the same number of rows

Parameters

- **fixed** – point set, N x 3 ndarray
- **moving** – point set, N x 3 ndarray of corresponding points

Returns nothing

Raises TypeError, ValueError

3.1.3 Pivot Calibration

Functions for pivot calibration.

sksurgerycore.algorithms.pivot.**pivot_calibration**(*tracking_matrices*)

sksurgerycore.algorithms.pivot_calibration is depreciated from v0.6.0, please use skscurgerycalibration.algorithms.pivot_calibration from from scikit-surgerycalibration instead

Raises NotImplementedError

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

sksurgerycore.algorithms.pivot_calibration_with_ransac is depreciated from v0.6.0, please use skscurgerycalibration.algorithms.pivot_calibration_with_ransac from from scikit-surgerycalibration instead

Raises NotImplementedError

3.1.4 Procrustes Registration

Functions for point based registration using Orthogonal Procrustes.

sksurgerycore.algorithms.procrustes.**orthogonal_procrustes**(*fixed*, *moving*)

Implements point based registration via the Orthogonal Procrustes method.

Based on Arun's method:

> Least-Squares Fitting of two, 3-D Point Sets, Arun, 1987, 10.1109/TPAMI.1987.4767965.

Also see this and this.

Parameters

- **fixed** – point set, N x 3 ndarray
- **moving** – point set, N x 3 ndarray of corresponding points

Returns 3x3 rotation ndarray, 3x1 translation ndarray, FRE

Raises ValueError

3.1.5 Tracker Data Smoothing

3.1.6 Math Utilities

Various small maths utilities.

sksurgerycore.algorithms.vector_math.**distance_from_line**(*p_1*, *p_2*, *p_3*)
Computes distance of a point p_3, from a line defined by p_1 and p_2.

See here.

Returns euclidean distance

3.1.7 Configuration Manager

Class to load application configuration information from a json file.

Design principles:

- All errors as Exceptions
- Fail early in constructor, so the rest of the program never has an invalid instance of ConfigurationManager. If its constructed, its valid.
- Setter and Getter do a deepcopy, so only suitable for small config files.
- Pass ConfigurationManager to any consumer of the data, its up to the consumer to know where to find the data.

class sksurgerycore.configuration.configuration_manager.**ConfigurationManager**(*file_name*, *write_on_setter=Fa*

Bases: object

Class to load application configuration from a json file. For example, this might be used at the startup of an application.

Parameters

- **file_name** – a json file to read.
- **write_on_setter** – if True, will write back to the same file whenever the setter is called.

Raises All errors raised as various Exceptions.

get_copy()
Returns a copy of the data read from file.

Returns deep copy of whatever data structure is stored internally.

get_dir_name()
Returns the directory name of the file that was used when creating the ConfigurationManager.

Returns str dir name

get_file_name()
Returns the absolute filename that was used when the ConfigurationManager was created.

Returns str absolute file name

set_data(*config_data*)
Stores the provided data internally.

Note that: you would normally load settings from disk, and then use get_copy() to get a copy, change some settings, and then use set_data() to pass the data structure back in. So, the data provided for this method should still represent the settings you want to save, not just be a completely arbitrary data structure.

Parameters **config_data** – data structure representing your settings.

3.1.8 Data Loading

Functions to load MITK's mps point set file.

sksurgerycore.io.load_mps.**load_mps**(*file_name*)

Load a pointset from a .mps file. For now, just loads points, without geometry information.

Parameters **file_name** – string representing file path.

Returns ids (length N), points (Nx3)

3.1.9 Matrix Functions

Construct 3x3 rotation matrices for rotating around the x, y and z axes individually, as well as 3x3 rotation matrices from sequences of three Euler angles.

sksurgerycore.transforms.matrix.**construct_rigid_transformation**(*rot_m*, *trans_v*)

Construct a 4x4 rigid-body transformation from a 3x3 rotation matrix and a 3x1 vector as translation

Parameters

- **rot_m** – 3x3 rotation matrix, numpy array
- **trans_v** – 3x1 vector as translation, numpy array

Returns rigid_transformation – 4x4 rigid transformation matrix,

numpy array

sksurgerycore.transforms.matrix.**construct_rotm_from_euler**(*angle_a*, *angle_b*, *angle_c*, *sequence*, *is_in_radians=True*)

Construct a rotation matrix from a sequence of three Euler angles, to pre-multiply column vectors. In the case of tracking, the column vector is under the local coordinate system and the resulting vector is under the world (reference) coordinate system. The three elemental rotations represented by the Euler angles are about the INTRINSIC axes. They can also be interpreted to be about the EXTRINSIC axes, in the reverse order.

Parameters

- **angle_a** – first Euler angle, float, int allowed if in degrees
- **angle_b** – second Euler angle, float, int allowed if in degrees
- **angle_c** – third Euler angle, float, int allowed if in degrees
- **sequence** – the sequence of axes the three elemental rotations are about, with respect to the intrinsic axes, string
- **is_in_radians** – if the angles are in radians, default being True, bool

Returns rot_m – the 3x3 rotation matrix, numpy array

Raises TypeError if angles are not float or of difference types

Raises ValueError if sequence is not 3 letters long or contains letters other than x, y or z

sksurgerycore.transforms.matrix.**construct_rx_matrix**(*angle*, *is_in_radians=True*)
Construct a rotation matrix for rotation around the x axis.

Parameters **angle** – the angle to rotate radians, float

Returns rot_x – the 3x3 rotation matrix constructed, numpy array

Raises TypeError if angle is not float or int

sksurgerycore.transforms.matrix.**construct_ry_matrix**(*angle*, *is_in_radians=True*)
Construct a rotation matrix for rotation around the y axis.

Parameters

- **angle** – the angle to rotate, float
- **is_in_radians** – if angle is in radians, default being True, bool

Returns rot_y – the 3x3 rotation matrix constructed, numpy array

Raises TypeError if angle is not float or int

sksurgerycore.transforms.matrix.**construct_rz_matrix**(*angle*, *is_in_radians=True*)
Construct a rotation matrix for rotation around the z axis.

Parameters

- **angle** – the angle to rotate, float
- **is_in_radians** – if angle is in radians, default being True, bool

Returns rot_z – the 3x3 rotation matrix constructed, numpy array

Raises TypeError if angle is not float or int

3.1.10 Transform Manager

Class implementing a general purpose 4x4 transformation matrix manager.

class sksurgerycore.transforms.transform_manager.**TransformManager**
Bases: object

Class for managing 4x4 transformation matrices. This class is NOT designed to be thread-safe.

The transforms are required to be 4x4 matrices. There is no checking that the upper left 3x3 is an orthonormal rotation matrix.

Usage:

```
tm = TransformManager()

# Imagine some example transformations:
t1 = np.eye(4)
t2 = np.eye(4)
t3 = np.eye(4)

# Add transformations to the TransformManager.
tm.add("model2world", t1)
tm.add("hand2eye",t2)
tm.add("hand2world",t3)

# Returns a transform from model to eye,
# by working through the above transforms.
t4 = tm.get("model2eye")
```

and so on.

add(*name*, *transform*)
Adds a transform called name. If the name already exists, the corresponding transform is replaced without warning.

Parameters

- **name** – the name of the transform, e.g. model2world
- **transform** – the transform, e.g. 4x4 matrix

count()
Returns how many transforms are in the manager. Internally this class also stores the inverse, so this method will count those matrices as well.

exists(*name*)
Returns True if the transform exists in the manager, and False otherwise. Internally this class stores the inverse. So, if you add model2world, you are also implicitly adding world2model, so this method will return True for both the originally added transform, and its own inverse.

static flip_name(*name*)
Returns the inverse name.

Parameters **name** – the name of a transformation, e.g. model2world

Returns str – the opposite transformation name, e.g. world2model

get(*name*)
Returns the named transform or throws ValueError.

Raises ValueError

static is_valid_name(*name*)
Validates the name, which must match "^([a-z]+)2([a-z]+)$".

i.e. one or more lowercase letters, followed by the number 2, followed by one or more lowercase letters.

For example:

```
a2b
model2world
```

Identity transforms such as model2model raise ValueError.

Parameters **name** – the name of the transform, eg. model2world

Raises TypeError, ValueError

Returns str, str – parts of string before and after the 2.

static is_valid_transform(*transform*)
Validates the transform as a 4x4 numpy matrix.

Parameters **transform** – 4x4 transformation matrix.

Raises TypeError, ValueError

multiply_point(*name*, *points*)
Multiplies points (4xN) by the named transform (4x4).

Returns ndarray – 4xN matrix of transformed points

Raises ValueError

remove(*name*)
Removes a transform from the manager. If the transform name doesn't exist, will throw ValueError.

Raises ValueError

3.1.11 File Utilities

File processing utils.

sksurgerycore.utilities.file_utilities.**get_absolute_path_of_file**(*file_name*, *dir_name=None*)
Filenames in our .json config could be absolute or relative to the current working dir. This method tries to find the valid, full file path.

Parameters

- **file_name** –
- **dir_name** – prefix, for example, the dirname of our .json file.

Returns absolute path name of file if found, otherwise None.

Various file utilities, often calling standard functions in the os package, but throwing nice informative Exception messages.

sksurgerycore.utilities.validate_file.**validate_is_file**(*file_name*)
Check if file_name is a file.

Parameters **file_name** – string containing path name

Raises TypeError if not string, ValueError if not file

Returns True

sksurgerycore.utilities.validate_file.**validate_is_writable_file**(*file_name*)
Check if file_name is a writable file.

Parameters **file_name** – string containing path name

Raises TypeError if not string, ValueError if not file

Returns True

3.1.12 Matrix Validation

Various validation routines for checking matrices.

sksurgerycore.utilities.validate_matrix.**validate_camera_matrix**(*matrix*)
Validates that a matrix is a camera (intrinsic) matrix.

1. Is a numpy array
2. Is 2D
3. Has 3 rows
4. Has 3 columns

Parameters **matrix** – camera matrix

Raises TypeError, ValueError if not

Returns True

sksurgerycore.utilities.validate_matrix.**validate_distortion_coefficients**(*matrix*)
Validates that a matrix is a set of OpenCV style distortion coefficients.

1. Is a numpy array
2. Is 2D
3. Has 1 row
4. Has either 4, 5, 8, 12 or 14 columns

Parameters **matrix** – set of distortion coefficients

Raises TypeError, ValueError if not

Returns True

sksurgerycore.utilities.validate_matrix.**validate_rigid_matrix**(*matrix*)
Validates that a matrix is a 4x4 rigid transform.

Parameters **matrix** – rigid transform

Raises TypeError, ValueError if not

Returns True

sksurgerycore.utilities.validate_matrix.**validate_rotation_matrix**(*matrix*)
Validates that a matrix is rotation matrix.

1. Is a numpy array
2. Is 2D
3. Has 3 rows
4. Has 3 columns
5. Is orthogonal, i.e., transpose(matrix) * matrix = identity matrix.
6. Is its determinant positive (+1) (c.f., it is a reflection matrix (improper rotation) if the determinant is negative (-1))

Parameters **matrix** – rotation matrix

Raises TypeError, ValueError if not

Returns True

sksurgerycore.utilities.validate_matrix.**validate_translation_column_vector**(*matrix*)
Validates that a translation matrix is a column vector.

1. Is numpy array
2. Is 2D
3. Has 3 rows
4. Has 1 column

Parameters **matrix** – translation matrix

Raises TypeError, ValueError if not

Returns True

3.2 scikit-surgeryvtk

3.2.1 Custom QtVTK Widgets

Overlay Widget

Module to provide a VTK scene on top of a video stream, thereby enabling a basic augmented reality viewer.

Expected usage:

```
window = VTKOverlayWindow()
window.add_vtk_models(list)        # list of VTK models
window.add_vtk_actor(actor)        # or individual actor
window.set_camera_matrix(ndarray)  # Set 3x3 ndarray of camera matrix

while True:

    image = # acquire np.ndarray image some how
    window.set_video_image(image)

    window.set_camera_pose(camera_to_world) # set 4x4 ndarray
```

class sksurgeryvtk.widgets.vtk_overlay_window.**VTKOverlayWindow**(*offscreen=False*, *camera_matrix=None*, *clipping_range=(1, 1000)*, *zbuffer=False*, *opencv_style=True*, *init_pose=False*, *reset_camera=True*)

Bases: sksurgeryvtk.widgets.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor

Sets up a VTK Overlay Window that can be used to overlay multiple VTK models on a video stream. Internally, the Window has 3 renderers. The background renderer displays the video image in the background. The foreground renderer displays a VTK scene overlaid on the background. If you make your VTK models semi-transparent you get a merging effect. An additional rendering layer is just for overlays like picture-in-picture ultrasound.

Parameters

- **offscreen** – Enable/Disable offscreen rendering.
- **camera_matrix** – Camera extrinsics matrix.
- **clipping_range** – Near/Far clipping range.
- **zbuffer** – if True, will only render zbuffer of main renderer.
- **opencv_style** – If True, adopts OpenCV convention, otherwise OpenGL.
- **init_pose** – If True, will initialise the camera pose to identity.
- **reset_camera** – If True, resets camera when a new model is added.

add_vtk_actor(*actor*, *layer=1*)

Add a vtkActor directly.

Parameters

- **actor** – vtkActor
- **layer** – Render layer to add to, defualt 1(foreground)

add_vtk_models(*models*, *layer=1*)
Add VTK models to a renderer. Here, a 'VTK model' is any object that has an attribute called actor that is a vtkActor.

Parameters

- **models** – list of VTK models.
- **layer** – Render layer to add to, default 1 (forground)

convert_scene_to_numpy_array()
Convert the current window view to a numpy array.

Return output Scene as numpy array

get_camera_state()
Get all the necessary variables to allow the camera view to be restored.

get_foreground_camera()
Returns the camera for the foreground renderer.

Returns vtkCamera

get_foreground_renderer()
Returns the foreground vtkRenderer.

Returns vtkRenderer

resizeEvent(*ev*)
Ensures that when the window is resized, the background renderer will correctly reposition the camera such that the image fully fills the screen, and if the foreground renderer is calibrated, also updates the projection matrix.

Parameters **ev** – Event

save_scene_to_file(*file_name*)
Save's the current screen to file. VTK works in RGB, but OpenCV assumes BGR, so swap the colour space before saving to file. :param file_name: must be compatible with cv2.imwrite()

set_camera_matrix(*camera_matrix*)
Sets the camera projection matrix from a numpy 3x3 array. :param camera_matrix: numpy 3x3 ndarray containing fx, fy, cx, cy

set_camera_pose(*camera_to_world*)
Sets the camera position and orientation, from a numpy 4x4 array. :param camera_to_world: camera_to_world transform.

set_camera_state(*camera_properties*)
Set the camera properties to a particular view poisition/angle etc.

set_foreground_camera(*camera*)
Set the foreground camera to track the view in another window.

set_screen(*screen*)
Link the widget with a particular screen. This is necessary when we have multi-monitor setups.

Parameters **screen** – QScreen object.

set_stereo_left()
Set the render window to left stereo view.

set_stereo_right()
Set the render window to right stereo view.

set_video_image(*input_image*)
Set the video image that is used for the background.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

Stereo interlaced Widget

Module to provide an interlaced stereo window, designed for driving things like the Storz 3D laparoscope monitor.

class sksurgeryvtk.widgets.vtk_interlaced_stereo_window.**VTKStereoInterlacedWindow**(*offscreen=Fa left_camera_ right_camer clip- ping_range= 10000)*)

Bases: PySide2.QtWidgets.QWidget

Class to contain a pair of VTKOverlayWindows, stacked with a QLabel widget containing the resulting interlaced picture.

add_vtk_actor(*actor*)
Adds a vtkActor to both left and right widgets.

Parameters **actor** – vtkActor

add_vtk_models(*models*)
Add models to both left and right widgets. Here a model is anything with an attribute called actor that is a vtkActor.

Parameters **models** – vtk_base_model

paintEvent(*ev*)
Ensure that the interlaced image is recomputed.

render()
Calls Render on all 3 contained vtk_overlay_windows.

resizeEvent(*ev*)
Ensure that the interlaced image is recomputed.

save_scene_to_file(*file_name*)
Writes the currently displayed widget contents to file.

Parameters **file_name** – file name compatible with cv2.imwrite()

set_camera_matrices(*left_camera_matrix*, *right_camera_matrix*)
Sets both the left and right camera matrices.

Parameters

- **left_camera_matrix** – numpy 3x3 ndarray containing fx, fy, cx, cy
- **right_camera_matrix** – numpy 3x3 ndarray containing fx, fy, cx, cy

set_camera_poses(*left_camera_to_world*)
Sets the pose of both the left and right camera. If you haven't set the left_to_right transform, it will be identity.

Parameters **left_camera_to_world** – 4x4 numpy ndarray, rigid transform

set_current_viewer_index(*viewer_index*)
Sets the current viewer selection. Defaults to self.default_viewer_ndex.

0 = left 1 = right 2 = interlaced 3 = stacked

Parameters **viewer_index** – index of viewer, as above.

set_left_to_right(*left_to_right*)
Sets the left_to_right transform (stereo extrinsics).

Parameters **left_to_right** – 4x4 numpy ndarray, rigid transform

set_video_images(*left_image*, *right_image*)
Sets both left and right video images. Images must be the same shape, and have an even number of rows.

Parameters

- **left_image** – left numpy image
- **right_image** – right numpy image

:raises ValueError, TypeError

set_view_to_interlaced()
Sets the current view to interlaced.

set_view_to_stacked()
Sets the current view to stacked.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

Rendering Generator

Module to provide a basic VTK render window for test data generation.

class sksurgeryvtk.widgets.vtk_rendering_generator.**VTKRenderingGenerator**(*models_file*, *background_image*, *intrinsic_file*, *camera_to_world=None*, *left_to_right=None*, *offscreen=False*, *zbuffer=False*, *gaussian_sigma=0.0*, *gaussian_window_size=11*, *clipping_range=(1, 1000)*)

Bases: PySide2.QtWidgets.QWidget

Class contains a VTKOverlayWindow and a few extra functions to facilitate rendering loops for generating test data.

Parameters

- **models_file** – JSON file describing VTK models, in SNAPPY format
- **background_image** – RGB image to render in background
- **intrinsic_file** – [3x3] matrix in text file, in numpy format
- **camera_to_world** – list of [rx,ry,rz,tx,ty,tz] in degrees/millimetres
- **left_to_right** – list of [rx,ry,rz,tx,ty,tz] in degrees/millimetres
- **offscreen** – if true, renders offscreen
- **zbuffer** – if true, causes VTK to render just the z-buffer
- **gaussian_sigma** – if non-zero, adds blurring to the rendered image
- **gaussian_window_size** – window size of OpenCV Gaussian kernel
- **clipping_range** – VTK clipping range (near, far)

get_image()
Returns the rendered image, with post processing like smoothing. :return: numpy ndarray representing rendered image (RGB)

get_masks()
If we want to render masks for test data for DL models for instance, we typically want distinct masks per model object. This method returns a dictionary of new images corresponding to each named model.

If model is shaded, the shading is turned off to get masks, the masks are acquired, and the shading is applied again.

Note: You should ensure self.gaussian_sigma == 0 (the default), and in the .json file.

set_all_model_to_world(*model_to_world*)
Decomposes the model_to_world string into rx,ry,rx,tx,ty,rz, constructs a 4x4 matrix, and applies it to all models.

Parameters **model_to_world** – [4x4] numpy ndarray, rigid transform

set_clipping_range(*minimum*, *maximum*)
Sets the clipping range on the foreground camera.

Parameters

- **minimum** – minimum in millimetres
- **maximum** – maximum in millimetres

set_model_to_worlds(*dict_of_transforms*)
Given a dictionary of transforms, will iterate by name, and apply the transform to the named object. :param dict_of_transforms: {name, [rx, ry, rz, tx, ty, tz]}

set_smoothing(*sigma*, *window_size*)
Sets the Gaussian blur.

Parameters

- **sigma** – standard deviation of Gaussian function.
- **window_size** – sets the window size of Gaussian kernel (pixels).

setup_camera_extrinsics(*camera_to_world*, *left_to_right=None*)
Decomposes parameter strings into 6DOF parameters, and sets up camera-to-world and left_to_right for stereo.

Parameters

- **camera_to_world** – list of [rx,ry,rz,tx,ty,tz] in degrees/mm
- **left_to_right** – list of [rx,ry,rz,tx,ty,tz] in degrees/mm

setup_intrinsics()
Set the intrinsics of the foreground vtkCamera.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

Reslice Widget

Module to show slice views of volumetric data.

class sksurgeryvtk.widgets.vtk_reslice_widget.**MouseWheelSliceViewer**(*input_data*)
Bases: *sksurgeryvtk.widgets.vtk_reslice_widget.VTKSliceViewer*

Orthogonal slice viewer using mouse wheel to control slice position.

Example usage:

qApp = QtWidgets.QApplication([]) input_data = 'tests/data/dicom/LegoPhantom_10slices'

slice_viewer = MouseWheelSliceViewer(input_data) slice_viewer.start() qApp.exec_()

start()
Start a timer which will update the 3D view.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_fourth_panel()
Update 3D view.

class sksurgeryvtk.widgets.vtk_reslice_widget.**TrackedSliceViewer**(*input_data*, *tracker*)
Bases: *sksurgeryvtk.widgets.vtk_reslice_widget.VTKSliceViewer*

Orthogonal slice viewer combined with tracker to control slice position. :param input_data: Path to file/folder containing volume data :param tracker: scikit-surgery tracker object,

used to control slice positions.

Example usage:

qApp = QtWidgets.QApplication([]) input_data = 'tests/data/dicom/LegoPhantom_10slices' tracker = ArUcoTracker()

slice_viewer = MouseWheelSliceViewer(input_data, tracker) slice_viewer.start() qApp.exec_()

start()
Show the overlay widget and set a timer running

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_position()
Get position from tracker and use this to set slice positions.

class sksurgeryvtk.widgets.vtk_reslice_widget.**VTKResliceWidget**(*reader*, *axis*, *parent*)
Bases: sksurgeryvtk.widgets.QVTKRenderWindowInteractor.QVTKRenderWindowInteractor

Widget to show a single slice of Volumetric Data. :param reader: vtkReader class e.g. DICOM/Niftii/gipl :param axis: x/y/z axis selection :param parent: parent QWidget.

get_slice_position()
Return the current slice position.

on_mouse_wheel_backward(*obj*, *event*)
Callback to change slice position using mouse wheel.

on_mouse_wheel_forward(*obj*, *event*)
Callback to change slice position using mouse wheel.

reset_position()
Set slice position to the middle of the axis.

set_lookup_table_min_max(*min*, *max*)
Set the minimum/maximum values for the VTK lookup table i.e. change displayed range of intensity values.

set_mouse_wheel_callbacks()
Add callbacks for scroll events.

set_slice_position_mm(*pos*)
Set the slice position in the volume in mm

set_slice_position_pixels(*pos*)
Set the slice position in the volume in pixels

staticMetaObject = <PySide2.QtCore.QMetaObject object>

class sksurgeryvtk.widgets.vtk_reslice_widget.**VTKSliceViewer**(*input_data*)
Bases: PySide2.QtWidgets.QWidget

Othrogonal slice viewer showing Axial/Sagittal/Coronal views :param input_data: path to volume data

reset_slice_positions()
Set slcie positions to some default values.

set_lookup_table_min_max(*min*, *max*)
Set lookup table min/max for all slice views

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_slice_positions_mm(*x_pos*, *y_pos*, *z_pos*)
Set the slice positions for each view. :param x: slice 1 position :param y: slice 2 position :param z: slice 3 position

update_slice_positions_pixels(*x_pos*, *y_pos*, *z_pos*)
Set the slice positions for each view. :param x: slice 1 position :param y: slice 2 position :param z: slice 3 position

3.2.2 VTK Model Data

Base Model

Base class to provide a base class definition of what a 'VTK model' is. In the context of this project, at this current moment in time, its an object that has a member variable called 'actor' that is a vtkActor.

class sksurgeryvtk.models.vtk_base_model.**VTKBaseModel**(*colour*, *visibility=True*, *opacity=1.0*, *pickable=True*)
Bases: object

Defines a base class for 'VTK Models' which are objects that contain a vtkActor. This class enables you to set the colour, visibility and opacity. Note that this colour property is set on the actor. It is possible for various VTK implementations to ignore this. For example a point set could store an RGB tuple for each point, so when

rendered, the overall colour property is effectively ignored. However, the property has been kept at this base class level for simplicity.

get_colour()
Returns the current colour of the model.

Returns R, G, B where each are floats [0-1]

get_name()
Returns the name of the model.

Returns str, the name, which can be None if not yet set.

get_pickable()
Returns the pickable flag.

get_user_matrix()
Getter for vtkActor UserMatrix. :return: vtkMatrix4x4

get_visibility()
Returns bool, True if Actor is visible and False otherwise. :return: bool

set_colour(*colour*)
Set the colour of the model.

Parameters **colour** – (R,G,B) where each are floats [0-1]

:raises TypeError if R,G,B not float, ValueError if outside range.

set_name(*name*)
Sets the name.

Parameters **name** – str containing a name

Raises TypeError if not string, ValueError if empty

set_opacity(*opacity*)
Set the opacity.

Parameters **opacity** – [0-1] float between 0 and 1.

Raises TypeError if not a float, ValueError if outside range.

set_pickable(*pickable*)
Enables the user to set the pickable flag.

Parameters **pickable** –

Raises TypeError if not a boolean

set_user_matrix(*matrix*)
Sets the vtkActor UserMatrix. This simply tells the graphics pipeline to move/translate/rotate the actor. It does not transform the original data.

Parameters **matrix** – vtkMatrix4x4

set_visibility(*visibility*)
Sets the visibility.

Parameters **visibility** – [True|False]

Raises TypeError if not a boolean

toggle_visibility()
Toggles model visbility on/off.

Surface Models

VTK pipeline to represent a surface model via a vtkPolyData.

class sksurgeryvtk.models.vtk_surface_model.**VTKSurfaceModel**(*filename*, *colour*, *visibility=True*, *opacity=1.0*, *pickable=True*)

Bases: *sksurgeryvtk.models.vtk_base_model.VTKBaseModel*

Class to represent a VTK surface model. Normally read from a file, but could be created on the fly.

get_model_transform()
Gets the model to world transform. :return: vtkMatrix4x4

get_no_shading()
Returns whether or not this model is rendered with or without shading. :return: bool

get_normals_as_numpy()

Returns the vtkPolyData point normals as a numpy array.

Returns nx3 numpy ndarray

get_number_of_points()
Returns the number of points in the vtkPoylData. :return: unsigned int

get_points_as_numpy()
Returns the vtkPolyData points as a numpy array. :return: nx3 numpy ndarray

get_source_file()
Returns the filename that the model was loaded from, or empty string if the VTKSurfaceModel was not made from a file.

:return:str filename

get_vtk_source_data() → vtkmodules.vtkCommonDataModel.vtkPolyData
Return original vtk poly data for this object

Returns vtkPolyData

Return type vtk.vtkPolyData

set_model_transform(*matrix*)
Sets the model to world transform onto a vtkPolyDataFilter. This enables all the points and point data to be transformed according to a vtkMatrix4x4 similarity transform.

Parameters **matrix** – vtkMatrix4x4

set_no_shading(*no_shading: bool*)
Turns off/on all shading, so you can generate masks, with solid blocks of colour. Note: Even though I'm tempted to call this flat shading, you can't because flat shading is something different. So we have to call it "no shading".

Parameters **no_shading** – if true, outputs solid blocks of colour

set_texture(*filename*)
Sets an image from a file as a texture for the model. :param filename: :return:

Module to load VTK surfaces using dictionary from ConfigurationManager.

class sksurgeryvtk.models.surface_model_loader.**SurfaceModelLoader**(*data*, *directory_prefix=None*)

Bases: object

Class to load VTK surface models and (optionally) associate them with vtkAssembly's. Surfaces should be defined in a .json file and loaded for example using sksurgerycore.ConfigurationManager.

Surfaces have format:

Assemblies have format:

get_assembly(*name*)
Fetches a vtkAssembly using the name.

Parameters **name** – name of the assembly, as string

Returns vtkAssembly

get_assembly_names()
Returns the set of valid assembly names.

Returns keys from self.named_assemblies

get_surface_model(*name*)
Fetches a VTKSurfaceModel using the name.

Parameters **name** – name of the model

Returns VTKSurfaceModel

get_surface_model_names()
Returns the set of valid model names.

Returns keys from self.named_surfaces

get_surface_models()
Convenience method, to get all models.

Useful for unit testing for example.

Returns list of VTKSurfaceModel

Image Model

VTK pipeline to represent an image with a vtkImageActor.

class sksurgeryvtk.models.vtk_image_model.**VTKImageModel**(*filename*, *visibility=True*, *opacity=1.0*)
Bases: *sksurgeryvtk.models.vtk_base_model.VTKBaseModel*

Class to represent a VTK image model. Normally read from a file, but could be created on the fly.

Point Model

VTK pipeline to represent a point model via a vtkPolyData with a separate (RGB) component for each point, such that each point is rendered with the correct colour. Note that this model is designed to have a fixed number of points. If you want varying number of points for each render pass, you should consider another way of doing this.

class sksurgeryvtk.models.vtk_point_model.**VTKPointModel**(*points*, *colours*, *visibility=True*, *opacity=1.0*)
Bases: *sksurgeryvtk.models.vtk_base_model.VTKBaseModel*

Class to represent a VTK point model. Note, that if

get_number_of_points()
Returns the number of points (hence vertices) in the model. :return: number of points

get_point_size()
Returns the current point size in pixels. :return: size

set_point_size(*size*)
Sets the size of each point in pixels.

Geometric Primitives

VTK pipeline to represent a set of points, as sphere glyphs.

class sksurgeryvtk.models.vtk_sphere_model.**VTKSphereModel**(*points*, *radius*, *colour=(1.0, 1.0, 1.0)*, *visibility=True*, *opacity=1.0*, *pickable=True*, *resolution=12*)

Bases: *sksurgeryvtk.models.vtk_base_model.VTKBaseModel*

Class to represent a set of points as sphere glyphs (one sphere per point).

VTK pipeline to represent a surface model via a vtkPolyData.

class sksurgeryvtk.models.vtk_cylinder_model.**VTKCylinderModel**(*height=10.0*, *radius=3.0*, *colour=(1.0, 0.0, 0.0)*, *name='cylinder'*, *angle=90.0*, *orientation=(1.0, 0.0, 0.0)*, *resolution=88*, *visibility=True*, *opacity=1.0*)

Bases: *sksurgeryvtk.models.vtk_surface_model.VTKSurfaceModel*

Class to create a VTK surface model of a cylinder.

3.2.3 Camera Utilities

Functions to setup a VTK camera to match the OpenCV calibrated camera.

sksurgeryvtk.camera.vtk_camera_model.**compute_projection_matrix**(*width*, *height*, *f_x*, *f_y*, *c_x*, *c_y*, *near*, *far*)

Computes the OpenGL projection matrix.

Thanks to: Andrew Straw.

whose method was also implemented in: NifTK.

Note: If you use this method, the display will look ok, but as of VTK 8.1.0, it won't work with vtkWindowToImageFilter, as the window to image filter tries to render the image in tiles. This requires instantiating temporary new vtkCamera, and the vtkCamera copy constructor, shallow copy and deep copy do not actually copy the UseExplicitProjectionTransformMatrixOn or ExplicitProjectionTransformMatrix.

Parameters

- **width** – image width in pixels
- **height** – image height in pixels

- **f_x** – focal length in x direction, (K_00)
- **f_y** – focal length in y direction, (K_11)
- **c_x** – principal point x coordinate, (K_02)
- **c_y** – principal point y coordinate, (K_12)
- **near** – near clipping distance in world coordinate frame units (mm)
- **far** – far clipping distance in world coordinate frame units (mm)

Returns vtkMatrix4x4 containing a 4x4 projection matrix

sksurgeryvtk.camera.vtk_camera_model.**compute_right_camera_pose**(*left_camera_to_world*, *left_to_right*)

Returns the right_camera_to_world, computed from the combination of left_camera_to_world, and left_to_right.

Parameters

- **left_camera_to_world** – 4x4 numpy ndarray representing rigid transform
- **left_to_right** – 4x4 numpy ndarray representing rigid transform

Returns right_camera_to_world as 4x4 numpy ndarray

sksurgeryvtk.camera.vtk_camera_model.**compute_scissor**(*window_width*, *window_height*, *image_width*, *image_height*, *aspect_ratio*)

Used on vtkCamera when you are trying to set the viewport to only render to a part of the total window size. For example, this occurs when you have calibrated a video camera using OpenCV, on images of 1920 x 1080, and then you are displaying in a VTK window that is twice as wide/high.

This was implemented in: NifTK.

and it appears it should also be available in: VTK.

Parameters

- **window_width** – in pixels
- **window_height** – in pixels
- **image_width** – in pixels
- **image_height** – in pixels
- **aspect_ratio** – relative physical size of pixels, as x/y.

Returns scissor_x, scissor_y, scissor_width, scissor_height in pixels

sksurgeryvtk.camera.vtk_camera_model.**compute_viewport**(*window_width*, *window_height*, *scissor_x*, *scissor_y*, *scissor_width*, *scissor_height*)

Used on vtkCamera when you are trying to set the viewport to only render to a part of the total window size. For example, this occurs when you have calibrated a video camera using OpenCV, on images of 1920 x 1080, and then you are displaying in a VTK window that is twice as wide/high.

Parameters

- **window_width** – in pixels
- **window_height** – in pixels
- **scissor_x** – output from compute_scissor

- **scissor_y** – output from compute_scissor
- **scissor_width** – output from compute_scissor
- **scissor_height** – output from compute_scissor

Returns x_min, y_min, x_max, y_max as normalised viewport coordinates

sksurgeryvtk.camera.vtk_camera_model.**set_camera_intrinsics**(*vtk_renderer*, *vtk_camera*, *width*, *height*, *f_x*, *f_y*, *c_x*, *c_y*, *near*, *far*)

Used to setup a vtkCamera according to OpenCV conventions.

Thanks to: benoitrosa

Parameters

- **vtk_renderer** – vtkRenderer
- **vtk_camera** – vtkCamera
- **width** – image width in pixels
- **height** – image height in pixels
- **f_x** – focal length in x direction, (K_00)
- **f_y** – focal length in y direction, (K_11)
- **c_x** – principal point x coordinate, (K_02)
- **c_y** – principal point y coordinate, (K_12)
- **near** – near clipping distance in world coordinate frame units (mm).
- **far** – far clipping distance in world coordinate frame units (mm).

sksurgeryvtk.camera.vtk_camera_model.**set_camera_pose**(*vtk_camera*, *vtk_matrix*, *opencv_style=True*)

Sets the camera position and orientation from a camera to world matrix.

If opencv_style is False, the camera defaults to the origin, facing along the -z axis, with +y being up.

If opencv_style is True (default for legacy compatibility), the camera defaults to the origin, facing along the +z axis, with +y being down. This is more in-line with Opencv. So, if you are calibrating with OpenCV, and want to use those extrinsic matrices to set the pose, then you want this option.

Parameters

- **vtk_camera** – a vtkCamera
- **vtk_matrix** – a vtkMatrix4x4 representing the camera to world.
- **opencv_style** – If True uses OpenCV (+z), otherwise OpenGL (-z)

3.2.4 Text Overlay

Classes to implement text overlay. Includes Corner Annotation, Large Centered Text and generic text overlay.

class sksurgeryvtk.text.text_overlay.**VTKCornerAnnotation**

Bases: object

Wrapper for vtkCornerAnnotaiton class.

get_text()
Returns the current list of text annotations :return: [bottom-left, bottom-right, top-left, top-right]

set_text(*text_list*)
Set the text in each of the four corners

Parameters **text_list** (*List of 4 strings.*) – Text to display. [bottom-left, bottom-right, top-left, top-right].

set_text_on_bottom_left(*text*)
Set the text on the bottom-left corner.

Parameters **text** – Text to display.

set_text_on_bottom_right(*text*)
Set the text on the bottom-right corner.

Parameters **text** – Text to display.

set_text_on_top_left(*text*)
Set the text on the top-left corner.

Parameters **text** – Text to display.

set_text_on_top_right(*text*)
Set the text on the top-right corner.

Parameters **text** – Text to display.

validate_input(*text_list*)
Check that the text_list input is a list of four strings.

Parameters **text_list** – input to check.

class sksurgeryvtk.text.text_overlay.**VTKLargeTextCentreOfScreen**(*text*)
Bases: *sksurgeryvtk.text.text_overlay.VTKTextBase*

Display large text in the centre of the screen. Useful for error messages/warnings etc.

Parameters **text** – text to display.

calculate_text_size(*_obj_unused*, *_ev_unused*)
Calculate the position and size of the text. Text should span the central half (x & y) of the window.

set_parent_window(*parent_window*)
Attach text to a particular window. :param parent_window: VTKOverlayWindow that message will

be displayed in.

class sksurgeryvtk.text.text_overlay.**VTKText**(*text*, *x*, *y*, *font_size=24*, *colour=(1.0*, *0*, *0)*)
Bases: *sksurgeryvtk.text.text_overlay.VTKTextBase*

VTKText object that can be placed following a left click event. Text will rescale if the window resizes, to try and maintain relative positioning.

Parameters

- **text** – text to display.
- **x** – x position (pixels)
- **y** – y position (pixels)
- **font_size** – Font size

param colour: Colour, RGB tuple

calculate_relative_position_in_window()
Calculate position relative to the middle of the screen. Can then be used to re-set the position if the window is resized.

callback_update_position_in_window(*_obj_unused*, *_ev_unused*)
Update position, maintaing relative distance to the centre of the background image.

set_parent_window(*parent_window*)
Link the object to a VTKOverlayWindow and set up callbacks. :param parent_window: VTKOverlayWindow

class sksurgeryvtk.text.text_overlay.**VTKTextBase**
Bases: object

Wrapper around vtkTextActor class to set position, colour, size etc.

set_colour(*r*, *g*, *b*)
Set the text colour. :param r: Red (0.0 - 1.0) :param g: Green (0.0 - 1.0) :param b: Blue (0.0 - 1.0)

set_font_size(*size*)
Set the font size. :param size: size in points

set_text_position(*x*, *y*)
Set the x,y coordinates of the text (bottom-left) :param x: x location in pixels :param y: y locaiton in pixels

set_text_string(*text*)
Set the text string. :param text: text to display.

validate_text_input(*text*)
Check text input is a valid string. :param text: Input to validate.

validate_x_y_inputs(*x*, *y*)
Check that coordinate inputs are valid. :param x: x location. :param y: y location

3.2.5 Misc Utilities

Matrix Utilities

Any useful little utilities to do with matrices.

sksurgeryvtk.utils.matrix_utils.**calculate_l2r_matrix**(*left_extrinsics: numpy.ndarray*, *right_extrinsics: numpy.ndarray*) → numpy.ndarray

Return the left to right transformation matrix: l2r = R * L^-1

sksurgeryvtk.utils.matrix_utils.**create_matrix_from_list**(*params*, *is_in_radians=False*)
Generates a 4x4 numpy ndarray from a list of rx,ry,rz,tx,ty,tz in degrees, millimetres.

This is designed to match VTK. VTK states that vtkProp3D uses 'Orientation is specified as X,Y and Z rotations in that order, but they are performed as RotateZ, RotateX, and finally RotateY. However vtkTransform by default uses pre-multiplication. So, in mathematical notation, this would be written as

[Output Point] = [RotateZ][RotateX][RotateY][Input Point]

which, if you read the maths expression from right to left, would actually be termed RotateY, then RotateX, then RotateZ.

The function in scikit-surgerycore called construct_rotm_from_euler takes an input string, e.g. 'zxy' and follows mathematical notation. So, 'zxy' means RotateY, RotateX, RotateZ in that order, reading from right to left, and so matches VTK.

Furthermore, the construct_rotm_from_euler function in scikit-surgerycore expectes the user to pass the parameters in, in the order specified in the provided string.

:param params list of exactly 6 numbers. :param is_in_radians True if radians, False otherwise, default is False

sksurgeryvtk.utils.matrix_utils.**create_matrix_from_string**(*parameter_string*, *is_in_radians=False*)

Generates a 4x4 numpy ndarray from a comma separated string of the format rx,ry,rz,tx,ty,tz in degrees, millimetres.

Parameters **parameter_string** – rx,ry,rz,tx,ty,tz in degrees/millimetres

:param is_in_radians True if radians, False otherwise, default is False :return: 4x4 rigid body transform

sksurgeryvtk.utils.matrix_utils.**create_numpy_matrix_from_vtk**(*matrix*)

Returns a new numpy 4x4 matrix from a vtkMatrix4x4.

sksurgeryvtk.utils.matrix_utils.**create_vtk_matrix_from_numpy**(*array*)

Return a new vtkMatrix4x4 from a numpy array.

sksurgeryvtk.utils.matrix_utils.**get_l2r_smartliver_format**(*l2r_matrix: numpy.ndarray*) → numpy.ndarray

Convert 4x4 left to right matrix to smartliver l2r format:

R1 R2 R3 R4 R5 R6 R7 R8 R9 T1 T2 T3

sksurgeryvtk.utils.matrix_utils.**validate_vtk_matrix_4x4**(*matrix*)

Checks that a matrix is a vtkMatrix4x4. :param matrix: vtkMatrix4x4 :raises TypeError :return: True

Projection Utilities

Any useful little utilities to do with projecting 3D to 2D.

sksurgeryvtk.utils.projection_utils.**compute_rms_error**(*model_points*, *image_points*, *renderer*, *scale_x*, *scale_y*, *image_height*)

Mainly for unit testing. Computes rms error between projected model points, and image points.

Parameters

- **model_points** – nx3 numpy array of 3D points
- **image_points** – nx2 numpy array of 2D expected points
- **renderer** – vtkRenderer
- **scale_x** – scale factor for x
- **scale_y** – scale factor for y
- **image_height** – image height

sksurgeryvtk.utils.projection_utils.**project_facing_points**(*points*, *normals*, *camera_to_world*, *camera_matrix*, *distortion=None*, *upper_cos_theta=0*)

Projects 3D points that face the camera to 2D pixels.

This assumes:

Camera direction is a unit vector from the camera, towards focal point. Surface Normal is a unit vector pointing out from the surface.

Vectors are not checked for unit length.

Parameters

- **points** – nx3 ndarray representing 3D points, typically in millimetres
- **normals** – nx3 ndarray representing unit normals for the same points
- **camera_to_world** – 4x4 ndarray representing camera to world transform
- **camera_matrix** – 3x3 ndarray representing OpenCV camera intrinsics
- **distortion** – 1x4,5 etc. OpenCV distortion parameters
- **upper_cos_theta** – upper limit for cos theta, angle between normal

and viewing direction, where cos theta is normally -1 to 0. :raises ValueError, TypeError: :return: projected_facing_points_2d

sksurgeryvtk.utils.projection_utils.**project_points**(*points*, *camera_to_world*, *camera_matrix*, *distortion=None*)

Projects all 3D points to 2D, using OpenCV cv2.projectPoints().

Parameters

- **points** – nx3 ndarray representing 3D points, typically in millimetres
- **camera_to_world** – 4x4 ndarray representing camera to world transform
- **camera_matrix** – 3x3 ndarray representing OpenCV camera intrinsics
- **distortion** – 1x4,5 etc. OpenCV distortion parameters

Raises **ValueError, TypeError** –

Returns nx2 ndarray representing 2D points, typically in pixels

Polydata Utilities

Utilities for operations on vtk polydata

sksurgeryvtk.utils.polydata_utils.**check_overlapping_bounds**(*polydata_0*, *polydata_1*)

Checks whether two polydata have overlapping bounds

Parameters

- **polydata_0** – vtkPolyData representing a 3D mesh
- **polydata_1** – vtkPolyData representing a 3D mesh

:return : True if bounding boxes overlap, False otherwise

sksurgeryvtk.utils.polydata_utils.**two_polydata_dice**(*polydata_0*, *polydata_1*)

Calculates the DICE score for two polydata. Will probably struggle with complex topologies, but should be fine for vaguely spherical shape. This function uses vtk.vtkMassProperties() so does not convert polydata to image data

Parameters

- **polydata_0** – vtkPolyData representing a 3D mesh

- **polydata_1** – vtkPolyData representing a 3D mesh

Return dice The DICE score

Return volume_0 The enclosed volume of polydata_0

Return volume_1 The enclosed volume of polydata_1

Return volume_01 The enclosed volume of the intersection

3.2.6 Voxelisation & Distance Fields

Re-impelmentaiton of voxeilsation code from https://gitlab.com/nct_tso_public/Volume2SurfaceCNN

sksurgeryvtk.models.voxelise.**applyTransformation**(*dataset*, *tf*)

Apply a transformation to each data array stored in vtk object.

Parameters

- **dataset** – Vtk object containing array(s)
- **tf** (*vtk.vtkTransform*) – Transform

sksurgeryvtk.models.voxelise.**apply_displacement_to_mesh**(*mesh: Union[vtkmodules.vtkCommonDataModel.vtkData str], field: Union[vtkmodules.vtkCommonDataModel.vtkStruc str], save_mesh: Union[bool, str] = False, disp_array_name: str = 'estimatedDisplacement'*)

Apply a displacement field to a mesh. The displacement field is stored as an array within a vtkStructuredGrid.

Parameters

- **mesh** (*Union[vtk.vtkDataObject, str]*) – Mesh to deform, can either be path to file or vtk object.
- **field** (*Union[vtk.vtkStructuredGrid, str]*) – Grid containing displacement field, can either be path to file or vtk object.
- **save_mesh** (*Union[bool, str], optional*) – If a file name is passed, the deformed mesh is saved to disk, defaults to False
- **disp_array_name** (*str, optional*) – Name of array within vtkStructuredGrid containing the displacement field, defaults to 'estimatedDisplacement'

Returns Displaced mesh

Return type vtk.vtkPolyData

sksurgeryvtk.models.voxelise.**createGrid**(*total_size: float*, *grid_elements: int*)

Returns a vtkStrucutredGrid.

Parameters

- **total_size** – Total size of the grid i.e. How long is each dimension. Each indivdual element has size equal to total_size/grid_dims
- **grid_dims** (*int*) – Number of grid points in x/y/z

Returns grid

Return type vtkStructuredGrid

sksurgeryvtk.models.voxelise.**distanceField**(*surfaceMesh*, *targetGrid*, *targetArrayName: str*, *signed=False*)

Create a distance field between a vtkStructuredGrid and a surface.

Parameters

- **surfaceMesh** – Outer polygonal surface
- **targetGrid** (*vtk.vtkStructuredGrid*) – Grid array of points
- **targetArrayName** (*str*) – The distance field values will be stored in the target grid, with this array name.
- **signed** (*bool, optional*) – Signed/unsigned distance field, defaults to False (unsigned)

sksurgeryvtk.models.voxelise.**distanceFieldFromCloud**(*surfaceCloud*, *targetGrid*, *targetArrayName*)

Create a distance field between a vtkStructuredGrid and a point cloud.

Parameters

- **surfaceMesh** – Pointcloud of surface
- **targetGrid** (*vtk.vtkStructuredGrid*) – Grid array of points
- **targetArrayName** – The distance field values will be stored in the target grid, with this array name.

sksurgeryvtk.models.voxelise.**extractSurface**(*inputMesh*)

Extract surface of a mesh.

sksurgeryvtk.models.voxelise.**extract_array_from_grid**(*input_grid: vtkmodules.vtkCommonDataModel.vtkStructuredGrid*, *array_name: str*) → numpy.ndarray

Read an array from a vtkStructuredGrid object

Parameters

- **input_grid** (*vtk.vtkStructuredGrid*) – Input data grid
- **array_name** (*str*) – Array to extract from grid

Returns Extracted array

Return type np.ndarray

sksurgeryvtk.models.voxelise.**extract_array_from_grid_file**(*input_grid_file: str*, *array_name: str*) → numpy.ndarray

Read an array from vtkStructuredGrid file

Parameters

- **input_grid_file** (*str*) – Input file, should be a vtkStructuredGrid file
- **array_name** (*str*) – Array to extract from grid

Returns Extracted array

Return type np.ndarray

sksurgeryvtk.models.voxelise.**extract_surfaces_for_v2snet**(*input_grid: vtkmodules.vtkCommonDataModel.vtkStructuredGrid*) → Tuple[numpy.ndarray, numpy.ndarray]

Conveience function to extract the pre and intraoperative surfaces, to pass to V2SNet.

Parameters **input_grid** (*vtk.vtkStructuredGrid*) – Grid containing pre and intraoperative surfaces

Returns pre and intraoperative surfaces as numpy arrays

Return type Tuple[np.ndarray, np.ndarray]

sksurgeryvtk.models.voxelise.**loadTransformationMatrix**(*grid*)

Extract a transformation matrix fom a vtk grid array.

sksurgeryvtk.models.voxelise.**load_points_from_file**(*filename*)

Extract vtk mesh from input file. :returns: Vtk mesh.

sksurgeryvtk.models.voxelise.**load_structured_grid**(*input_file: str*)

Load vtkStructuredGrid from file

Parameters **input_file** (*str*) – Path to vtk structured grid file

Raises **TypeError** –

Returns Loaded grid

Return type vtk.vtkStructuredGrid

sksurgeryvtk.models.voxelise.**save_displacement_array_in_grid**(*array: numpy.ndarray, out_grid: Union[vtkmodules.vtkCommonDataModel.v str], array_name: str = 'estimatedDisplacement'*)

Save numpy data as an array within a vtkStructuredGrid. Mainly used for storing calculated displacement field.

Parameters

- **array** (*np.ndarray*) – Numpy array
- **out_grid** (*Union[vtk.vtkStructuredGrid, str]*) – Grid in which to store array
- **array_name** (*str, optional*) – Array name, defaults to "estimatedDisplacement"

sksurgeryvtk.models.voxelise.**storeTransformationMatrix**(*grid*, *tf*)

Store a transformation matrix inside a vtk grid array.

sksurgeryvtk.models.voxelise.**unstructuredGridToPolyData**(*ug*)

Convert vtk unstructured grid to vtk poly data.

sksurgeryvtk.models.voxelise.**voxelise**(*input_mesh: Union[numpy.ndarray, vtkmodules.vtkCommonDataModel.vtkDataObject, str], output_grid: Union[vtkmodules.vtkCommonDataModel.vtkStructuredGrid, str] = None, array_name: str = '', size: float = 0.3, grid_elements: int = 64, move_input: float = None, center: bool = False, scale_input: float = None, reuse_transform: bool = False, signed_df: bool = True*)

Creates a voxelised distance field, stores it in a vtkStructuredGrid, optinally writes to disk.

Parameters

- **input_mesh** (*Union[np.ndarray, str]*) – Input mesh/points. Can be path to model file, or numpy array. Units of mesh should be in metres.
- **output_grid** – Either a vtkStrucutredGrid object, or a file that

contains one (or will be created), if not specified, a grid will be created. :type output_grid: Union[vtk.vtkStructuredGrid, str], optional :param array_name: Name of array in which to store distance field, if not specified, defaults to preoperativeSurface for if signed_df = True,

else intraoperativeSurface

Parameters

- **size** (*float, optional*) – Grid size, defaults to 0.3
- **grid_elements** – Number of x/y/z elements in grid, defaults to 64 :type grid_elements: int, optional
- **move_input** (*float, optional*) – Move the input before transforming to distance field (movement is applied before scaling! defaults to None
- **center** (*bool, optional*) – Center the data around the origin. defaults to False
- **scale_input** (*float, optional*) – Scale the input before transforming to distance field (movement is applied before scaling!). Input is expected to be in metres, if it is in mm, set scale_input to 0.001 defaults to None
- **reuse_transform** (*bool, optional*) – Reuse transformation already stored in the grid. Use this if you want to center mesh 1 and then apply the same transformation to mesh 2. Mutually exclusive with center, scale_input and move_input. defaults to False
- **signed_df** (*bool, optional*) – Calcualte signed or unsigned distance field. defaults to True

Return grid Grid containing distance field.

Return type vtk.vtkStructuredGrid

sksurgeryvtk.models.voxelise.**write_grid_to_file**(*grid: vtkmodules.vtkCommonDataModel.vtkStructuredGrid*, *output_grid: str*)

Write vtkStructuredGrid to file

Parameters

- **grid** (*vtk.vtkStructuredGrid*) – Grid to write
- **output_grid** (*str*) – File path

3.3 scikit-surgeryimage

3.3.1 Data Acquisition

Timestamped Video Source

Module for video source acquisition. Classes capture data from a video source into a numpy array.

class sksurgeryimage.acquire.video_source.**TimestampedVideoSource**(*source_num_or_file*, *dims=None*)

Bases: object

Capture and store data from camera/file source. Augments the cv2.VideoCapture() to provide passing of camera dimensions in constructor, and storage of frame data.

grab()
Call the cv2.VideoCapture grab function and get a timestamp.

isOpened()
Call the cv2.VideoCapture isOpened function.

read()
Do a grab(), then retrieve() operation.

release()
Release the cv2.VideoCapture source.

retrieve()
Call the cv2.VideoCapture retrieve function and store the returned frame.

set_resolution(*width: int*, *height: int*)
Set the resolution of the input source.

Parameters

- **width** (*int*) – Width
- **height** (*int*) – Height

Raises **ValueError** – If resolution is not supported.

class sksurgeryimage.acquire.video_source.**VideoSourceWrapper**
Bases: object

Wrapper for multiple TimestampedVideoSource objects.

add_camera(*camera_number*, *dims=None*)
Create VideoCapture object from camera and add it to the list of sources.

Parameters

- **camera_number** – integer camera number
- **dims** – (width, height) as integer numbers of pixels

add_file(*filename*, *dims=None*)
Create videoCapture object from file and add it to the list of sources.

Parameters

- **filename** – a string containing a valid file path
- **dims** – (width, height) as integer numbers of pixels

add_source(*camera_num_or_file*, *dims=None*)

Add a video source (camera or file) to the list of sources.

Parameters

- **camera_num_or_file** – either an integer camera number or filename
- **dims** – (width, height) as integer numbers of pixels

are_all_sources_open()
Check all input sources are active/open.

get_next_frames()
Do a grab() operation for each source, followed by a retrieve().

grab()
Perform a grab() operation for each source

release_all_sources()
Close all camera/file sources.

retrieve()
Perform a retrieve operation for each source. Should only be run after a grab() operation.

:returns list of views on frames

Stereo Video Source

Module for stereo video source acquisition.

class sksurgeryimage.acquire.stereo_video.**StereoVideo**(*layout*, *channels*, *dims=None*)

Bases: object

Provides a convenient object to manage various stereo input styles. Developed firstly for laparoscopic surgery, but broadly applicable to any stereo setup using our TimestampedVideoSource and VideoSourceWrapper.

Design Principles:

1. Fail early, throwing exceptions for all errors.
2. Works with or without camera parameters.
3. If no camera parameters, calling get_undistorted() or get_rectified() is an Error.

get_images()
Returns the 2 channels, unscaled, as a list of images.

Returns list of images

get_rectified()
Returns the 2 channels, rectified, as a list of images.

Returns list of images

Raises ValueError, TypeError - if camera parameters are not set.

get_scaled()
Returns the 2 channels, scaled, as a list of images.

Returns list of images

get_undistorted()
Returns the 2 channels, undistorted, as a list of images.

Returns list of images

Raises ValueError - if you haven't already provided camera parameters

grab()
Asks internal VideoSourceWrapper to grab images.

release()
Asks internal VideoSourceWrapper to release all sources.

retrieve()
Asks internal VideoSourceWrapper to retrieve images.

set_extrinsic_parameters(*rotation*, *translation*, *dims*)
Sets the stereo extrinsic parameters.

Parameters

- **rotation** – 3x3 numpy array representing rotation matrix.
- **translation** – 3x1 numpy array representing translation vector.
- **dims** – new image size for rectification

Raises ValueError, TypeError

set_intrinsic_parameters(*camera_matrices*, *distortion_coefficients*)
Sets both sets of intrinsic parameters.

Parameters

- **camera_matrices** – list of 2, 3x3 numpy arrays.
- **distortion_coefficients** – list of 2, 1xN numpy arrays.

Raises ValueError, TypeError

class sksurgeryimage.acquire.stereo_video.**StereoVideoLayouts**
Bases: object

Class to hold some constants, like an enum.

DUAL = 0

INTERLACED = 1

VERTICAL = 2

Video Writing

Write stream of frames to file using OpenCV

class sksurgeryimage.acquire.video_writer.**ThreadedTimestampedVideoWriter**(*filename*, *fps=25*, *width=640*, *height=480*, *codec='MJPG'*)
Bases: *sksurgeryimage.acquire.video_writer.TimestampedVideoWriter*

TimestampedVideoWriter that can be run in a thread. Uses Queue.Queue() to store data, which is thread safe.

Frames will be processed as they are added to the queue:

threaded_vw = ThreadedTimestampedVideoWriter(file, fps, w, h) threaded_vw.start()

threaded_vw.add_to_queue(frame, timestamp) threaded_vw.add_to_queue(frame, timestamp) threaded_vw.add_to_queue(frame, timestamp)

threaded_vw.stop()

run()
Write data from the queue to the output file(s).

start()
Start the thread running.

stop()
Stop thread running.

write_frame(*frame*, *timestamp=None*)
Add a frame and a timestamp to the queue for writing. Named for consistency with the non-threaded version. Actual writing to disk is done by write_frame_to_disk() :param frame: Image frame :type frame: numpy array :param timestamp: Frame timestamp :type timestamp: datetime.datetime object

write_frame_to_disk()
Get frame and timestamp from queue, then write to output.

class sksurgeryimage.acquire.video_writer.**TimestampedVideoWriter**(*filename*, *fps=25*, *width=640*, *height=480*, *codec='MJPG'*)
Bases: *sksurgeryimage.acquire.video_writer.VideoWriter*

Class to write images and timestamps to disk, inherits from VideoWriter.

Parameters

- **fps** – Frames per second to save to disk.
- **filename** – Filename to save output video to. Timestamp file is "filename + 'timestamps'"

close()
Close/release the output files for video and timestamps.

write_frame(*frame*, *timestamp=None*)
Write a frame and timestamp to the output files. If no timestamp provided, write a defualt value. :param frame: Image data :type frame: numpy array :param timestamp: Timestamp data :type timestamp: datetime.datetime object

class sksurgeryimage.acquire.video_writer.**VideoWriter**(*filename*, *fps=25*, *width=640*, *height=480*, *codec='MJPG'*)
Bases: object

Class to write images to disk using cv2.VideoWriter.

Parameters

- **fps** – Frames per second to save to disk.
- **filename** – Filename to save output video to.
- **width** – width of input frame
- **height** – height of input frame

check_valid_filename(*filename*)
Return true if filename is a string.

close()
Close/release the output file for video.

create_output_dir_if_needed()
Check if the directory specified in file path exists and create if not.

set_filename(*filename*)
Set the filename to write to.

write_frame(*frame*)
Write a frame to the output file.

3.3.2 Calibration Tools

Point Detector

Base class for a PointDetector.

e.g. Chessboard corners, SIFT points, Charuco points etc.

class sksurgeryimage.calibration.point_detector.**PointDetector**(*scale=(1, 1)*, *camera_intrinsics=None*, *distortion_coefficients=None*)

Bases: object

Class to detect points in a 2D video image.

These point detectors are often used to detect points for camera calibration. However, it would also be possible for some subclasses to utilise camera intrinsics and distortion coefficients in order to improve the point detection process itself. It would be up to the derived class to decide how to use them, if at all.

Parameters

- **scale** – tuple (x scale, y scale) to scale up/down the image
- **camera_intrinsics** – [3x3] camera matrix
- **distortion_coefficients** – [1xn] distortion coefficients

get_camera_parameters()

Returns a copy of the camera matrix, and distortion coefficients. Throws RuntimeError if either are None.

Returns [3x3], [1xn] matrices

get_model_points()

Derived classes should override this, to detector returns the complete model of 3D points. e.g. for a chessboard this would be all the corners in chessboard coordinates (e.g. z=0).

By design, this can return an ndarray with zero rows, if the detector does not support 3D coordinates.

Returns [Nx3] numpy ndarray representing model points.

get_points(*image*, *is_distorted=True*)

Client's call this method to extract points from an image.

Parameters

- **image** – numpy 2D RGB/grayscale image.
- **is_distorted** – False if the input image has already been undistorted.

Returns ids, object_points, image_points as Nx[1,3,2] ndarrays

set_camera_parameters(*camera_intrinsics*, *distortion_coefficients*)

Enables camera parameters to be set dynamically at run-time. Calls _validate_camera_parameters().

Parameters

- **camera_intrinsics** – [3x3] camera matrix
- **distortion_coefficients** – [1xn] distortion coefficients

Chessboard Detector

Chessboard implementation of PointDetector.

class sksurgeryimage.calibration.chessboard_point_detector.**ChessboardPointDetector**(*number_of_*
square_size
scale=(1, 1))

Bases: *sksurgeryimage.calibration.point_detector.PointDetector*

Class to detect chessboard points in a 2D grey scale video image.

get_model_points()
Returns a [Nx3] numpy ndarray representing the model points in 3D.

ArUco Point Detector

ArUco implementation of PointDetector.

class sksurgeryimage.calibration.aruco_point_detector.**ArucoPointDetector**(*dictionary*, *parameters*, *model*, *scale=(1, 1)*)

Bases: *sksurgeryimage.calibration.point_detector.PointDetector*

Class to detect ArUco points in a 2D grey scale video image.

Note: For ArUco points, these don't have to be on a regular grid. If you provide a 'model' which is a map of id : 3D point, the function _internal_get_points will provide the corresponding 3D points of those points that were detected.

get_model_points()
Returns a [Nx3] numpy ndarray representing the model points in 3D.

sksurgeryimage.calibration.aruco_point_detector.**get_intersect**(*a_1*, *a_2*, *b_1*, *b_2*)

Returns the point of intersection of the lines passing through a2,a1 and b2,b1.

See https://stackoverflow.com/questions/3252194/numpy-and-line-intersections

Parameters

- **a_1** – [x, y] a point on the first line
- **a_2** – [x, y] another point on the first line
- **b_1** – [x, y] a point on the second line
- **b_2** – [x, y] another point on the second line

ChArUco Point Detector

ChArUco implementation of PointDetector.

class sksurgeryimage.calibration.charuco_point_detector.**CharucoPointDetector**(*dictionary*, *number_of_squares*, *size*, *scale=(1, 1)*, *camera_matrix=None*, *distortion_coefficients=N*, *filtering=False*)

Bases: *sksurgeryimage.calibration.point_detector.PointDetector*

Class to detect ChArUco points in a 2D video image.

get_model_points()

Returns a [Nx3] numpy ndarray representing the model points in 3D.

ChArUco & Chesboard Detector

ChArUco + Chessboard implementation of PointDetector.

class sksurgeryimage.calibration.charuco_plus_chessboard_point_detector.**CharucoPlusChessboa**

Bases: *sksurgeryimage.calibration.point_detector.PointDetector*

Class to detect ChArUco points and Chessboard points in a 2D grey scale video image.

get_model_points()
Returns a [Nx3] numpy ndarray representing the model points in 3D.

Dotty Grid Point Detector

Dotty Grid implementation of PointDetector.

class sksurgeryimage.calibration.dotty_grid_point_detector.**DottyGridPointDetector**(*model_point list_of_index cam- era_intrinsic dis- tor- tion_coefficie scale=(1, 1), ref- er- ence_image_ rms=30, gaus- sian_sigma= thresh- old_window_ thresh- old_offset=2 min_area=5 max_area=5 dot_detector*

Bases: *sksurgeryimage.calibration.point_detector.PointDetector*

Class to detect a grid of dots in a 2D grey scale video image.

More specifically, a grid of dots with 4 larger dots at known locations.

get_model_points()

Returns a [Nx3] numpy ndarray representing the model points in 3D.

sksurgeryimage.calibration.dotty_grid_point_detector.**get_model_points**(*dots_rows_columns: (<class 'int'>, <class 'int'>), pix- els_per_mm: int, dot_separation: float*) → numpy.ndarray

Generate the expected locations of dots in the pattern, in pixel space.

Parameters

- **dots_rows_columns** (*[int, int]*) – Number of rows, number of columns
- **pixels_per_mm** (*int*) – Pixels per mm
- **dot_separation** (*float*) – Distance between dots in mm

Returns array pf point info - [id, x_pix, y_pix, x_mm, y_mm, z_mm]

Return type np.ndarray

ChArUco Helper Functions

Functions to support camera calibration using ChArUco chessboard markers.

sksurgeryimage.calibration.charuco.**detect_charuco_points**(*dictionary*, *board*, *image*, *camera_matrix=None*, *distortion_coefficients=None*, *filtering=False*)

Extracts ChArUco points. If you can provide camera matrices, it may be more accurate.

Parameters

- **dictionary** – aruco dictionary definition
- **board** – aruco board definition
- **image** – grey scale image in which to search
- **camera_matrix** – if specified, the 3x3 camera intrinsic matrix
- **distortion_coefficients** – if specified, the distortion coefficients
- **filtering** – if True, filter out wrongly detected markers

Returns marker_corners, marker_ids, chessboard_corners, chessboard_ids

sksurgeryimage.calibration.charuco.**draw_charuco_corners**(*image*, *chessboard_corners*, *chessboard_ids*)

Function to draw chessboard corners on an image.

Parameters

- **image** – input image
- **chessboard_corners** – from detect_charuco_points
- **chessboard_ids** – from detect_charuco_points

Returns new image with corners marked

sksurgeryimage.calibration.charuco.**erase_charuco_markers**(*image*, *marker_corners*)

Method to automatically blank out ChArUco markers, leaving an image that looks like it contains just a chessboard, rather than ChArUco board. It does this by drawing a plain white polygon, with vertices defined by the tag detection process. So, on a synthetic image, this works perfectly. On a real image, due to blurring or other artefacts such as combing, there may be some residual.

Parameters

- **image** – image containing a view of a ChArUco board.
- **marker_corners** – detected corners

Returns edited image

sksurgeryimage.calibration.charuco.**filter_out_wrong_markers**(*marker_corners*, *marker_ids*, *board*)

Filters out markers that were mis-labelled. For each inner corner on the ChArUco board, if both neighbouring markers are detected, look at the projected positions of this corner using the perspective transformations obtained form the two markers. If the two positions are not close (further than 20 pixels away), then at least one of the markers is mis-labelled but we won't know which one. Remove both markers.

Parameters

- **marker_corners** – marker corners detected by OpenCV

- **marker_ids** – ids of markers detected
- **board** – charuco board definition

Returns marker_corners, marker_ids

sksurgeryimage.calibration.charuco.**make_charuco_board**(*dictionary*, *number_of_squares*, *size*, *image_size*)

Generates a ChArUco pattern.

Don't forget to select an image size that is a nice multiple of the square size in millimetres, to avoid any interpolation artefacts. You should check the resultant image has only 2 values, [0|255], and nothing interpolated between these two numbers.

Parameters

- **dictionary** – aruco dictionary definition
- **number_of_squares** – tuple of (number in x, number in y)
- **size** – tuple of (size of chessboard square, size of internal tag), mm.
- **image_size** – tuple of (image width, image height), pixels.

Returns image, board

sksurgeryimage.calibration.charuco.**make_charuco_with_chessboard**(*dictionary=<cv2.aruco.Dictionary 0x7f5fe5e42c10>*, *charuco_squares=(19, 26)*, *charuco_size=(5, 4)*, *pixels_per_millimetre=10*, *chessboard_squares=(9, 14)*, *chessboard_size=3*, *chessboard_border=0.7*)

Helper function to make an image of a calibration target combining ChArUco markers and a chessboard. It's up to the caller to work out a nice number of pixels per millimetre, so that the resultant image is correctly scaled.

Defaults are as used in SmartLiver project. Not also, that we compute the image and coordinates in portrait, but it's used in landscape.

Parameters

- **dictionary** – ChArUco dictionary
- **charuco_squares** – tuple of (squares in x, squares in y)
- **charuco_size** – tuple of (external size, internal tag size) in mm
- **pixels_per_millimetre** – which determines size of eventual image.
- **chessboard_squares** – tuple of (squares in x, squares in y)
- **chessboard_size** – size of chessboard squares in mm
- **chessboard_border** – border round chessboard, as fraction of square

Returns calibration image

Point Detector Utils

Utilities, specific to the PointDetector stuff.

sksurgeryimage.calibration.point_detector_utils.**write_annotated_image**(*input_image*, *ids*, *image_points*, *image_file_name*)

Takes an input image, copies it, annotates point IDs and writes to the testing output folder.

3.3.3 Utilities

Camera Utilities

Functions to check cameras.

sksurgeryimage.utilities.camera_utilities.**count_cameras**()

Count how many camera sources are available. This is done by trying to instantiate cameras 0..9, and presumes they are in order, sequential, starting from zero.

Returns int, number of cameras

sksurgeryimage.utilities.camera_utilities.**validate_camera_input**(*camera_input*)

Checks that camera_input is an integer, and it is a valid camera.

Param camera_input, integer of camera

Video Interlacing Functions

Functions to support deinterlacing, reinterlacing and vertical destacking of 2D video frames.

sksurgeryimage.processing.interlace.**deinterlace_to_new**(*interlaced*)

Takes the interlaced image, and splits into two new images of even_rows and odd_rows.

Returns even_rows, odd_rows images

sksurgeryimage.processing.interlace.**deinterlace_to_preallocated**(*interlaced*, *even_rows*, *odd_rows*)

Deinterlaces the interlaced image into even_rows and odd_rows images, which must be pre-allocated, and the correct size.

sksurgeryimage.processing.interlace.**deinterlace_to_view**(*interlaced*)

Takes the interlaced image, and returns two new views of even_rows and odd_rows.

Returns even_rows, odd_rows images

sksurgeryimage.processing.interlace.**interlace_to_new**(*even_rows*, *odd_rows*)

Interlaces even_rows and odd_rows images into a new output image.

sksurgeryimage.processing.interlace.**interlace_to_preallocated**(*even_rows*, *odd_rows*, *interlaced*)

Interlaces even_rows and odd_rows images into the interlaced image, where all inputs must be pre-allocated to the correct size.

sksurgeryimage.processing.interlace.**split_stacked_to_new**(*stacked*)
Takes the input stacked image, and extracts the top and bottom half.

Useful if you have stereo 1080x1920 inputs, into an AJA Hi5-3D which stacks them vertically into 2 frames of 540x1920 in the same image.

Returns top_half and bottom_half images

sksurgeryimage.processing.interlace.**split_stacked_to_preallocated**(*stacked*, *top*, *bottom*)
Splits a vertically stacked image, extracting the top and bottom halves, assuming images are the right size and pre-allocated.

Useful if you have stereo 1080x1920 inputs, into an AJA Hi5-3D which stacks them vertically into 2 frames of 540x1920 in the same image.

sksurgeryimage.processing.interlace.**split_stacked_to_view**(*stacked*)
Takes the input stacked image, and returns views that refer to the top and bottom half.

Returns top_half, bottom_half images

sksurgeryimage.processing.interlace.**stack_to_new**(*left*, *right*)
Vertically stack left and right array into single output array. Left and right images should have the same dimensions.

Parameters

- **left** (*numpy array*) – left image
- **right** (*numpy array.*) – right image

sksurgeryimage.processing.interlace.**validate_interlaced_image_sizes**(*even_rows*, *odd_rows*, *interlaced*)
Validates the sizes of the even_rows, odd_rows and interlaced images.

1. Inputs must all be numpy images.
2. Inputs must all have the same number of columns.
3. Inputs must all have an even number of rows.
4. even_rows and odd_rows must have the same number of rows.
5. even_rows and odd_rows must have half the number of rows as interlaced.

Morphological Operators

Functions to support morphological operators.

In many cases, these will just be convenience wrappers around OpenCV functions.

sksurgeryimage.processing.morphological_operators.**dilate_with_cross**(*src*, *dst=None*, *size=3*, *iterations=1*)
Dilates an image with a cross element. OpenCV supports both grey scale and RGB erosion.

Parameters

- **src** – source image

- **dst** – if provided, an image of the same size as src
- **size** – size of structuring element
- **iterations** – number of iterations

Returns the eroded image

sksurgeryimage.processing.morphological_operators.**erode_with_cross**(*src*, *dst=None*, *size=3*, *iterations=1*)

Erodes an image with a cross element. OpenCV supports both grey scale and RGB erosion.

Parameters

- **src** – source image
- **dst** – if provided, an image of the same size as src
- **size** – size of structuring element
- **iterations** – number of iterations

Returns the eroded image

Image Cropper

Misc

Various utilities, like preparing overlay text.

sksurgeryimage.utilities.utilities.**are_similar**(*image0*, *image1*, *threshold=0.995*, *metric=5*, *mean_threshold=0.005*)

Compares two images to see if they are similar.

Parameters

- **image0** (*image0,*) – The images
- **threshold** – The numerical threshold to use, default 0.995
- **method** – The comparison metric, default normalised cross correlation, cv2.TM_CCOEFF_NORMED
- **mean_threshold** – Also compare the mean values of each array, return false if absolute difference of image means divided by the average of both images is greater than the mean_threshold, if less than zero this test will be skipped

Returns True if the metric is greater than the threshols, false otherwise or if the images are not the same dimensions or type

sksurgeryimage.utilities.utilities.**image_means_are_similar**(*image0*, *image1*, *threshold=0.005*)

Compares two images to see if they have similar mean pixel values

Parameters

- **image0** (*image0,*) – The images
- **threshold** – The mean value threshold to use. return false if absolute difference of image means divided by the average of both images is greater than the mean_threshold.

Returns false if absolute difference of image means divided by the average of both images is greater than the mean_threshold, true otherwise or if threshold is less than zero.

sksurgeryimage.utilities.utilities.**noisy_image**(*image*, *mean=0*, *stddev=(50, 5, 5)*)
Creates a noise image, based on the dimensions of the passed image. param: the image to define size and channels of output returns: a noisy image

sksurgeryimage.utilities.utilities.**prepare_cv2_text_overlay**(*overlay_text*, *frame*, *text_scale=1*)
Return settings for text overlay on a cv2 frame.

A class for making a natty WEISS logo

class sksurgeryimage.utilities.weisslogo.**WeissLogo**(*image_size=331.0*)
Bases: object

Creates a WEISS logo and passes a copy on request

get_logo()
Returns the WEISS Logo

Returns The WEISS Logo as a Numpy array

get_noisy_logo()
Returns the WEISS Logo with some noise added

Returns A noisy WEISS Logo as Numpy Array

sksurgeryimage.utilities.weisslogo.**circle**(*img*, *center*, *radius*, *color*[, *thickness*[, *lineType*[, *shift*]]]) → img
. @brief Draws a circle. . . The function cv::circle draws a simple or filled circle with a given center and radius. . @param img Image where the circle is drawn. . @param center Center of the circle. . @param radius Radius of the circle. . @param color Circle color. . @param thickness Thickness of the circle outline, if positive. Negative values, like #FILLED, . mean that a filled circle is to be drawn. . @param lineType Type of the circle boundary. See #LineTypes . @param shift Number of fractional bits in the coordinates of the center and in the radius value.

sksurgeryimage.utilities.weisslogo.**fillConvexPoly**(*img*, *points*, *color*[, *lineType*[, *shift*]]) → img
. @brief Fills a convex polygon. . . The function cv::fillConvexPoly draws a filled convex polygon. This function is much faster than the . function #fillPoly . It can fill not only convex polygons but any monotonic polygon without . self-intersections, that is, a polygon whose contour intersects every horizontal line (scan line) . twice at the most (though, its top-most and/or the bottom edge could be horizontal). . . @param img Image. . @param points Polygon vertices. . @param color Polygon color. . @param lineType Type of the polygon boundaries. See #LineTypes . @param shift Number of fractional bits in the vertex coordinates.

3.4 scikit-surgerycalibration

3.4.1 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*, *concensus_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.

3.4.2 Video Calibration

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 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

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

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

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_patter
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

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.

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_mar**

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

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_*, *min-i-mum_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*, *im*, *ag*, *vi*, *id*, *ob*, *je*, *im*, *ag*, *ho*, *m*, *ra*, *ph*, *ca*, *er*, *di*, *to*, *tio*)

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

3.5 scikit-surgeryutils

3.5.1 Common Overlay Apps

Common use cases for vtk_overlay_window

class sksurgeryutils.common_overlay_apps.**OverlayBaseWidget**(*video_source*, *dims=None*)

Bases: PySide2.QtWidgets.QWidget

Base class for applications that use vtk_overlay_window. The update() method should be implemented in the child class.

Parameters

- **video_source** – OpenCV compatible video source (int or filename)
- **dims** – size of video feed

add_vtk_models_from_dir(*directory*)
Add VTK models to the foreground. :param: directory, location of models

start()
Starts the timer, which repeatedly triggers the update_view() method.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

stop()
Stops the timer.

terminate()
Make sure that the VTK Interactor terminates nicely, otherwise it can throw some error messages, depending on the usage.

update_view()
Update the scene background and/or foreground. Should be implemented by sub class

class sksurgeryutils.common_overlay_apps.**OverlayOnVideoFeed**(*video_source*, *dims=None*)

Bases: *sksurgeryutils.common_overlay_apps.OverlayBaseWidget*

Uses the acquired video feed as the background image, with no additional processing.

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_view()
Get the next frame of input and display it.

class sksurgeryutils.common_overlay_apps.**OverlayOnVideoFeedCropRecord**(*video_source*, *output_filename=None*, *dims=None*)

Bases: *sksurgeryutils.common_overlay_apps.OverlayBaseWidget*

Add cropping of the incoming video feed, and the ability to record the vtk_overlay_window.

Parameters

- **video_source** – OpenCV compatible video source (int or filename)
- **output_filename** – Location of output video file when recording. If none specified, the current date/time is used as the filename.

get_output_frame()
Get the output frame to write in numpy format.

on_record_start()
Start recording data on each frame update. It is expected that this will be triggered using a Qt signal e.g. from a button click. (see sksurgerydavinci.ui.Viewers for examples)

on_record_stop()
Stop recording data.

set_roi()
Crop the incoming video stream using ImageCropper. Function is depreciated due to moving to opencv-headless in sksurgeryvtk. I've left it in for the minute in case any one is using it without my knowlegde

staticMetaObject = <PySide2.QtCore.QMetaObject object>

update_view()
Get the next frame of input, crop and/or write to file (if either enabled).

3.5.2 Misc Utilities

Various image utilities that might be useful in this package.

sksurgeryutils.utils.image_utils.**image_to_pixmap**(*rgb_image*)
Converts an OpenCV image to a Qt pixmap.

Parameters **rgb_image** – OpenCV image, 3 channel, RGB.

Returns QPixmap

Any useful utilities relating to displays/screens.

class sksurgeryutils.utils.screen_utils.**ScreenController**
Bases: object

This class detects the connected screens/monitors, and returns the primary screen and a list of any secondary screens.

list_of_screens()
Return the primary screen and list of other available screens

3.6 scikit-surgerysurfacematch

3.6.1 Algorithms

ICP

PCL ICP implementation of RigidRegistration interface.

class sksurgerysurfacematch.algorithms.pcl_icp_registration.**RigidRegistration**(*max_iterations: int = 100*, *max_corresponder float = 1*, *transformation_epsilon: float = 0.0001*, *fitness_epsilon: float = 0.0001*, *use_lm_icp: bool = True*)

Bases: *sksurgerysurfacematch.interfaces.rigid_registration.RigidRegistration*

Class that uses PCL implementation of ICP to register fixed/moving clouds.

register(*moving_cloud: numpy.ndarray*, *fixed_cloud: numpy.ndarray*)

Uses PCL library, wrapped in scikit-surgerypclcpp.

Parameters

- **moving_cloud** – [Nx3] source/moving point cloud.
- **fixed_cloud** – [Mx3] target/fixed point cloud.

Returns [4x4] transformation matrix, moving-to-fixed space.

GoICP

Go ICP implementation of RigidRegistration interface.

class sksurgerysurfacematch.algorithms.goicp_registration.**RigidRegistration**(*dt_size: int = 200, dt_factor: float = 2.0, normalise: bool = True, num_moving_points: int = 1000, rotation_limits=[-45, 45], trans_limits=[-0.5, 0.5]*)

Bases: *sksurgerysurfacematch.interfaces.rigid_registration.RigidRegistration*

Class that uses GoICP implementation to register fixed/moving clouds. At the moment, we are just relying on all default parameters. :param dt_size: Nodes per dimension of distance transform :param dt_factor: GoICP distance transform factor TODO: rest of params

register(*moving_cloud: numpy.ndarray*, *fixed_cloud: numpy.ndarray*) → numpy.ndarray

Uses GoICP library, wrapped in scikit-surgerygoicp.

Parameters

- **fixed_cloud** – [Nx3] fixed point cloud.
- **moving_cloud** – [Mx3] moving point cloud.
- **normalise** – If true, data will be centred around 0 and normalised.
- **num_moving_points** – How many points to sample from moving cloud if 0, use all points

Returns [4x4] transformation matrix, moving-to-fixed space.

sksurgerysurfacematch.algorithms.goicp_registration.**create_scaling_matrix**(*scale: float*) → numpy.ndarray

Create a scaling matrix, with the same value in each axis.

sksurgerysurfacematch.algorithms.goicp_registration.**create_translation_matrix**(*translate: numpy.ndarray*) → numpy.ndarray

Create translation matrix from 3x1 translation vector.

sksurgerysurfacematch.algorithms.goicp_registration.**demean_and_normalise**(*points_a: numpy.ndarray*, *points_b: numpy.ndarray*)

Independently centre each point cloud around 0,0,0, then normalise both to [-1,1].

Parameters

- **points_a** (*np.ndarray*) – 1st point cloud
- **points_b** (*np.ndarray*) – 2nd point cloud

Returns normalised points clouds, scale factor & translations

sksurgerysurfacematch.algorithms.goicp_registration.**numpy_to_POINT3D_array**(*numpy_pointcloud*)

Covert numpy array to POINT3D array suitable for GoICP algorithm.

sksurgerysurfacematch.algorithms.goicp_registration.**set_rotnode**(*limits_degrees*) → sksurgery-goicp-python.ROTNODE

Setup a ROTNODE with upper/lower rotation limits

sksurgerysurfacematch.algorithms.goicp_registration.**set_transnode**(*trans_limits*) → sksurgery-goicp-python.TRANSNODE

Setup a TRANSNODE with upper/lower limits

Stereo Recon Base Class

Base class for surface reconstruction on already rectified images.

class sksurgerysurfacematch.algorithms.reconstructor_with_rectified_images.**StereoReconstruc**

Bases: *sksurgerysurfacematch.interfaces.stereo_reconstructor.StereoReconstructor*

Base class for those stereo reconstruction methods that work specifically from rectified images. This class handles rectification and the necessary coordinate transformations. Note: The client calls the reconstruct() method which requires undistorted images, which are NOT already rectified. It's THIS class that does the rectification for you, and calls through to the _compute_disparity() method that derived classes must implement.

extract(*left_mask: numpy.ndarray*)

Extracts the actual point cloud. This is a separate method, so that you can reconstruct once using reconstruct(), and then call this extract method with multiple masks, without incurring the cost of multiple calls to the reconstruction algorithm, which may be expensive. :param left_mask: mask image, single channel, same size as left_image :return: [Nx6] point cloud where the 6 columns are x, y, z in left camera space, followed by r, g, b colours.

reconstruct(*left_image: numpy.ndarray*, *left_camera_matrix: numpy.ndarray*, *right_image: numpy.ndarray*, *right_camera_matrix: numpy.ndarray*, *left_to_right_rmat: numpy.ndarray*, *left_to_right_tvec: numpy.ndarray*, *left_mask: numpy.ndarray = None*)

Implementation of stereo surface reconstruction that takes undistorted images, rectifies them, asks derived classes to compute a disparity map on the rectified images, and then sorts out extracting points and their colours.

Camera parameters are those obtained from OpenCV.

Parameters

- **left_image** – undistorted left image, BGR
- **left_camera_matrix** – [3x3] camera matrix
- **right_image** – undistorted right image, BGR
- **right_camera_matrix** – [3x3] camera matrix
- **left_to_right_rmat** – [3x3] rotation matrix
- **left_to_right_tvec** – [3x1] translation vector
- **left_mask** – mask image, single channel, same size as left_image

Returns [Nx6] point cloud where the 6 columns

are x, y, z in left camera space, followed by r, g, b colours.

SGBM Stereo Recon

Surface reconstruction using OpenCV's SGBM reconstruction

class sksurgerysurfacematch.algorithms.sgbm_reconstructor.**SGBMReconstructor**(*min_disparity=16*, *num_disparities=112*, *block_size=3*, *p_1=360*, *p_2=1440*, *disp_12_max_diff=0*, *uniqueness_ratio=0*, *speckle_window_size*, *speckle_range=0*)

Bases: *sksurgerysurfacematch.algorithms.reconstructor_with_rectified_images.StereoReconstructorWithRectifiedImages*

Constructor. See OpenCV StereoSGBM for parameter comments.

Stoyanov Stereo Recon

Surface reconstruction using Stoyanov MICCAI 2010 paper.

class sksurgerysurfacematch.algorithms.stoyanov_reconstructor.**StoyanovReconstructor**(*use_hartl*

Bases: *sksurgerysurfacematch.interfaces.stereo_reconstructor.StereoReconstructor*

Constructor.

reconstruct(*left_image: numpy.ndarray*, *left_camera_matrix: numpy.ndarray*, *right_image: numpy.ndarray*, *right_camera_matrix: numpy.ndarray*, *left_to_right_rmat: numpy.ndarray*, *left_to_right_tvec: numpy.ndarray*, *left_mask: numpy.ndarray = None*)

Implementation of dense stereo surface reconstruction using Dan Stoyanov's MICCAI 2010 method.

Camera parameters are those obtained from OpenCV.

Parameters

- **left_image** – undistorted left image, BGR
- **left_camera_matrix** – [3x3] camera matrix
- **right_image** – undistorted right image, BGR
- **right_camera_matrix** – [3x3] camera matrix
- **left_to_right_rmat** – [3x3] rotation matrix
- **left_to_right_tvec** – [3x1] translation vector
- **left_mask** – mask image, single channel, same size as left_image

Returns [Nx6] point cloud where the 6 columns

are x, y, z in left camera space, and r, g, b, colors.

3.6.2 Interfaces

Rigid Registration

Base class (pure virtual interface) for rigid registration.

class sksurgerysurfacematch.interfaces.rigid_registration.**RigidRegistration**

Bases: object

Base class for classes that can rigidly register (align), two point clouds.

register(*source_cloud: numpy.ndarray*, *target_cloud: numpy.ndarray*)

A derived class must implement this.

Parameters

- **source_cloud** – [Nx3] fixed point cloud.
- **target_cloud** – [Mx3] moving point cloud.

Returns residual, [4x4] transformation matrix, moving-to-fixed space.

Stereo Reconstruction

Base class (pure virtual interface) for classes that do stereo recon.

class sksurgerysurfacematch.interfaces.stereo_reconstructor.**StereoReconstructor**

Bases: object

Base class for stereo reconstruction algorithms. Clients call the reconstruct() method, passing in undistorted images. The output is an [Nx6] array where the N rows are each point, and the 6 columns are x, y, z, r, g, b.

reconstruct(*left_image: numpy.ndarray*, *left_camera_matrix: numpy.ndarray*, *right_image: numpy.ndarray*, *right_camera_matrix: numpy.ndarray*, *left_to_right_rmat: numpy.ndarray*, *left_to_right_tvec: numpy.ndarray*, *left_mask: numpy.ndarray = None*)

A derived class must implement this.

Camera parameters are those obtained from OpenCV.

Parameters

- **left_image** – left image, BGR
- **left_camera_matrix** – [3x3] camera matrix
- **right_image** – right image, BGR
- **right_camera_matrix** – [3x3] camera matrix
- **left_to_right_rmat** – [3x3] rotation matrix
- **left_to_right_tvec** – [3x1] translation vector
- **left_mask** – mask image, single channel, same size as left_image

Returns [Nx6] point cloud in left camera space, where N is the number

of points, and 6 columns are x,y,z,r,g,b.

Video Segmentation

Base class (pure virtual interface) for classes to do video segmentation

class sksurgerysurfacematch.interfaces.video_segmentor.**VideoSegmentor**

Bases: object

Base class for classes that can segment a video image into a binary mask. For example, a deep network that can produce a mask of background=0, foreground=255.

segment(*image: numpy.ndarray*)

A derived class must implement this.

Parameters **image** – image, BGR

Returns image, same size as input, 1 channel, uchar, [0-255].

3.6.3 Processing Pipelines

Register Point Cloud To Stereo Images

Pipeline to register 3D point cloud to 2D stereo video

class sksurgerysurfacematch.pipelines.register_cloud_to_stereo_reconstruction.**Register3DToS**

Bases: object

Class for single-shot, registration of 3D point cloud to stereo video.

register(*reference_cloud: numpy.ndarray*, *left_image: numpy.ndarray*, *right_image: numpy.ndarray*, *initial_ref2recon: numpy.ndarray = None*) → Tuple[numpy.ndarray, numpy.ndarray, numpy.ndarray, numpy.ndarray]

Main method to do a single 3D cloud to 2D stereo video registration.

Camera calibration parameters are in OpenCV format.

Parameters

- **reference_cloud** – [Nx3] points, each row, x,y,z, e.g. from CT/MR.
- **left_image** – undistorted, BGR image
- **right_image** – undistorted, BGR image
- **initial_ref2recon** – [4x4] of initial rigid transform.

Returns residual, [4x4] transform, of reference_cloud to left camera space, [Mx3] downsampled xyz points and [Mx6] reconstructed point cloud, as [x, y, z, r, g, b] rows.

Register Point Cloud To Mosaic

Pipeline to register 3D point cloud to mosaic'ed surface reconstruction.

class sksurgerysurfacematch.pipelines.register_cloud_to_stereo_mosaic.**Register3DToMosaiced**

Bases: object

Class to register a point cloud to a series of surfaces derived from stereo video, and stitched together.

grab(*left_image: numpy.ndarray*, *right_image: numpy.ndarray*)

Call this repeatedly to grab a surface and use ORM key points to match previous reconstruction to the current frame.

Parameters

- **left_image** – undistorted, BGR image
- **right_image** – undistorted, BGR image

register(*point_cloud: numpy.ndarray*, *initial_transform: numpy.ndarray = None*)
Registers a point cloud to the internal mosaicc'ed reconstruction.

Parameters

- **point_cloud** – [Nx3] points, each row, x,y,z, e.g. from CT/MR.
- **initial_transform** – [4x4] of initial rigid transform.

Returns residual, [4x4] transform, of point_cloud to left camera space,

and [Mx6] reconstructed point cloud, as [x, y, z, r, g, b] rows.

reset()
Reset's internal data members, so that you can start accumulating data again.

3.7 scikit-surgerytf

3.7.1 Segmentation

Liver Segmentation UNet

Module to implement a semantic (pixelwise) segmentation using UNet on 512x512.

class sksurgerytf.models.rgb_unet.**RGBUNet**(*logs='logs/fit'*, *data=None*, *working=None*, *omit=None*, *model=None*, *learning_rate=0.0001*, *epochs=50*, *batch_size=2*, *input_size=(512, 512, 3)*, *patience=20*)
Class to encapsulate RGB UNet semantic (pixelwise) segmentation network.

Thanks to Zhixuhao, and ShawDa for getting me started, and 'Harshall Lamba <https://towardsdatascience.com/understanding-semantic-segmentation-with-unet-6be4f42d4b47>_, for further inspiration.

predict(*rgb_image*)
Method to test a single image. Image resized to match network, segmented and then resized back to match the input size.

Parameters **rgb_image** – 3 channel RGB, [0-255], uchar.

Returns single channel, [0=bgl255=fg].

save_model(*filename*)
Method to save the whole trained network to disk.

Parameters **filename** – file to save to.

train()
Method to train the neural network. Writes each epoch to tensorboard log files.

Returns output of self.model.evaluate on validation set, or None.

sksurgerytf.models.rgb_unet.**run_rgb_unet_model**(*logs*, *data*, *working*, *omit*, *model*, *save*, *test*, *prediction*, *epochs*, *batch_size*, *learning_rate*, *patience*)

Helper function to run the RGBUnet model from the command line entry point.

Parameters

- **logs** – directory for log files for tensorboard.
- **data** – root directory of training data.
- **working** – working directory for organising data.
- **omit** – patient identifier to omit, when doing Leave-One-Out.
- **model** – file of previously saved model.
- **save** – file to save model to.
- **test** – input image to test.
- **prediction** – output image, the result of the prediction on test image.
- **epochs** – number of epochs.
- **batch_size** – batch size.
- **learning_rate** – learning rate for optimizer.
- **patience** – number of steps to tolerate non-improving accuracy

3.7.2 Fashion MNIST Example Classifer

Module to implement a basic classifier for the Fashion MNIST dataset. The aim of this module is to demonstrate how to create a class that can be developed, tested and re-used effectively. It is not a demonstration on how to do deep learning, or classification per se.

Inspired by TensorFlow tutorials.

class sksurgerytf.models.fashion.**FashionMNIST**(*logs='logs/fit'*, *model=None*, *learning_rate=0.001*, *epochs=1*)

Class to encapsulate a classifier for the Fashion MNIST dataset.

extract_failures(*number_to_fetch*)

Returns incorrectly classified test images. :param number_to_fetch: int, the number to find.

This method is slow, its only for demo purposes.

Returns indexes, images, predicted, labels

get_class_names()

Returns a copy of the valid class names. We return copies to stop external people accidentally editing the internal copies. It's safer in the long run, although in Python easy to work around.

Returns list of strings

get_test_image(*index*)

Extracts an image from the test data. Useful for unit testing, as the original data comes packaged up in a zip file.

Parameters **index** – int [0-9999], unchecked

Returns image, (28 x 28), numpy, single channel, [0-255], uchar.

save_model (*filename*)
Method to save the whole trained network to disk.

Parameters **filename** – file to save to.

test (*image*)
Method to test a single (28 x 28) image.

Parameters **image** – (28 x 28), numpy, single channel, [0-255], uchar.

Returns (class_index, class_name)

train ()
Method to train the neural network. Writes each epoch to tensorboard log files.

Returns output of self.model.evaluate on test set.

sksurgerytf.models.fashion.**run_fashion_model** (*logs*, *model*, *save*, *test*)
Helper function to run the Fashion MNIST model from the command line entry point.

Parameters

- **logs** – directory for log files for tensorboard.
- **model** – file of previously saved model.
- **save** – file to save weights to
- **test** – image to test

3.8 scikit-surgerytorch

3.8.1 Stereo Reconstruction

High Resolution Stereo

Module to implement Hierarchical Deep Stereo Matching on High Resolution Images network.

class sksurgerytorch.models.high_res_stereo.**HSMNet** (*max_disp: int = 255*, *entropy_threshold: float = -1*, *level: int = 1*, *scale_factor: float = 0.5*, *weights=None*)
Class to encapsulate network form 'Hierarchical Deep Stereo Matching on High Resolution Images'.

Thanks to Gengshang Yang, for their network implementation.

Parameters

- **max_disp** – Maximum number of disparity levels
- **entropy_threshold** – Pixels with entropy above this value will be ignored in the disparity map. Disabled if set to -1.
- **level** – Set to 1, 2 or 3 to trade off quality of depth estimation against runtime. 1 = best depth estimation, longer runtime, 3 = worst depth estimation, fastest runtime.
- **scale_factor** – Images can be resized before passing to the network, for perfomance impromvents. This sets the scale factor.
- **weights** – Path to trained model weights (.tar file)

predict (*left_image: numpy.ndarray*, *right_image: numpy.ndarray*) → numpy.ndarray
Predict disparity from a pair of stereo images.

Parameters

- **left_image** (*np.ndarray*) – Left stereo image, 3 channel RGB
- **right_image** (*np.ndarray*) – Right stero image, 3 channel RGB

Returns Predicted disparity, grayscale

Return type np.ndarray

sksurgerytorch.models.high_res_stereo.**run_hsmnet_model**(*max_disp*, *entropy_threshold*, *level*, *scale_factor*, *weights*, *left_image*, *right_image*, *output_file*)
This is for the command line entry point

class sksurgerytorch.models.high_res_stereo.**toTensorLegacy**
.

3.8.2 Non Rigid Registration

Volume 2 Surface CNN

V2SNet Model Impementation

class sksurgerytorch.models.volume_to_surface.**Volume2SurfaceCNN**(*mask: bool = True*, *weights: str = None*, *grid_size: int = 64*)
Class to encapsulate network form 'Non-Rigid Volume to Surface Registration using a Data-Driven Biomechanical Model'.

Thanks to Micha Pfieffer, for their network implementation.

Parameters

- **mask** (*bool*) – If true, use masking
- **weights** (*str*) – Path to trained model weights (.tar file)

predict (*preoperative: numpy.ndarray*, *intraoperative: numpy.ndarray*) → numpy.ndarray
Predict the displacement field between model and surface.

Parameters

- **preoperative** (*np.ndarray*) – Preoperative surface/point cloud
- **intraoperative** (*np.ndarray*) – Intraoperative surface/point cloud

Returns Displacement field

Return type np.ndarray

3.9 scikit-surgerynditracker

3.9.1 NDI Tracking

Class implementing communication with NDI (Northern Digital) trackers

class sksurgerynditracker.nditracker.**NDITracker**(*configuration*)
Bases: sksurgerycore.baseclasses.tracker.SKSBaseTracker

Class for communication with NDI trackers. Should support Polaris, Aurora, and Vega. Currently only tested with wireless tools on Vega

close()
Closes the connection to the NDI Tracker and deletes the tracker device.

Raises **Exception** – ValueError

get_frame()
Gets a frame of tracking data from the NDI device.

Returns

port_numbers : list of port handles, one per tool

time_stamps : list of timestamps (cpu clock), one per tool

frame_numbers : list of framenumbers (tracker clock) one per tool

tracking : list of 4x4 tracking matrices, rotation and position, or if use_quaternions is true, a list of tracking quaternions, column 0-2 is x,y,z column 3-6 is the rotation as a quaternion.

tracking_quality : list the tracking quality, one per tool.

Note: The time stamp is based on the host computer clock. Read the following extract from NDI's API Guide for advice on what to use: "Use the frame number, and not the host computer clock, to identify when data was collected. The frame number is incremented by 1 at a constant rate of 60 Hz. Associating a time from the host computer clock to replies from the system assumes that the duration of time between raw data collection and when the reply is received by the host computer is constant. This is not necessarily the case."

get_tool_descriptions()
Returns the port handles and tool descriptions

start_tracking()
Tells the NDI devices to start tracking.

Raises **Exception** – ValueError

stop_tracking()
Tells the NDI devices to stop tracking.

Raises **Exception** – ValueError

sksurgerynditracker.nditracker.**int2byte**()
S.pack(v1, v2, ...) -> bytes

Return a bytes object containing values v1, v2, ... packed according to the format string S.format. See help(struct) for more on format strings.

3.10 scikit-surgeryarucotracker

3.10.1 ARuCo Tracking

A class for straightforward tracking with an ARuCo

class sksurgeryarucotracker.arucotracker.**ArUcoTracker**(*configuration*)
Bases: sksurgerycore.baseclasses.tracker.SKSBaseTracker

Initialises and Configures the ArUco detector

Parameters **configuration** – A dictionary containing details of the tracker.

video source: defaults to 0

aruco dictionary: defaults to DICT_4X4_50

marker size: defaults to 50 mm

camera projection: defaults to None

camera distortion: defaults to None

smoothing buffer: specify a buffer over which to average the tracking, defaults to 1

rigid bodies: a list of rigid bodies to track, each body should have a 'name', a 'filename' where the tag geometry is defined, and an 'aruco dictionary' to use. Additionally we can include 'tag width' in mm when the tag has been scaled during printing or is displayed on a mobile phone screen or similar

Raises **Exception** – ImportError, ValueError

close()
Closes the connection to the Tracker and deletes the tracker device.

Raises **Exception** – ValueError

get_frame(*frame=None*)
Gets a frame of tracking data from the Tracker device.

Parameters **frame** – an image to process, if None, we use the OpenCV video source.

Returns

port_numbers: If tools have been defined port numbers are the tool descriptions. Otherwise port numbers are the aruco tag ID prefixed with aruco

time_stamps : list of timestamps (cpu clock), one per tool

frame_numbers : list of framenumbers (tracker clock) one per tool

tracking : list of 4x4 tracking matrices, rotation and position,

tracking_quality : list the tracking quality, one per tool.

Raises **Exception** – ValueError

get_tool_descriptions()
Returns tool descriptions

has_capture()
:Returns true if the tracker has it's own opencv source otherwise false

start_tracking()
Tells the tracking device to start tracking. :raise Exception: ValueError

stop_tracking()
Tells the tracking devices to stop tracking. :raise Exception: ValueError

Python Module Index

S

Index

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

W