scikit-surgerysurfacematch Documentation

Matt Clarkson

Oct 22, 2020

Contents

Author: Matt Clarkson

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

scikit-surgerysurfacematch supports Python 3.6 - 3.8

scikit-surgerysurfacematch contains algorithms that are useful in stereo reconstruction from video images, and matching to a pre-operative 3D model, represented as a point cloud.

CHAPTER 1

Features

- Base classes (pure virtual interfaces), for video segmentation, stereo reconstruction, rigid registration / pose estimation. See `sksurgerysurfacematch/algorithms`
- A base class to handle rectification properly, and the right coordinate transformation, to save you the trouble.
- Stereo reconstruction classes based on Stoyanov MICCAI 2010, and OpenCV SGBM reconstruction, using above interface, and both allowing for optional masking.
- Rigid registration using PCL's ICP implementation, which is wrapped in scikit-surgerypclcpp
- A pipeline to combine the above, segment a video pair, do reconstruction, and register to a 3D model, where each part can then be swapped with whatever implementation you want, as long as you implement the right interface.
- A pipeline to take multiple stereo video snapshots, do surface reconstruction, mosaic them together, and then register to a 3D model. Again, each main component (video segmentation, surface reconstruction, rigid registration) is swappable. Inspired by: [Xiaohui Zhang's](https://doi.org/10.1007/s11548-019-01974-6) method.

CHAPTER 2

Developing

2.1 Cloning

You can clone the repository using the following command:

```
git clone https://github.com/UCL/scikit-surgerysurfacematch
```

2.2 Running tests

Pytest is used for running unit tests:

```
pip install pytest
python -m pytest
```

2.3 Linting

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

```
pip install pylint
pylint --rcfile=tests/pylintrc sksurgerysurfacematch
```

CHAPTER 3

Installing

You can pip install directly from the repository as follows:

```
pip install git+https://github.com/UCL/scikit-surgerysurfacematch
```

3.1 Contributing

Please see the contributing guidelines.

3.2 Useful links

- Source code repository
- Documentation

CHAPTER 4

Licensing and copyright

CHAPTER 5

Acknowledgements

Supported by Wellcome and EPSRC.

5.1 Requirements for scikit-surgerysurfacematch

This is the software requirements file for scikit-surgerysurfacematch, part of the SNAPPY project. The requirements listed below should define what scikit-surgerysurfacematch does. Each requirement can be matched to a unit test that checks whether the requirement is met.

5.1.1 Requirements

ID	Description	Test
0000	Module has a help page	pylint, see tests/pylint.rc and tox.ini
0001	Functions are documented	pylint, see tests/pylint.rc and tox.ini
0002	Package has a version number	No test yet, handled by git.

5.2 stable

5.2.1 sksurgerysurfacematch package

Subpackages

sksurgerysurfacematch.algorithms package

Submodules

sksurgerysurfacematch.algorithms.goicp_registration module

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

sksurgerysurfacematch.algorithms.pcl_icp_registration module

PCL ICP implementation of RigidRegistration interface.

class sksurgerysurfacematch.algorithms.pcl_icp_registration.**RigidRegistration**(*max_iterations: int = 100*, *max_corresponder float = 1*, *trans-for-ma-tion_epsilon: float = 0.0001*, *fit-ness_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.

sksurgerysurfacematch.algorithms.reconstructor_with_rectified_images module

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.

sksurgerysurfacematch.algorithms.sgbm_reconstructor module

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.

sksurgerysurfacematch.algorithms.stoyanov_reconstructor module

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.

sksurgerysurfacematch.algorithms.value_threshold_segmentor module

Dummy segmentor, just to test the framework.

class sksurgerysurfacematch.algorithms.value_threshold_segmentor.**ValueThresholdSegmentor**(*th*
Bases: *sksurgerysurfacematch.interfaces.video_segmentor.VideoSegmentor*

Dummy segmentor, to test the framework. Simply converts BGR to HSV, extracts the value channel, and applies a threshold between [0-255].

It's not really useful for anything other than testing the interface.

segment(*image: numpy.ndarray*)
Converts image from BGR to HSV and thresholds the Value channel.

Parameters **image** – image, BGR

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

Module contents

sksurgerysurfacematch.interfaces package

Submodules

sksurgerysurfacematch.interfaces.rigid_registration module

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.

sksurgerysurfacematch.interfaces.stereo_reconstructor module

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.

sksurgerysurfacematch.interfaces.video_segmentor module

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].

Module contents

sksurgerysurfacematch.pipelines package

Submodules

sksurgerysurfacematch.pipelines.register_cloud_to_stereo_mosaic module

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

class sksurgerysurfacematch.pipelines.register_cloud_to_stereo_mosaic.**Register3DToMosaicedS**

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.

sksurgerysurfacematch.pipelines.register_cloud_to_stereo_reconstruction module

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.

Module contents

sksurgerysurfacematch.ui package

Module contents

scikit-surgerysurfacematch

sksurgerysurfacematch.utils package

Submodules

sksurgerysurfacematch.utils.ply_utils module

Methods for saving .ply files etc.

sksurgerysurfacematch.utils.ply_utils.**write_ply**(*ply_data: list*, *ply_file: str*)
Writes a .ply format file.

Parameters

- **ply_data** – points and colours stored as list
- **ply_file** – file name

sksurgerysurfacematch.utils.ply_utils.**write_pointcloud**(*points: numpy.ndarray*, *colours: numpy.ndarray*, *file_name: str*)
Write point cloud points and colours to .ply file. :param points: [Nx3] ndarray, of x, y, z coordinates :param colours: [Nx3] ndarray, of r, g, b colours :param file_name: filename including .ply extension

sksurgerysurfacematch.utils.projection_utils module

Various utilities, mainly to help testing.

sksurgerysurfacematch.utils.projection_utils.**reproject_and_save**(*image*, *model_to_camera*, *point_cloud*, *camera_matrix*, *output_file*)
For testing purposes, projects points onto image, and writes to file.

Parameters

- **image** – BGR image, undistorted.
- **model_to_camera** – [4x4] ndarray of model-to-camera transform
- **point_cloud** – [Nx3] ndarray of cloud of points to project
- **camera_matrix** – [3x3] OpenCV camera_matrix (intrinsics)

- **output_file** – file name

sksurgerysurfacematch.utils.registration_utils module

Various registration routines to reduce duplication.

sksurgerysurfacematch.utils.registration_utils.**do_rigid_registration**(*reconstructed_cloud*, *reference_cloud*, *rigid_registration: sksurgerysurfacematch.interfaces.rigid_registrat*, *initial_ref2recon: numpy.ndarray = None*)

Triggers a rigid body registration using rigid_registration. :param reconstructed_cloud: [Nx3] point cloud, e.g. from video. :param reference_cloud: [Mx3] point cloud, e.g. from CT/MR :param rigid_registration: Object that implements a rigid registration. :param initial_ref2recon_transform: [4x4] ndarray representing an initial estimate. :return: residual (float), [4x4] transform

Module contents

Module contents

scikit-surgerysurfacematch

5.3 First notebook

You can write up experiments in notebooks, and they can be generated into Sphinx docs using `tox -e docs`, and for example set up to run on readthedocs.

See this and this examples.

5.3.1 NOTE:

Getting jupyter to run your code in this package relies on 3 things:

- You must ensure you start jupyter within the tox environment.

```
# If not already done.
source .tox/py36/bin/activate

# Then launch jupyter
jupyter notebook
```

- Then when you navigate to and run this notebook, select the right kernel (named after your project) from the kernel menu item, in the web browser.
- Add project folder to system path, as below.

[1]:
```
# Jupyter notebook sets the cwd to the folder containing the notebook.
# So, you want to add the root of the project to the sys path, so modules load
↪correctly.
import sys
sys.path.append("../../")
```

- modindex
- genindex
- search

Python Module Index

Index

V

W