bb_stitcher Documentation

Release 0.0.0.dev6

gitmirgut

Jan 07, 2018

Contents

CHAPTER 1

Installation

This part of the documentation covers the installation of the `bb_stitcher`.

1.1 (Recommended) Conda Installation

The following steps will install all dependencies including opencv.

1. Download and install Conda:

```
$ wget https://repo.continuum.io/miniconda/Miniconda3-latest-Linux-x86_64.sh
$ bash Miniconda3-latest-Linux-x86_64.sh
$ conda update conda
```

2. Setup conda environment for `bb_stitcher`:

```
$ conda create --name bb_stitcher_env python=3
$ source activate bb_stitcher_env
(bb_stitcher_env)$ conda install --channel https://conda.binstar.org/menpo opencv3
```

3. Install the `bb_stitcher`:

```
(bb_stitcher_env) $ pip install git+https://github.com/BioroboticsLab/bb_stitcher.
↪git
```

1.2 Pip Installation

Important: The following requirements must be installed manually and cannot be installed by pip:

- OpenCV3

- opencv_contrib

Good Instruction for installing opencv with opencv_contrib package can be found under pyimagesearch.

1.2.1 Installation from GitHub

Direct install from GitHub:

```
pip install git+https://github.com/BioroboticsLab/bb_stitcher.git
```

1.2.2 Installation from source

For developers the following setup instruction is recommended.

Get the source

You can clone the public repository:

```
$ git clone https://github.com/BioroboticsLab/bb_stitcher.git
```

Or, download the tarball:

```
$ wget curl -OL https://github.com/BioroboticsLab/bb_stitcher/tarball/master
```

Install the source

Enter the directory and install dependencies using:

```
$ pip install -r requirements.txt
```

Then, install bb_stitcher using:

```
$ pip install -e .
```

CHAPTER 2

Tutorial

2.1 Estimate Parameters

For the Estimation of the parameters for stitching/surveying of two images of one comb side, you could use the `bb_stitcher` command from commandline. The following command gives you an overview of the possible commands:

```
$ bb_stitcher -h
```

If you want to estimate the parameters for the stitching, you have to use:

```
$ bb_stitcher estimate
```

To see all the positional arguments needed, use `$bb_stitcher estimate -h`:

```
positional arguments:
{fb,rect}    Define the stitcher to use:
                fb - FeatureBasedStitcher
                rect - RectangleStitcher
left         Path of the left image.
right        Path of the right image.
left_angle   Rotation angle of the left image (counter-clockwise).
right_angle  Rotation angle of the right image (counter-clockwise).
left_camID   Cam ID of the camera which shot the left image.
right_camID  Cam ID of the camera which shot the right image.
out          Output path of the stitching data.
             Supported Types: .npz,.csv
```

As you can see there are two options for estimation of the parameters. `fb`-FeatureBasedStitcher and `rect`-RectangleSticher. These are two different approaches, the FeatureBasedStitcher is based on Feature Detection and Feature Matching. The second one so-called RectangleStitcher maps special marked points on the comb border to an abstracted rectangle.

Example:

```
$bb_stitcher estimate fb Cam_0_2016-09-01T12:56:50.801920Z.jpg Cam_1_2016-09-
↪01T12:56:50.801926Z.jpg 90 -90 0 1 parameters.csv
```

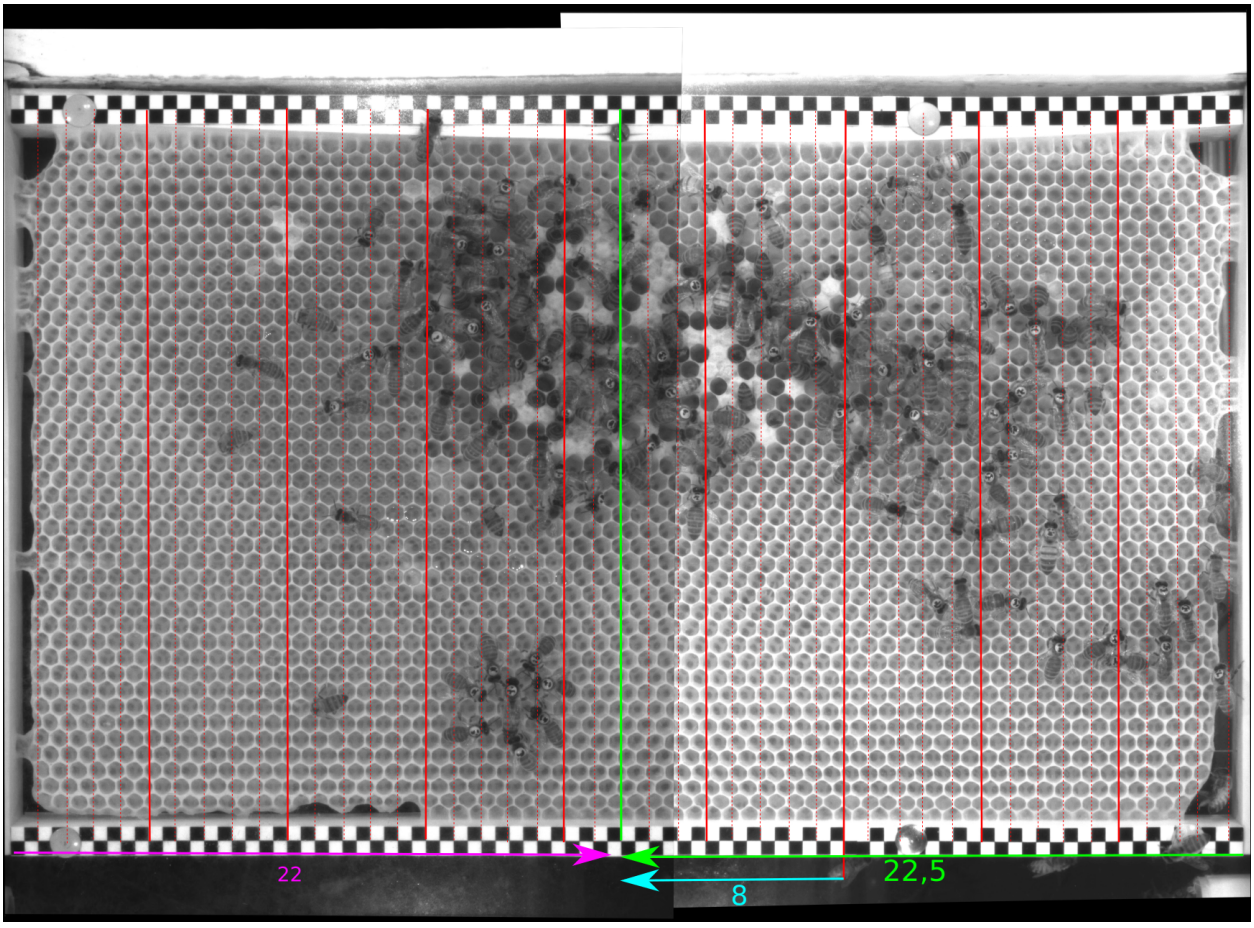
2.1.1 Steps of Estimation

The process of parameter estimation is divided in 3 steps:

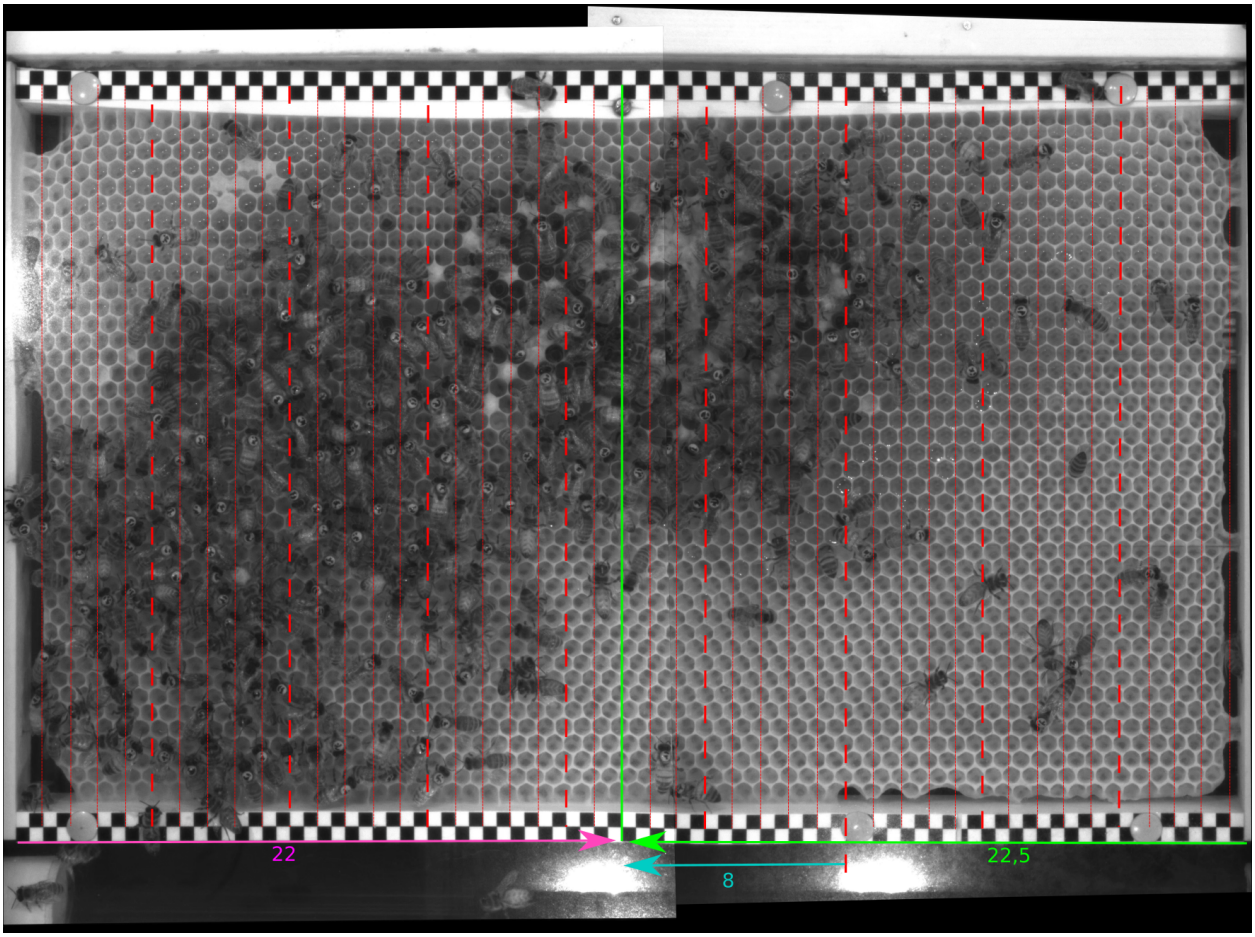
1. Step: Estimate parameters to form a panorama.
2. Step: Determine the origin.
3. Step: Determine the ratio between pixels an mm.

2.2 Reference Images

2.2.1 Cam 0 and 1



2.2.2 Cam 2 and 3



CHAPTER 3

Python API reference

3.1 Core

Module to connect the stitching and mapping from image coordinates to world coordinates.

class Surveyor(*config=None*)

Bases: object

Class to determine the relationship between two images of one comb side .

The *Surveyor* determines all needed data to stitch two images from different areas of one comb side to a complete view of the comb. On this basis the *Surveyor* can also be used to map the coordinates from these images to hive coordinates.

load(*path*)

Load saved parameters for mapping from file.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

save(*path*)

Save parameters of the *Surveyor* needed for later stitching to a file.

Parameters **path** (*str*) – Path of the output file. The extension must be '.npz' or '.csv'.

See also:

- *io_utils*

determine_mapping_parameters(*path_l*, *path_r*, *angl_l*, *angl_r*, *cam_id_l*, *cam_id_r*, *stitcher_type*)

Determine the parameters for mapping of images and coordinates.

This functions is used to calculate all needed data to stitch two images and to map image coordinates/angels to hive coordinates/angles.

Parameters

- **path_l** (*str*) – Path to the left image.

- **path_r** (*str*) – Path to the right image.
- **angl_l** (*int*) – Angle in degree to rotate left image.
- **angl_r** (*int*) – Angle in degree to rotate right image.
- **cam_id_l** (*int*) – ID of the camera, which shot the left image.
- **cam_id_r** (*int*) – ID of the camera, which shot the right image.
- **stitcher_type** (*Stitcher*) – Stitcher to use for stitching of the images.

get_parameters()
Return the estimated or loaded parameters of the *Surveyor* needed for later stitching.

With this function you could save the *Surveyor* parameters and load them later for further stitching of images and mapping of image coordinates/angels to hive coordinates/angles in relation to hive.

set_parameters(*homo_left*, *homo_right*, *size_left*, *size_right*, *cam_id_l*, *cam_id_r*, *origin*, *ratio_px_mm*)
Load needed parameters for mapping image points/angles to hive coordinates/angles.

This function becomes handy if you calculated the parameters in an earlier surveying process and did not want to calculate the parameters again and just want to map image points/angles to hive coordinates/angles.

Parameters

- **homo_left** (*ndarray*) – homography *(3,3)* for data from the left side to form a panorama.
- **homo_right** (*ndarray*) – homography *(3,3)* for data from the right side to form a panorama.
- **size_left** (*tuple*) – Size of the left image in px, which was used to calculate homography.
- **size_right** (*tuple*) – Size of the right image in px, which was used to calculate homography.
- **cam_id_l** (*int*) – ID of the camera, which shot the left image.
- **cam_id_r** (*int*) – ID of the camera, which shot the right image.
- **origin** (*ndarray*) – Origin of the stitched data/image in px *(2,)*.
- **ratio_px_mm** (*float*) – Ratio to convert pixel to mm.
- **pano_size** (*tuple*) – Size of the panorama in px.

map_points_angles(*points*, *angles*, *cam_id*)
Map image points/angles to points/angles in relation to world/hive.

This happens under the assumption that the mapping parameters were estimated or loaded before.

Parameters

- **points** (*ndarray*) – List of points from left image in px *(N,2)*.
- **angles** (*ndarray*) – Angles in rad (length *(N,)*).
- **cam_id** (*ndarray*) – ID of the camera, which shot the image.

Returns

- **points_mapped** (ndarray) – `points` mapped to hive in mm *(N,2)*.
- **angles_mapped** (ndarray) – `angles` mapped to *(N,)*.

Note: For all angles in `angles` it is assumed that a 0°-angle shows to the right border of the image and that a positive angle means clockwise rotation.

compose_panorama(*path_l*, *path_r*, *grid=False*)
Try to compose the given images into the final panorama.

This happens under the assumption that the mapping parameters were estimated or loaded before.

Parameters

- **path_l** (*str*) – Path to the left image.
- **path_r** (*str*) – Path to the right image.
- **grid** (*bool, optional*) – If `True` a grid with axes in mm will be drawn on the image.

3.2 Preparation

This module provides functions to prepare an image or points for stitching.

class Rectificator(*config*)
Bases: `object`

Class to rectify images and points.

Remove lens distortion from images and points.

rectify_image(*image*)
Remove lens distortion from an image.

Parameters **image** (*ndarray*) – Input (distorted) image.

Returns Output (corrected) image with same size and type as `image`.

Return type ndarray

rectify_points(*points*, *size*)
Remove lens distortion from points.

Map points determined from distorted image to its position in an undistorted image.

Parameters

- **points** (*ndarray*) – List of (distorted) points *(N,2)*.
- **size** (*tuple*) – Size *(width, height)* of the image, which was used to determine the points.

Returns List of corrected `points`.

Return type ndarray

rectify_points_angles(*points*, *angles*, *size*)
Remove lens distortion from angles.

Map angles determined from distorted image to its angles in an undistorted image.

Parameters

- **points** (*ndarray*) – List of (distorted) points *(N,2)*.
- **angles** (*ndarray*) – List of Angles in rad (length *(N,)*).

- **size** (*tuple*) – Size *(width, height)* of the image, which was used to determine the points.

Returns

- **rect_points** (ndarray) – List of corrected `points`
- **rect_angles** (ndarray) – List of corrected `angles`

rotate_image(*image*, *angle*)
Rotate image by given angle.

Parameters

- **image** (*ndarray*) – Input image.
- **angle** (*int*) – Rotation angle in degree. Positive value means counter-clockwise rotation.

Returns

- **rot_image** (ndarray) – Rotated `image`.
- **affine_mat** (ndarray) – An affine *(3,3)*–matrix for rotation of image or points.

rotate_points(*points*, *angle*, *size*)
Rotate points by given angle and in relation to the size of an image.

Parameters

- **points** (*ndarray*) – List of points *(N, 2)*.
- **angle** (*int*) – Rotation angle in degree. Positive values mean counter-clockwise rotation.
- **size** (*tuple*) – Size *(width, height)* of the image, which was used to determine the `points`.

Returns Rotated `points` *(N, 2)*.

Return type ndarray

3.3 Stitcher

This module contains various image stitchers especially designed for the BeesBook Project.

class Stitcher(*config=None*, *rectify=True*)
Bases: `object`

Class to create a 'panorama' from two images.

> **Warning:** This class is more like an abstract class. This *Stitcher* can not be used to estimate the required parameters for stitching. But if you already estimated the parameters with an other *Stitcher* you could use this on for stitching.

See also:

- *FeatureBasedStitcher*
- *RectangleStitcher*

load_parameters(*homo_left=None*, *homo_right=None*, *size_left=None*, *size_right=None*)
Load needed parameters for stitching points, angles and images.

This function becomes handy if you calculate the parameters in an earlier stitching process and did not want to calculate the parameters again and just want to map points, angles or images which were made under the same camera setup as the earlier stitching process.

Parameters

- **homo_left** (*ndarray*) – homography *(3,3)* for data from the left side to form a panorama.
- **homo_right** (*ndarray*) – homography *(3,3)* for data from the right side to form a panorama.
- **size_left** (*tuple*) – Size of the left image, which was used to calculate homography.
- **size_right** (*tuple*) – Size of the right image, which was used to calculate homography.

get_parameters()

Return the estimated or loaded parameters of the stitcher needed for later stitching.

With this function you could save the stitching parameters and load them later for further stitching of points and angles (see `set_parameters`).

Use this function if you estimated the transform and did not want to estimate the parameters again.

estimate_transform(*image_left*, *image_right*, *angle_left=0*, *angle_right=0*)

Estimate transformation/homography of the left and right images/data to form a panorama.

Return the transformation matrix for the left and right image.

Parameters

- **image_left** (*ndarray*) – Input left image.
- **image_right** (*ndarray*) – Input right image.
- **angle_left** (*int*) – Angle in degree to rotate left image.
- **angle_right** (*int*) – Angle in degree to rotate right image.

> **Warning:** This must be overridden by a sublcass to customize stitching.

compose_panorama(*image_left*, *image_right*)

Try to compose the given images into the final panorama.

This happens under the assumption that the image transformations were estimated or loaded before.

Parameters

- **image_left** (*ndarray*) – Input left image.
- **image_right** (*ndarray*) – Input right image.

Returns panorama (stitched image)

Return type ndarray

map_left_points(*points*)

Map points from the left image to the panorama.

This happens under the assumption that the image transformations were estimated or loaded before.

Parameters **points** (*ndarray (float)*) – List of points from left image *(N,2)*.

Returns `points` mapped to panorama *(N,2)*

Return type ndarray

map_left_points_angles(*points*, *angles*)
Map points and angles from the left image to the panorama.

This happens under the assumption that the image transformations were estimated or loaded before.

Parameters

- **points** (*ndarray (float)*) – List of points from left image *(N,2)*.
- **angles** (*ndarray*) – Angles in rad (length *(N,)*).

Returns

- **points_mapped** (ndarray) – `points` mapped to panorama *(N,2)*
- **angles_mapped** (ndarray) – `angles` mapped to panorama *(N,)*

map_right_points(*points*)
Map points from the right image to the panorama.

This happens under the assumption that the image transformations were estimated or loaded before.

Parameters **points** (*ndarray (float)*) – List of points from right image *(N,2)*.

Returns `points` mapped to panorama *(N,2)*

Return type ndarray

map_right_points_angles(*points*, *angles*)
Map points and angles from the right image to the panorama.

This happens under the assumption that the image transformations were estimated or loaded before.

Parameters

- **points** (*ndarray (float)*) – List of points from right image *(N,2)*.
- **angles** (*ndarray*) – Angles in rad (length *(N,)*).

Returns

- **points_mapped** (ndarray) – `points` mapped to panorama *(N,2)*
- **angles_mapped** (ndarray) – `angles` mapped to panorama *(N,)*

class FeatureBasedStitcher(*config=None*, *rectify=True*)
Bases: *bb_stitcher.stitcher.Stitcher*

Class to create a feature based stitcher.

estimate_transform(*image_left*, *image_right*, *angle_left=0*, *angle_right=0*)
Estimate transformation for stitching of images based on feature matching.

Parameters

- **image_left** (*ndarray*) – Input left image.
- **image_right** (*ndarray*) – Input right image.
- **angle_left** (*int*) – Angle in degree to rotate left image.
- **angle_right** (*int*) – Angle in degree to rotate right image.

class RectangleStitcher(*config=None*, *rectify=True*)
Bases: *bb_stitcher.stitcher.Stitcher*

Class to create a rectangle stitcher.

The `RectangleStitcher` maps selected points to an abstracted rectangle.

estimate_transform(*image_left*, *image_right*, *angle_left=0*, *angle_right=0*)
Estimate transformation for stitching of images based on 'rectangle' Stitching.

Parameters

- **image_left** (*ndarray*) – Input left image.
- **image_right** (*ndarray*) – Input right image.
- **angle_left** (*int*) – Angle in degree to rotate left image.
- **angle_right** (*int*) – Angle in degree to rotate right image.

3.4 Helpers

This module provides various helper functions.

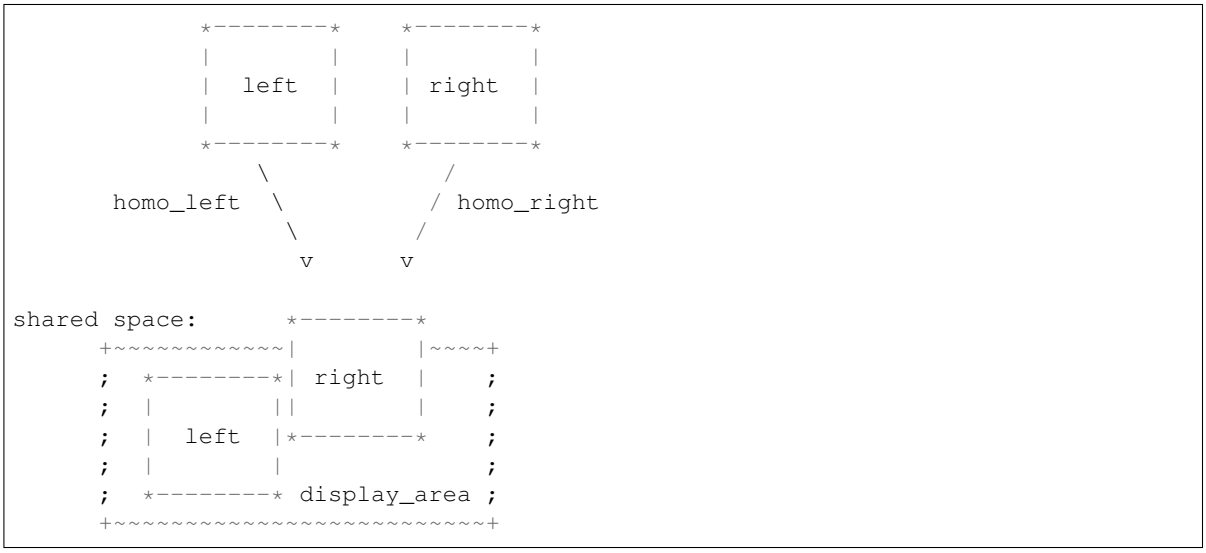
get_default_config()
Return the default config.

get_default_debug_config()
Return the default logging config file.

get_boundaries(*size_left*, *size_right*, *homo_left*, *homo_right*)
Determine the boundaries of two transformed images.

When two images have been transformed by homographies to a 'shared space' (which holds both images), it's possible that this 'shared space' is not aligned with the displayed area. Its possible that various points are outside of the display area. This function determines the max/min values of x and y of the both images in shared space in relation to the origin of the display area.

Example



(In this example `xmin` would be the x value of the left border from the left image and `ymin` would be the y value of the top border from the right image)

Parameters

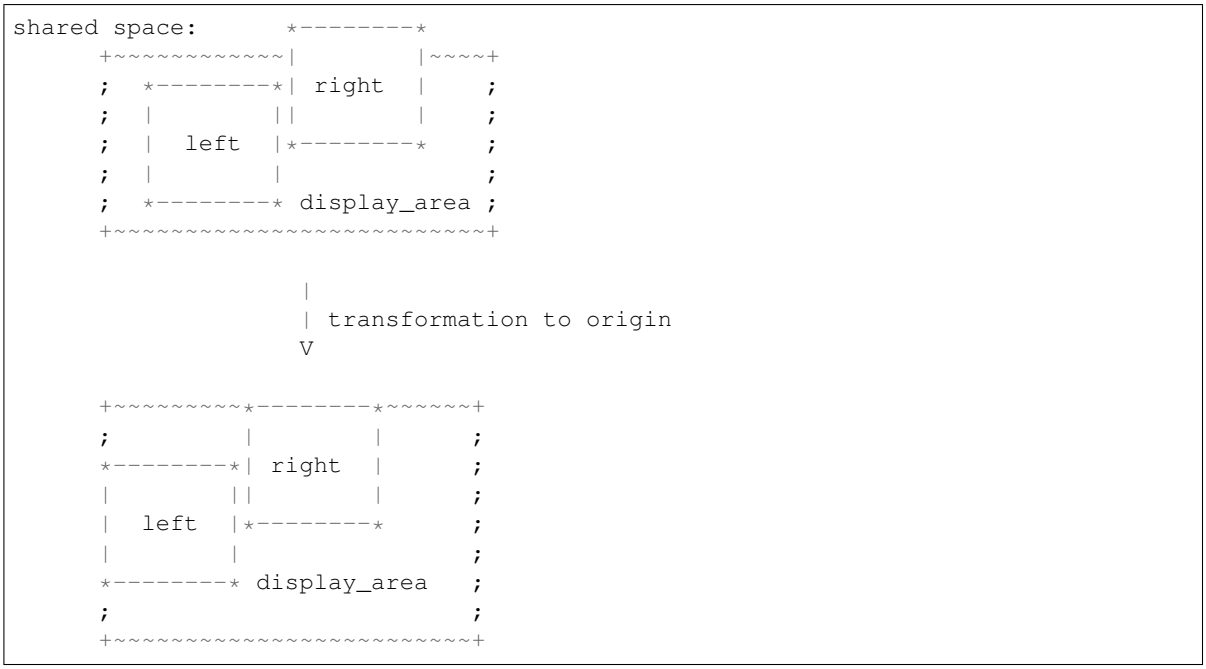
- **size_left** (*tuple*) – Size *(width, height)* of the left image.
- **size_right** (*tuple*) – Size *(width, height)* of the right image.
- **homo_left** (*ndarray*) – An homography *(3,3)* which is used to transform the left image.
- **homo_right** (*ndarray*) – An homography *(3,3)* which is used to transform the right image.

Returns – **xmin** (float) – Minimal x value of both images after transformation. – **ymin** (float) – Minimal y value of both images after transformation. – **xmax** (float) – Maximal x value of both images after transformation. – **ymax** (float) – Maximal x value of both images after transformation.

get_transform_to_origin_mat (*xmin*, *ymin*)

Determine homography matrix to align 'shared_space' to display area origin.

Example



Parameters

- **xmin** (*float*) – Minimal x value of images in 'shared space'.
- **ymin** (*float*) – Minimal y value of images in 'shared space'.

Returns *(3,3)* homography to align 'shared space' the to the origin of display area.

Return type ndarray

See also:

- *get_boundaries()*

add_alpha_channel(*image*)
Add alpha channel to image for transparent areas.

Parameters **image** (*ndarray*) – Image of shape *(M,N)* (black/white), *(M,N,3)* (BGR) or *(M,N,4)* already with alpha channel.

Returns `image` extended with alpha channel

Return type ndarray

form_rectangle(*width*, *height*)
Return a rectangle represented by 4 points ndarray *(4,2)*.

The starting point is the Origin and the points are sorted in clockwise order.

Parameters

- **width** (*float*) – Width of the rectangle.
- **height** (*float*) – Height of the rectangle.

Returns Rectangle represented by 4 points as ndarray *(4,2)*.

Return type ndarray

sort_pts(*points*)
Sort points as convex quadrilateral.

Sort points in clockwise order, so that they form a convex quadrilateral.

Example

```
pts:                 sorted_pts:
     x   x                        A---B
                  --->           /     \
   x       x                    D-------C
```

Parameters **points** (*ndarray*) – Array of points *(N,2)*.

Returns Clockwise ordered `points` *(N,2)*, where the most up left point is the starting point.

Return type ndarray

raw_estimate_rect(*points*)
Abstract an rectangle from an convex quadrilateral.

The convex quadrilateral is defined by `Points`. The points must be sorted in clockwise order where the most up left point is the starting point. (see sort_pts)

Example

```
points:              rectangled points:
     A---B                       A'------B'
    /     \        --->         |        |
   D-------C                     D'------C'
```

The dimension of the rectangle is estimated in the following manner: `|A'B'|=|D'C'|=max(|AB|,|DC|)` and `|A'D'|=|B'C'|=max(|AD|,|BC|)`

Parameters **points** (*ndarray*) – Array of clockwise ordered points *(4,2)*, where most up left point is the starting point.

Returns 'Rectangled' points (the rectangle is aligned to the origin).

Return type ndarray

harmonize_rects(*rect_a*, *rect_b*)

Harmonize two rectangles in their vertical dimension.

Example

```
rect_a:    rect_b:        harm_rect_a:       harm_rect_b:

             W-----X          A'--------------B'    W'----X'
 A-----B     |     |          |                |     |     |
 |     |     |     |    -->   |                |     |     |
 D-----C     |     |          |                |     |     |
             Z-----Y          D'--------------C'    Z'----Y'
```

Parameters

- **rect_a** (*ndarray*) – Array of clockwise ordered points *(4,2)*, where most up left point is the starting point.
- **rect_b** (*ndarray*) – Same as `rect_a`

Returns

- **harm_rect_a** (ndarray) – Harmonized version of `rect_a`
- **harm_rect_b** (ndarray) – Harmonized version of `rect_b`

angles_to_points

Calculate point representations of angles.

The angle point representations `points_reprs` are calculated in dependency of the `angle_center` and the ray starting from this center, which is perpendicular to the right border. Positive angles will be interpreted as clockwise rotation.

Example

```
angle_center
      *--------x-Axis------>
       \          |
        \ angle /
         \      /
          \ --´
           \
points_repr *
             \
              v
```

Parameters

- **angle_centers** (*ndarray*) – The centers of the `angles`. *(N,2)*
- **angles** (*ndarray*) – Angles in rad (length *(N,)*).
- **distance** (*int*) – The distance between the `angle_centers` and the point representations.

Returns

- **points_repr** (ndarray) – Angles represented by points. *(N,2)*

See also:

- *points_to_angles()*

points_to_angles(*angle_centers*, *points_repr*)

Convert angle point representation back to normal angle.

This function is the inverted version of *angles_to_points()*.

Parameters

- **angle_centers** (*ndarray*) – The centers of the `angles`. *(N,2)*
- **points_repr** (*ndarray*) – Angles represented by points. *(N,2)*

Returns Angles in rad *(N,)*

Return type ndarray

See also:

- *angles_to_points()*

get_ratio_px_to_mm(*start_point*, *end_point*, *distance_mm*)

Return ratio between pixel and millimetre.

The function calculates the distance of two points (`start_point`, `end_point`) in pixels and then calculates ratio using the distance in pixels and the distance in mm `distance_mm`.

Parameters

- **start_point** (*ndarray*) – Start point of the reference Line Segment *(2,)*
- **end_point** (*ndarray*) – End point of the reference Line Segment *(2,)*
- **distance_mm** (*float*) – The distance between the `start_point` and `end_point` of the line segment in real world in mm.

Returns The ratio between px and mm (the length of 1px in mm).

Return type float

3.5 Picking

3.5.1 Picker

Initialise a GUI to pick various points on images.

This Module provides a class to initialise a GUI, to pick various points on one or multiple images.

class PointPicker
Bases: object

GUI for picking points.

pick(*images*, *all_pts=True*)
Initialise a GUI to pick points on multiple images.

A matplot GUI will be initialised, where the user can pick multiple points on the **N** images. Afterwards the *PointPicker* will return **N** ndarrays, which holds the coordinates of the marked points. Each ndarray holds the points for one image.

Parameters

- **images** (*list (ndarray)*) – List of images (ndarray)
- **all_pts** (*bool*) – If True all points will be returned and else just 'selected' points will be returned.

Returns Returns a List of length **N**, where each cell contains a ndarray *(M,2)*, which holds the coordinates of the *M* marked points per image.

Return type list(ndarray)

3.5.2 Draggables

This module contains draggable objects for the matplotlib GUI.

The objects are used to mark specific points on an image. The module also contains a special list to save these objects. This list is extended with special functions to get the coordinates of the marked points.

class DraggableMark(*mark*, *img=None*)
Bases: object

Defines marks which can be dragged by mouse.

The placed mark can be dragged by simple left click and can be **refined** by pressing the *key_refine* specific button and they can be marked as **selected** by pressing *key_select*.

key_refine = 'r'

key_select = 's'

get_coordinate()
Return the current coordinate of the mark.

Returns center of the mark *(2,)*.

Return type ndarray

connect()
Connect to the mark to various Events.

disconnect()
Disconnect all the stored connection ids.

class DraggableMarkList(**args*)
Bases: list

Extended List with some extra functions for *DraggableMark* objects.

get_points(*all_pts=True*)
Convert the list of *DraggableMark* objects to a ndarray holding just coordinates.

Parameters **all_pts** (*bool*) – if `True` function returns all coordinate. If `False` function return just coordinates form *DraggableMark* objects marked as selected.

Returns array that contains just the coordinates of the *DraggableMark* objects of the list.

Return type ndarray

3.6 Visualisation

This module contains different functions to draw coordinates and orientations on images.

draw_arrows(*img*, *positions*, *angles*, *color=(0, 0, 255)*, *line_width=6*, *arrow_length=150*)
Draw arrows from positions in angle direction (clockwise).

(The 0°-Angle is the x-Axis.)

Parameters

- **img** (*ndarray*) – Image (min. 3 channel) to draw on.
- **positions** (*ndarray*) – The points the arrows starts from. *(N,2)*
- **angles** (*ndarray*) – Angles in rad (length *(N,)*).

draw_circles(*img*, *centres*, *radius=32*, *color=(0, 0, 255)*, *line_width=6*)
Draw circles around positions.

Parameters

- **img** (*ndarray*) – Image (min. 3 channel) to draw on.
- **centres** (*ndarray*) – The centres of the circles. *(N,2)*
- **radius** – Radius of the circles.

draw_marks(*img*, *positions*, *color=(0, 0, 255)*, *marker_types=<MagicMock id='140344249181968'>*)
Draw cross marks on position.

Parameters

- **img** (*ndarray*) – Image (min. 3 channel) to draw on.
- **positions** (*ndarray*) – The points to mark. *(N,2)*

draw_complex_marks(*img*, *centres*, *angles*, *color=(0, 0, 255)*, *marker_types=<MagicMock id='140344254280200'>*)
Draw more complex marks, with circles, marked centres and arrows for angles/direction.

Parameters

- **img** (*ndarray*) – Image (min. 3 channel) to draw on.
- **centres** (*ndarray*) – The centres of the marks and starting points of arrows. *(N,2)*
- **angles** (*ndarray*) – Angles in rad (length *(N,)*).

draw_grid(*image*, *origin*, *ratio_px_mm*, *step_size_mm=8*)
Draw a grid with axes in mm on the image.

Parameters

- **image** (*ndarray*) – Image to draw on.
- **origin** (*ndarray*) – The orgin of the grid / axes.
- **ratio_px_mm** – Ratio to convert pixel to mm.

- **step_size_mm** – The (step) distance between the grid lines.

3.7 IO-Utils

This module provides different file handlers to load and save the needed data for the *core.Surveyor*.

class FileHandler
Bases: object

Abstract base class of FileHandler.

visit_surveyor(*surveyor*)

save(*path*)
Save data to file.

load(*path*)
Load data from file.

class NPZHandler
Bases: *bb_stitcher.io_utils.FileHandler*

save(*path*)
Save *core.Surveyor* data to numpy file '.npz'.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

load(*path*)
Load *core.Surveyor* data to numpy file '.npz'.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

class CSVHandler
Bases: *bb_stitcher.io_utils.FileHandler*

save(*path*)
Save *core.Surveyor* data to csv file '.csv'.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

load(*path*)
Load *core.Surveyor* data to csv file '.csv'.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

class JSONHandler
Bases: *bb_stitcher.io_utils.FileHandler*

save(*path*)
Save *core.Surveyor* data to json file '.json'.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

load(*path*)
Load *core.Surveyor* data to json file '.json'.

Parameters **path** (*str*) – Path of the file, which holds the needed data.

get_file_handler(*path*)
Returns FileHandler in dependency of file path extension.

Parameters **path** (*str*) – Path of the file.

Returns file handler for load and save data from surveyor.

Return type *FileHandler*

3.8 Measure

This module is used to measure nearly planar objects on a surface.

The surface must be parallel to the image plane.

get_ratio(*image*)
Determine the ratio to convert from pixel to mm.

The user must select two points on the image. The selected points, will be used to determine the ratio between px and mm.

Parameters **image** (*ndarray*) – Reference image.

Returns Ratio to convert pixel to mm.

Return type float

get_origin(*image*)
Determine origin of the image.

The user must select one point on the image. The selected point will be used to define the origin of the image, this will assure that mapped coordinates will always be in relation to the origin and not to the image.

Parameters **image** (*ndarray*) – Reference image.

Returns origin (2,)

Return type ndarray

CHAPTER 4

Indices and tables

- genindex
- modindex
- search

Python Module Index

b

Index

M

N

P

R

S

V