stitch2d

Release 1.2

Adam Mansur

Mar 02, 2023

CONTENTS

stitch2D is a Python script that stitches a two-dimensional grid of tiles into a mosaic. It was originally developed for stitching together images collected on various microscopes in the Department of Mineral Sciences at the Smithsonian National Museum of Natural History.

When tiles are stitched together by stitch2d, they are translated, not rotated, resized, or warped. As a result, stitch2d requires all images to be the same size and orientation. Images must overlap, although they don't necessarily need to be arranged in a grid.

Source code for this project is located on GitHub.

CHAPTER

ONE

CONTENTS

1.1 User guide

1.1.1 Install

Install stitch2d with pip:

```
pip install stitch2d
```

Or install from the GitHub repository using git and pip:

```
git clone https://github.com/adamancer/stitch2d
cd stitch2d
pip install .
```

1.1.2 Quick start

The following code can be used to create and save a mosaic:

```
from stitch2d import create_mosaic


mosaic = create_mosaic("/path/to/tiles")

try:
    mosaic.load_params()
except FileNotFoundError:
    mosaic.downsample(0.6)
    mosaic.align()
    mosaic.reset_tiles()
    mosaic.save_params()

mosaic.smooth_seams()
mosaic.save("mosaic.jpg")
```

A simple stitching workflow is also available from the command line. To create a smoothed mosaic and save it as a JPEG, run:

```
stitch2d path/to/tiles --smooth -output mosaic.jpg
```

For more information about using this command, including available parameters, run:

```
stitch2d --help
```

1.1.3 Overview

stitch2d includes two classes that can be used to create mosaics from a list of tiles:

- *Mosaic*, which incorporates no information about how the tiles in the mosaic are arranged
- *StructuredMosaic*, which arranges the tiles into a grid based on parameters supplied by the user

You can also use *create_mosaic()*, as above, which accepts the same arguments as *StructuredMosaic*. This function returns a *StructuredMosaic* if grid parameters are provided or can be inferred from the filenames of the tiles or a *Mosaic* if not.

Mosaic

Since *Mosaic* doesn't know anything about the tile structure, it can be slow, especially for large grids where lots of tiles need to be compared. It's almost always faster to use *StructuredMosaic* where possible.

Initialize a *Mosaic* by pointing it to the directory where the tiles of interest live:

```
from stitch2d import Mosaic

mosaic = Mosaic("/path/to/tiles")
```

Mosaic also includes a class attribute, *num_cores*, to specify how many cores it should use when aligning and stitching a mosaic. By default, it uses one core. Modify this value with:

```
Mosaic.num_cores = 2
```

Even when using multiple cores, detecting and extracting features can be time consuming. One way to speed up the process is to reduce the resolution of the tiles being analyzed:

```
mosaic.downsample(0.6)  # downsamples all tiles larger than 0.6 mp
```

Alternatively you can resize the tiles without the size check:

```
mosaic.resize(0.6)      # resizes all tiles to 0.6 mp
```

You can then align the smaller tiles:

```
mosaic.align()
```

In either case, you can restore the full-size images prior to stitching the mosaic together:

```
mosaic.reset_tiles()
```

Sometimes brightness and contrast can vary significantly between adjacent tiles, producing a checkerboard effect when the mosaic is stitched together. This can be mitigated in many cases using *smooth_seams()*, which aligns brightness/contrast between neighboring tiles by comparing areas of overlap:

```
mosaic.smooth_seams()
```

Once the tiles have been positioned, the mosaic can be viewed:

```
mosaic.show()
```

Or saved to a file:

```
mosaic.save("mosaic.tif")
```

Or returned as a numpy array if you need more control over the final mosaic:

```
arr = mosaic.stitch()
```

The default backend, opencv, orders color channels as BGR. You may want to reorder the color channels before working with the image in a different program. To get an RGB image from a BGR image, use:

```
arr = arr[...,::-1].copy()
```

New in 1.1: Or specify the desired channel order when stitching:

```
arr = mosaic.stitch("RGB")
```

Once the tiles are positioned, their locations are stored in the *params* attribute, which can be saved as JSON:

```
mosaic.save_params("params.json")
```

Those parameters can then be loaded into a new mosaic if needed:

```
mosaic.load_params("params.json")
```

StructuredMosaic

StructuredMosaic allows the user to specify how the tiles in the mosaic should be arranged. For tilesets of known structure, it is generally faster but otherwise works the same as *Mosaic*. Initialize a structured mosaic with:

```
from stitch2d import StructuredMosaic

mosaic = StructuredMosaic(
    "/path/to/tiles",
    dim=15,                  # number of tiles in primary axis
    origin="upper left",     # position of first tile
    direction="horizontal",  # primary axis (i.e., the direction to traverse first)
    pattern="snake"          # snake or raster
  )
```

For large tilesets where adequate-but-imperfect tile placement is acceptable, *StructuredMosaic* can use its knowledge of the tile grid to quickly build a mosaic based on the positions of only a handful of tiles:

```
# Stop aligning once 5 tiles have been successfully placed
mosaic.align(limit=5)

# Build the rest of the mosaic based on the positioned tiles. If from_placed
# is True, missing tiles are appended to the already positioned tiles. If
# False, a new mosaic is calculated from scratch.
mosaic.build_out(from_placed=True)
```

The *build_out()* method can also be used to ensure that all tiles (including those that could not be placed using feature matching) appear in the final mosaic. The primary disadvantage of this method is that the placement of those tiles is less precise.

1.1.4 Beyond 8-bit images

New in 1.2: The Tile class now includes a *prep_imdata()* method that can be used to tweak the image data being used to align the mosaic. When using the default OpenCVTile class, this method creates an 8-bit copy of the image data to use for feature detection and matching while retaining the original data to use when building the mosaic.

The default behavior of *prep_imdata()* is simplistic. To customize it, use a subclass. For example, the default method scales the intensities of the original data based on the maximum intensity found in the array. For images with a small number of extremely bright pixels, this can yield unusably dim images. A better approach may be to use `np.percentile()`:

```
import numpy as np

class MyTile(OpenCVTile):

    def prep_imdata(self):
        imdata = self.imdata - self.imdata.min()
        return  np.uint8(255 * imdata / np.percentile(imdata, 99))

mosaic = create_mosaic("path/to/tiles", tile_class=MyTile)
```

1.1.5 Similar tools

The opencv package includes powerful tools for stitching 2D and 3D images). Much of that functionality has been ported to Python as the stitching package, which streamlines the opencv API and includes a useful tutorial. I didn't have any luck getting it to work consistently with microscope tilesets, but it includes advanced features missing from this package (lens corrections, affine transformations beyond simple translation, etc.) and can be configured to work with 2D images. It's definitely worth a look for tilesets more complex than the simple case handled here.

Fiji also includes a 2D/3D stitching tool.

1.2 API

Stitches a 2D grid of images into a mosaic

1.2.1 stitch2d.mosaic

Reads and stitches images from a 2D grid into a mosaic

class stitch2d.mosaic.**Mosaic**(*path_or_tiles*, *tile_class=None*)

Bases: `object`

Stitches a mosaic from a list of tiles

grid

tiles arranged into a grid

Type list

shape
shape of the mosaic as (height, width[, channels])

Type tuple

size
number of tiles in the mosaic

Type tuple

tile_class
class to use for tiles in the mosaic

Type class

num_cores = 1
Number of cores to use when processing images

Type int

__init__(*path_or_tiles*, *tile_class=None*)
Initializes a mosaic from a list of tiles

Parameters

- **path_or_tiles** (*str or list-like*) – either the path to a directory containing tiles, a list of Tiles, or a list of strings or arrays that can be used to create a Tile
- **tile_class** (*class*) – class to use for tiles in the mosaic. Defaults to OpenCVTile.

property tiles
Gets a flattened list of all tiles in grid order

property placed
Calculates number of tiles that have been placed in the mosaic

property params
Summarizes parameters needed to stitch mosaic

property detector
Gets a copy of the detector used to align tiles in the mosaic

property matcher
Gets a copy of the matcher used to align tiles in the mosaic

Only defined if using OpenCV.

property pool
Returns a shared joblib pool, creating it if needed

placeholder(*tile=None*, *fill_value=0*)
Creates a placeholder tile to fill in gaps in the mosaic

Parameters

- **tile** (Tile) – tile to base the placeholder on. If not given, uses the first tile in the tiles property.
- **fill_value** (*float or int*) – fill value

Returns placeholder tile filled with provided value

Return type *Tile*

bounds()
Calculates bounds of the mosaic comprising the placed tiles

Returns bounds of tile as (y1, x1, y2, x2)

Return type tuple

`copy`()
Creates a copy of the mosaic based on the grid

Using the grid instead of the list of tiles allows the grid-building step to be skipped when the copy is initialized.

Returns copy of the mosaic

Return type *Mosaic*

`save_params`(*path='params.json'*)
Saves coordinates for placed tiles

Parameters **path** (*str*) – path to the JSON file

`load_params`(*path_or_obj='params.json'*)
Loads coordinates for placed tiles at full scale

Parameters **path_or_obj** (*str or dict*) – path to the JSON file or param dict from another mosaic

Raises

- **FileNotFoundError** – thrown if input is path and path not found
- **ValueError** – thrown if JSON can't be decoded or does not match this mosaic

`reset_tiles`()
Reloads tiles at their full resolution

`resize`(*size_or_shape*, **args*, ***kwargs*)
Rescales all tiles in the mosaic using size or shape

Parameters

- **size_or_shape** (*int or tuple of ints*) – size in megapixels or shape of resized image
- ***args** – any argument accepted by the resize function used by the tile class
- ****kwargs** – any keyword argument accepted by the resize function used by the tile class

`downsample`(*size_or_shape*, **args*, ***kwargs*)
Downsamples all tiles in the mosaic using the given size or shape

Parameters

- **size_or_shape** (*int or tuple of ints*) – size in megapixels or shape of resized image
- ***args** – any argument accepted by the resize function used by the tile class
- ****kwargs** – any keyword argument accepted by the resize function used by the tile class

`detect_and_extract`()
Detects and extracts features in tiles

`align`(*origin=None*, *limit=None*, ***kwargs*)
Builds a mosaic by checking each tile against all others

Parameters

- **origin** (Tile) – the tile around which to build the mosaic. If not given, method will select the tile with the largest number of features.
- **limit** (*int*) – the number of tiles that must be successfully placed before the method finishes. If not given, the method will continue until it runs out of adjacent tiles with matching features. Setting a limit allows a decent mosaic to be created quickly.
- **kwargs** – any keyword argument accepted by the align_to method on the Tiles comprising this mosaic

build_out(**args*, ***kwargs*)
Warns user that build_out is not implemented in Mosaic class

Use StructuredMosaic instead to get this functionality.

smooth_seams(*origin=None*)
Smooths intensities at seams between tiles

Parameters **origin** (Tile) – starting tile

stitch(*channel_order=None*)
Stitches mosaic using either placed tiles or row/col of tiles

Parameters **channel_order** (*str*) – order of the three color channels in the stitched array, for example, RGB. Uses the backend order if not given, which can give unexpected results (for example, OpenCV uses BGR).

Returns

Return type numpy.ndarray

save(*path*)
Saves mosaic to path

Parameters **path** (*str*) – file path

show(**args*, ***kwargs*)
Shows the mosaic

class stitch2d.mosaic.**StructuredMosaic**(*path_or_tiles*, *tile_class=None*, *dim=None*, *origin='upper left'*, *direction='horizontal'*, *pattern='raster'*)
Bases: *stitch2d.mosaic.Mosaic*

Stitches a mosaic from a list of tiles with a known structure

__init__(*path_or_tiles*, *tile_class=None*, *dim=None*, *origin='upper left'*, *direction='horizontal'*, *pattern='raster'*)
Initializes a structured mosaic from a list of tiles

Parameters

- **path_or_tiles** (*str or list-like*) – either the path to a directory containing tiles, a list of Tiles, or a list of strings or arrays that can be used to create a Tile
- **tile_class** (*class*) – class to use for tiles in the mosaic. Defaults to OpenCVTile.
- **dim** (*tuple or int*) – either the shape of the mosaic as (height, width) or the number of tiles in the direction traversed first, that is, the number of columns (if horizontal) or number of rows (if vertical)
- **origin** (*str*) – the position of the first tile in the mosaic. One of "upper left", "upper right", "lower left", or "lower right".
- **direction** (*str*) – direction to traverse first when building the mosaic. Either "horizontal" or "vertical".

- **pattern** (*str*) – whether the grid is rastered or snaked. Either "raster" or "snake".

align(*origin=None*, *limit=None*, ***kwargs*)

Builds a mosaic outward from a single tile using feature matching

Parameters

- **origin** (Tile) – the tile around which to build the mosaic. If not given, method will select a tile near the center of the mosaic.
- **limit** (*int*) – the number of tiles that must be successfully placed before the method finishes. If not given, the method will continue until it runs out of adjacent tiles with matching features. Setting a limit allows a decent mosaic to be created quickly.
- **kwargs** – any keyword argument accepted by the align_to method on the Tiles comprising this mosaic

build_out(*from_placed=True*, *offsets=None*)

Builds out from already placed tiles using the given offset

Used to complete mosaics that include tiles that were not placed when the mosaic was built, either because the user assigned a limit or because the feature matching algorithm failed to find a home for them.

Parameters

- **from_placed** (*bool*) – if True, unplaced tiles will be tacked onto already placed tiles using the given offsets. If False, a new mosaic will be calculated from scratch using the given offsets.
- **offsets** (*tuple*) – offsets between adjacent tiles as dy_row, dx_row, dy_col, dx_col. If not given, the method will estimate the offsets if any tiles have been placed or will ignore offsets if not.

stitch2d.mosaic.**build_grid**(*items*, *dim*, *origin='upper left'*, *direction='horizontal'*, *pattern='raster'*, *fill_value=None*)

Builds a grid from a list

Parameters

- **items** (*list*) – list to convert to a grid
- **dim** (*tuple or int*) – either the shape of the mosaic as (height, width) or the number of tiles in the direction traversed first, that is, the number of columns (if horizontal) or number of rows (if vertical)
- **origin** (*str*) – the position of the first tile in the mosaic. One of "upper left", "upper right", "lower left", or "lower right".
- **direction** (*str*) – direction to traverse first when building the mosaic. Either "horizontal" or "vertical".
- **pattern** (*str*) – whether the grid is a raster or snake
- **fill_value** – value used to fill missing items in a ragged grid

Returns List of rows in the grid

Return type list

stitch2d.mosaic.**create_mosaic**(*path_or_tiles*, *tile_class=None*, *dim=None*, *origin=None*, *direction=None*, *pattern=None*)

Creates a mosaic

See StructuredMosaic for available parameters.

Returns tiles as either a structured or unstructured mosaic

Return type *Mosaic* or *StructuredMosaic*

stitch2d.mosaic.**is_grid**(*items*)
Tests if an iterable looks like a grid

Parameters **items** (*list-like*) – list of items

Returns True if tiles look like a grid, False if not

Return type bool

1.2.2 stitch2d.tile

Reads and helps place a single image from a 2D grid

class stitch2d.tile.**Tile**(*data*, *detector='sift'*)
Bases: object

An image tile in a mosaic

source
the original data used to created the tile. Either the path to an image file or an array with 1, 3, or 4 channels.

Type str or array-like

imdata
image data

Type numpy.ndarray

id
a UUID uniquely identifying the tile

Type str

row
the index of the row where the tile appears in the mosaic

Type int

col
the index of the column where the tile appears in the mosaic

Type int

y
the y coordinate of the image within the mosaic

Type float

x
the x coordinate of the image within the mosaic

Type float

channel_order
the order of the three color channels in the image, e.g., RGB or BGR

Type str

scale
the current scale of the tile relative to the original image

Type float

features_detected
whether any features were detected in this image

Type bool

descriptors
list of descriptors found in this image

Type numpy.ndarray

keypoints
list of coordinates of descriptors found in this image

Type numpy.ndarray

detectors = {}
maps strings to a subclass-specific feature detector

Type dict

matchers = {}
maps strings to a subclass-specific feature matcher

Type dict

__init__(*data*, *detector='sift'*)
Initializes a mosaic from a list of tiles

Parameters

- **data** (*str or numpy.ndarray*) – path to an image file or an array of image data
- **detector** (*str*) – name of the detector used to find/extract features. Currently only sift is supported.

property detector
Gets the detector used to align this tile to another tile

property matcher
Gets the matcher used to align this tile to another tile

property height
Gets the height of the image in pixels

property width
Gets the width of the image in pixels

property channels
Gets the number of channels in the image

property channel_axis
Gets the index where channel info is stored

property dtype
Gets the dtype of the image

property shape
Gets the shape of the image

property size
Gets the size of the image

property mp
Gets the size of the image in megapixels

property placed
Whether the tile has been assigned coordinates in the mosaic

load_imdata()
Loads copy of source data

Returns copy of source data

Return type numpy.ndarray

copy()
Creates a copy of the tile

Parameters **grid** (*list of lists*) – grid from the mosaic containing the tile

Returns copy of the tile

Return type *Mosaic*

bounds(*as_int=False*)
Calculates the position of the tile within the mosaic

Parameters **as_int** (*bool*) – whether bounds are converted to integers before returning

Returns bounds of the image in the mosaic coordinate system as (y1, x1, y2, x2)

Return type tuple

neighbors()
Finds adjacent tiles

Parameters

- **y** (*int*) – row index
- **x** (*int*) – column index

Returns neighboring tiles keyed to direction (top, right, bottom, left)

Return type dict

convert_mosaic_coords(*y1*, *x1*, *y2*, *x2*)
Converts mosaic coordinates to image coordinates

Returns mosaic coordinates translated to image coordinates

Return type tuple

update(*other*)
Updates attributes to match another tile

Parameters **other** (Tile) – a tile with attributes to copy over to this one

crop(*box*, *convert_mosaic_coords=True*)
Crops tile to the given box

Parameters

- **box** (*tuple*) – box to crop to as (y1, x1, y2, x2)
- **convert_mosaic_coords** (*bool*) – whether to convert the given coordinates from mosaic to image coordinates

Returns image data cropped to the given box

Return type numpy.ndarray

intersection(*other*)
Finds the intersection between two placed tiles

Parameters **other** (Tile) – an adjacent tile that has already been placed in the mosaic

Returns the overlapping portion of both tiles

Return type tuple of Tile

intersects(*other*)
Tests if two placed tiles intersect

Parameters **other** (Tile) – an adjacent tile that has already been placed in the mosaic

Returns True if tiles intersect, False otherwise

Return type bool

reset()
Restores original image and resets coordinate and feature attrs

Returns the original tile updated to restore the original image data

Return type *Tile*

match_gamma_to(*other*)
Scales intensity to match intersecting region of another tile

Parameters **other** (Tile) – a tile that intersects this one

Returns the original tile with its intensity modified

Return type *Tile*

draw(*others=None*)
Creates an image from the provided tiles

Parameters **others** (*list of Tiles*) – a list of tiles to include in the new image. Only tiles that have been placed will be included.

Returns an image including all provided tiles

Return type numpy.ndarray

save(*path*, *others=None*)
Saves an image created from the provided tiles

Parameters

- **path** (*str*) – file path
- **others** (*list of Tiles*) – a list of tiles to include in the new image. Only tiles that have been placed will be included.

show(*others=None*)
Shows an image created from the provided tiles

Parameters **others** (*list of Tiles*) – a list of tiles to include in the new image. Only tiles that have been placed will be included.

gray()
Returns copy of image converted to grayscale

Returns grayscale version of the original iamge

Return type numpy.ndarray

resize(*size_or_shape*, **args*, ***kwargs*)
Resizes image to a given size or shape

Parameters

- **size_or_shape** (*float, int, or tuple of ints*) – size in megapixels or shape of resized image
- ***args** – any argument accepted by the resize function used by the subclass
- ****kwargs** – any keyword argument accepted by the resize function used by the subclass

downsample(*size_or_shape*, **args*, ***kwargs*)
Downsamples image to a given size or shape if smaller than original

Parameters

- **size_or_shape** (*float, int, or tuple of ints*) – size in megapixels or shape of resized image as (height, width)
- ***args** – any argument accepted by the resize method of the subclass
- ****kwargs** – any keyword argument accepted by the resize method of the subclass

Returns the original tile downsampled to the given size or shape

Return type *Tile*

prep_imdata()
Returns a copy of the tile data suitable for feature detection

Users may wish to create a subclass with a custom version of this method, for example, to scale intensities, enhance contrast, or select image data from an array that include additional bands.

Returns copy of image data

Return type numpy.ndarray

detect_and_extract(**args*, ***kwargs*)
Detects and extracts features within the tile

Parameters

- ***args** – any argument accepted by the feature detection method on the detector
- ****kwargs** – any keyword argument accepted by the feature detection method on the detector

align_to(*other*, ***kwargs*)
Aligns tile to another, already placed tile

Parameters **other** (Tile) – a tile that has already been placed in the mosaic

static backend_save(*path*, *im*)
Saves image to path using the tile backend

Parameters

- **path** (*str*) – file path
- **im** (*numpy.ndarray*) – image data

static backend_show(*im*)
Shows an image using the tile backend

Parameters **im** (*numpy.ndarray*) – image data

class stitch2d.tile.**OpenCVTile**(*data*, *detector='sift'*, *matcher='flann'*)
Bases: *stitch2d.tile.Tile*

An image tile in a mosaic loaded and manipulated using OpenCV

See Tile for available attributes.

detectors = {'sift': <stitch2d.tile._DefaultInstance object>}
maps strings to a subclass-specific feature detector

Type dict

matchers = {'bf': <stitch2d.tile._DefaultInstance object>, 'flann': <stitch2d.tile._DefaultInstance object>}
maps strings to a subclass-specific feature matcher

Type dict

__init__(*data*, *detector='sift'*, *matcher='flann'*)
Initializes a mosaic from a list of tiles

Parameters

- **data** (*str or numpy.ndarray*) – path to an image file or an array of image data
- **detector** (*str*) – name of the detector used to find/extract features. Currently only sift is supported.

load_imdata()
Loads copy of source data

Returns copy of source data

Return type numpy.ndarray

gray()
Returns copy of image converted to grayscale

Returns grayscale version of the original iamge

Return type numpy.ndarray

resize(*size_or_shape*, **args*, ***kwargs*)
Resizes image to a given size or shape

Parameters

- **size_or_shape** (*float, int, or tuple of ints*) – size in megapixels or shape of resized image as (height, width)
- ***args** – any argument accepted by cv.resize
- ****kwargs** – any keyword argument accepted by cv.resize

Returns the original tile resized to the given size or shape

Return type *OpenCVTile*

prep_imdata()
Returns a copy of the tile data suitable for feature detection

The built-in version of this method checks if the imdata attribute is an 8-bit array, returning a copy if so. Otherwise, it rescales the intensities and returns an 8-bit copy of the array. The conversion is simplistic, and users may prefer to create a subclass with a custom version of this method instead, for example, to scale intensities, enhance contrast, or select image data from an array that include additional bands.

Returns copy of image data as an 8-bit array

Return type numpy.ndarray

detect_and_extract(**args*, ***kwargs*)
Detects and extracts features within the tile

Parameters

- ***args** – any argument accepted by the detect_and_extract method on the detector
- ****kwargs** – any keyword argument accepted by the detect_and_extract method on the detector

Returns the original tile updated with features and keypoints

Return type *OpenCVTile*

align_to(*other*, ***kwargs*)
Aligns tile to another, already placed tile

Parameters **other** (Tile) – a tile that has already been placed in the mosaic

Returns the original tile updated with x and y coordinates

Return type *OpenCVTile*

static backend_save(*path*, *im*)
Saves image to path using OpenCV

Parameters

- **path** (*str*) – file path
- **im** (*numpy.ndarray*) – image data

static backend_show(*im*, *title='OpenCV Image'*)
Shows an image using OpenCV

Parameters **im** (*numpy.ndarray*) – image data

class stitch2d.tile.**ScikitImageTile**(*data*, *detector='sift'*)
Bases: *stitch2d.tile.Tile*

An image tile in a mosaic loaded and manipulated using scikit-image

See Tile for available attributes.

detectors = {'sift': <stitch2d.tile._DefaultInstance object>}
maps strings to a subclass-specific feature detector

Type dict

__init__(*data*, *detector='sift'*)
Initializes a mosaic from a list of tiles

Parameters

- **data** (*str or numpy.ndarray*) – path to an image file or an array of image data
- **detector** (*str*) – name of the detector used to find/extract features. Currently only sift is supported.

load_imdata()
Loads copy of source data

Returns copy of source data

Return type numpy.ndarray

`gray`()
Returns copy of image converted to grayscale

Returns grayscale version of the original iamge

Return type numpy.ndarray

`resize`(*size_or_shape*, **args*, ***kwargs*)
Resizes image to a given size or shape

Parameters

- **size_or_shape** (*float, int, or tuple of ints*) – size in megapixels or shape of resized image
- ***args** – any argument accepted by skimage.transform.resize
- ****kwargs** – any keyword argument accepted by skimage.transform.resize

Returns the original tile resized to the given size of shape

Return type *ScikitImageTile*

`detect_and_extract`(**args*, ***kwargs*)
Detects and extracts features within the tile

Parameters

- ***args** – any argument accepted by the detect_and_extract method on the detector
- ****kwargs** – any keyword argument accepted by the detect_and_extract method on the detector

Returns the original tile updated with features and keypoints

Return type *ScikitImageTile*

`align_to`(*other*, ***kwargs*)
Aligns tile to another, already placed tile

Parameters **other** (Tile) – a tile that has already been placed in the mosaic

Returns the original tile updated with x and y coordinates

Return type *ScikitImageTile*

`static backend_save`(*path*, *im*)
Saves image to path using skimage

Parameters

- **path** (*str*) – file path
- **im** (*numpy.ndarray*) – image data

`static backend_show`(*im*)
Shows an image using skimage

Parameters **im** (*numpy.ndarray*) – image data

PYTHON MODULE INDEX

S

INDEX

Symbols

A

B

C

D

F

G

H

I

K

L

M

N

O

P

R

S

T

U

W

X

Y