pystackreg Documentation

Release 0.2.2

Gregor Lichtner, Philippe Thevenaz

Jan 04, 2021

Contents:

CHAPTER 1

pystackreg API reference

class pystackreg.**StackReg**(*transformation*)

Python implementation of the ImageJ/Fiji StackReg plugin (http://bigwww.epfl.ch/thevenaz/stackreg/)

get_matrix()

Get the current transformation matrix

Return type ndarray(3,3) or ndarray(4,4) for bilinear transformation

Returns The transformation matrix

get_points()

Returns the pairs of corresponding points from which the transformation matrix can be calculated.

Return type (ndarray, ndarray)

Returns (Point coordinates of reference image, Point coordinates of image to be aligned)

is_registered()

Indicates whether register() was already called and a transformation matrix was calculated

Return type bool

Returns True if a transformation matrix was already calculated

register(*ref*, *mov*)

Registers an image to a reference image: Only the transformation matrix will be calculated, the image will not be transformed (use transform() or register_transform()).

Parameters

- **ref** (*array_like (Ni..., Nj...)*) – Reference image (static)
- **mov** (*array_like (Ni..., Nj...)*) – The image that should be aligned to the reference image

Return type ndarray(3,3)

Returns 2D transformation matrix

register_stack(*img*, *reference='previous'*, *n_frames=1*, *axis=0*, *moving_average=1*, *verbose=False*, *progress_callback=None*)

Register a stack of images (movie). Note that this function will not transform the image but only calculate the transformation matrices. For tranformation use transform_stack() after this function or use register_transform_stack() for a single call.

Parameters

- **img** (*array_like(Ni..., Nj..., Nk...)*) – The 3D stack of images that should be aligned
- **reference** (*string, one of ['previous', 'first', 'mean']*) –
  - *'previous'*: Aligns each frame (image) to its previous frame in the stack
  - *'first:'* Aligns each frame (image) to the first frame in the stack - if n_frames is > 1, then each frame is aligned to the mean of the first n_frames of the stack
  - *'mean'*: Aligns each frame (image) to the average of all images in the stack
- **n_frames** (*int, optional*) – If reference is 'first', then this parameter specifies the number of frames from the beginning of the stack that should be averaged to yield the reference image.
- **axis** (*int, optional*) – The axis of the time dimension
- **moving_average** (*int, optional*) – If moving_average is greater than 1, a moving average of the stack is first created (using a subset size of moving_average) before registration
- **verbose** (*bool, optional*) – Specifies whether a progressbar should be shown using tqdm.
- **progress_callback** (*function, optional*) – A function that is called after every iteration. This function should accept the keyword arguments current_iteration:int and end_iteration:int.

Return type ndarray(img.shape[axis], 3, 3)

Returns The transformation matrix for each image in the stack

register_transform(*ref*, *mov*)

Register and transform an image to a reference image.

Parameters

- **ref** (*ref: array_like (Ni..., Nj...)*) – Reference image (static)
- **mov** (*array_like (Ni..., Nj...)*) – The image that should be aligned to the reference image

Return type ndarray (Ni..., Nj...)

Returns Transformed image - will be of the same shape as the input image (cropping may occur)

register_transform_stack(*img*, *reference='previous'*, *n_frames=1*, *axis=0*, *moving_average=1*, *verbose=False*, *progress_callback=None*)

Register and transform stack of images (movie).

Parameters

- **img** (*array_like(Ni..., Nj..., Nk...)*) – The 3D stack of images that should be aligned
- **reference** (*string, one of ['previous', 'first', 'mean']*) –

- *'previous'*: Aligns each frame (image) to its previous frame in the stack
- *'first:'* Aligns each frame (image) to the first frame in the stack - if n_frames is > 1, then each frame is aligned to the mean of the first n_frames of the stack
- *'mean'*: Aligns each frame (image) to the average of all images in the stack

- **n_frames** (*int, optional*) – If reference is 'first', then this parameter specifies the number of frames from the beginning of the stack that should be averaged to yield the reference image.
- **axis** (*int, optional*) – The axis of the time dimension
- **moving_average** (*int, optional*) – If moving_average is greater than 1, a moving average of the stack is first created (using a subset size of moving_average) before registration
- **verbose** (*bool, optional*) – Specifies whether a progressbar should be shown using tqdm.
- **progress_callback** (*function, optional*) – A function that is called after every iteration. This function should accept the keyword arguments current_iteration:int and end_iteration:int.

Return type ndarray(Ni..., Nj..., Nk...)

Returns The transformed stack

set_matrix(*mat*)

Sets the current transformation matrix

Parameters **mat** (*ndarray(3,3) or ndarray(4,4) for bilinear transformation*) – The transformation matrix

transform(*mov*, *tmat=None*)

Transform an image according to a previous registration.

Either a transformation matrix has to be explicitly supplied or register() has to be called before calling transform().

Parameters

- **mov** (*array_like (Ni..., Nj...)*) – The image that will be transformed
- **tmat** (*ndarray(3,3), optional*) – The transformation matrix

Return type ndarray (Ni..., Nj...)

Returns Transformed image - will be of the same shape as the input image (cropping may occur)

transform_stack(*img*, *axis=0*, *tmats=None*)

Transform a stack after registration.

Parameters

- **img** (*array_like(Ni..., Nj..., Nk...)*) – The 3D stack of images that should be aligned
- **axis** (*int, optional*) – The axis of the time dimension
- **tmats** (*ndarray(img.shape[axis], 3, 3), optional*) – The transformation matrix for each image in the stack

Return type ndarray(Ni..., Nj..., Nk...)

Returns The transformed stack

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

CHAPTER 4

pyStackReg

4.1 Summary

Python/C++ port of the ImageJ extension TurboReg/StackReg written by Philippe Thevenaz/EPFL.

A python extension for the automatic alignment of a source image or a stack (movie) to a target image/reference frame.

4.2 Description

pyStackReg is used to align (register) one or more images to a common reference image, as is required usually in time-resolved fluorescence or wide-field microscopy. It is directly ported from the source code of the ImageJ plugin `TurboReg` and provides additionally the functionality of the ImageJ plugin `StackReg`, both of which were written by Philippe Thevenaz/EPFL (available at http://bigwww.epfl.ch/thevenaz/turboreg/).

pyStackReg provides the following four types of distortion:

- translation
- rigid body (translation + rotation)
- scaled rotation (translation + rotation + scaling)
- affine (translation + rotation + scaling + shearing)
- bilinear (non-linear transformation; does not preserve straight lines)

pyStackReg supports the full functionality of StackReg plus some additional options, e.g., using different reference images and having access to the actual transformation matrices (please see the examples below).

Please note: The bilinear transformation cannot be propagated, as a combination of bilinear transformations does not generally result in a bilinear transformation. Therefore, stack registration/transform functions won't work with bilinear transformation when using "previous" image as reference image. You can either use another reference ("first" or "mean" for first or mean image, respectively), or try to register/transform each image of the stack separately to its respective previous image (and use the already transformed previous image as reference for the next image).

4.3 Installation

The package is available on conda forge and on PyPi.

- Install using **conda**

```
conda install pystackreg -c conda-forge
```

- Install using **pip**

```
pip install pystackreg
```

4.4 Documentation

The documentation can be found on readthedocs:

https://pystackreg.readthedocs.io/

4.5 Usage

The following example opens two different files and registers them using all different possible transformations

```
from pystackreg import StackReg
from skimage import io

#load reference and "moved" image
ref = io.imread('some_original_image.tif')
mov = io.imread('some_changed_image.tif')

#Translational transformation
sr = StackReg(StackReg.TRANSLATION)
out_tra = sr.register_transform(ref, mov)

#Rigid Body transformation
sr = StackReg(StackReg.RIGID_BODY)
out_rot = sr.register_transform(ref, mov)

#Scaled Rotation transformation
sr = StackReg(StackReg.SCALED_ROTATION)
out_sca = sr.register_transform(ref, mov)

#Affine transformation
sr = StackReg(StackReg.AFFINE)
```

(continues on next page)

(continued from previous page)

```
out_aff = sr.register_transform(ref, mov)

#Bilinear transformation
sr = StackReg(StackReg.BILINEAR)
out_bil = sr.register_transform(ref, mov)
```

The next example shows how to separate registration from transformation (e.g., to register in one color channel and then use that information to transform another color channel):

```
from pystackreg import StackReg
from skimage import io

img0 = io.imread('some_multiframe_image.tif')
img1 = io.imread('another_multiframe_image.tif')
# img0.shape: frames x width x height (3D)

sr = StackReg(StackReg.RIGID_BODY)

# register 2nd image to 1st
sr.register(img0[0, :, :], img0[1,:,:])

# use the transformation from the above registration to register another frame
out = sr.transform(img1[1,:,:])
```

The next examples shows how to register and transform a whole stack:

```
from pystackreg import StackReg
from skimage import io

img0 = io.imread('some_multiframe_image.tif') # 3 dimensions : frames x width x height

sr = StackReg(StackReg.RIGID_BODY)

# register each frame to the previous (already registered) one
# this is what the original StackReg ImageJ plugin uses
out_previous = sr.register_transform_stack(img0, reference='previous')

# register to first image
out_first = sr.register_transform_stack(img0, reference='first')

# register to mean image
out_mean = sr.register_transform_stack(img0, reference='mean')

# register to mean of first 10 images
out_first10 = sr.register_transform_stack(img0, reference='first', n_frames=10)

# calculate a moving average of 10 images, then register the moving average to the␣
↪mean of
# the first 10 images and transform the original image (not the moving average)
out_moving10 = sr.register_transform_stack(img0, reference='first', n_frames=10,␣
↪moving_average = 10)
```

The next example shows how to separate registration from transformation for a stack (e.g., to register in one color channel and then use that information to transform another color channel):

```
from pystackreg import StackReg
```

(continues on next page)

(continued from previous page)

```
from skimage import io

img0 = io.imread('some_multiframe_image.tif') # 3 dimensions : frames x width x height
img1 = io.imread('another_multiframe_image.tif') # same shape as img0

# both stacks must have the same shape
assert img0.shape == img1.shape

sr = StackReg(StackReg.RIGID_BODY)

# register each frame to the previous (already registered) one
# this is what the original StackReg ImageJ plugin uses
tmats = sr.register_stack(img0, reference='previous')
out = sr.transform_stack(img1)

# tmats contains the transformation matrices -> they can be saved
# and loaded at another time
import numpy as np
np.save('transformation_matrices.npy', tmats)

tmats_loaded = np.load('transformation_matrices.npy')

# make sure you use the correct transformation here!
sr = StackReg(StackReg.RIGID_BODY)

# transform stack using the tmats loaded from file
sr.transform_stack(img1, tmats=tmats_loaded)

# with the transformation matrices at hand you can also
# use the transformation algorithms from other packages:
from skimage import transform as tf

out = np.zeros(img0.shape).astype(np.float)

for i in range(tmats.shape[0]):
    tform = tf.AffineTransform(matrix=tmats[i, :, :])
    out[i, :, :] = tf.warp(img1[i, :, :], tform)
```

4.6 Author information

This is a port of the original Java code by Philippe Thevenaz to C++ with a Python wrapper around it. All credit goes to the original author:

```
/*=============================================================================
| Philippe Thevenaz
| EPFL/STI/IMT/LIB/BM.4.137
| Station 17
| CH-1015 Lausanne VD
| Switzerland
|
| phone (CET): +41(21)693.51.61
| fax: +41(21)693.37.01
| RFC-822: philippe.thevenaz@epfl.ch
| X-400: /C=ch/A=400net/P=switch/O=epfl/S=thevenaz/G=philippe/
```

(continues on next page)

(continued from previous page)

```
| URL: http://bigwww.epfl.ch/
\=====================================================================*/

/*=====================================================================
| This work is based on the following paper:
|
| P. Thevenaz, U.E. Ruttimann, M. Unser
| A Pyramid Approach to Subpixel Registration Based on Intensity
| IEEE Transactions on Image Processing
| vol. 7, no. 1, pp. 27-41, January 1998.
|
| This paper is available on-line at
| http://bigwww.epfl.ch/publications/thevenaz9801.html
|
| Other relevant on-line publications are available at
| http://bigwww.epfl.ch/publications/
\=====================================================================*/
```

4.7 License

```
You are free to use this software for commercial and non-commercial
purposes. However, we expect you to include a citation or acknowledgement
whenever you present or publish research results that are based
on this software. You are free to modify this software or derive
works from it, but you are only allowed to distribute it under the
same terms as this license specifies. Additionally, you must include
a reference to the research paper above in all software and works
derived from this software.
```

Python Module Index

p

Index