wbia-pyhesaff

Release latest

Nov 11, 2022

Contents:

CHAPTER 1

pyhesaff package

1.1 Submodules

1.2 pyhesaff.__main__ module

pyhesaff.__main__.**main**()

1.3 pyhesaff._pyhesaff module

The python hessian affine keypoint module

Command Line: python -m pyhesaff detect_feats –show –siftPower=0.5 –maxBinValue=-1 python -m pyhesaff detect_feats –show python -m pyhesaff detect_feats –show –siftPower=0.5,

pyhesaff._pyhesaff.**adapt_scale**(*img_fpath*, *kpts*)

pyhesaff._pyhesaff.**alloc_kpts**(*nKpts*)

pyhesaff._pyhesaff.**alloc_patches**(*nKpts*, *size=41*)

pyhesaff._pyhesaff.**alloc_vecs**(*nKpts*)

pyhesaff._pyhesaff.**argparse_hesaff_params**()

pyhesaff._pyhesaff.**detect_feats**(*img_fpath*, *use_adaptive_scale=False*, *nogravity_hack=False*, ***kwargs*)

driver function for detecting hessian affine keypoints from an image path. extra parameters can be passed to the hessian affine detector by using kwargs.

Parameters

- **img_fpath** (*str*) – image file path on disk
- **use_adaptive_scale** (*bool*) –

- **nogravity_hack** (*bool*) –

Kwargs: numberOfScales (int) : default=3 threshold (float) : default=5.33333333333 edgeEigenValueRatio (float) : default=10.0 border (int) : default=5 maxIterations (int) : default=16 convergenceThreshold (float) : default=0.05 smmWindowSize (int) : default=19 mrSize (float) : default=5.19615242271 spatialBins (int) : default=4 orientationBins (int) : default=8 maxBinValue (float) : default=0.2 initialSigma (float) : default=1.6 patchSize (int) : default=41 scale_min (float) : default=-1.0 scale_max (float) : default=-1.0 rotation_invariance (bool) : default=False affine_invariance (bool) : default=True

Returns (kpts, vecs)

Return type tuple

CommandLine: python -m pyhesaff detect_feats python -m pyhesaff detect_feats –show python -m pyhesaff detect_feats –show –fname star.png python -m pyhesaff detect_feats –show –fname zebra.png python -m pyhesaff detect_feats –show –fname astro.png python -m pyhesaff detect_feats –show –fname carl.jpg

python -m pyhesaff detect_feats –show –fname astro.png –ri python -m pyhesaff detect_feats –show –fname astro.png –ai

python -m pyhesaff detect_feats –show –fname easy1.png –no-ai python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –numberOfScales=1 –verbose python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –scale-max=100 –verbose python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –scale-min=20 –verbose python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –scale-min=100 –verbose python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –scale-max=20 –verbose

python -m vtool.test_constrained_matching visualize_matches –show python -m vtool.tests.dummy test-data_ratio_matches –show

python -m pyhesaff detect_feats –show –fname easy1.png –ai –verbose –rebuild-hesaff –scale-min=35 –scale-max=40 –no-rmbuild

python -m pyhesaff detect_feats –show –fname easy1.png –ai –verbose –scale-min=35 –scale-max=40& python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –verbose –scale-min=35 –scale-max=40& python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –verbose –scale-max=40 –darken .5 python -m pyhesaff detect_feats –show –fname easy1.png –no-ai –verbose –scale-max=30 –darken .5 python -m pyhesaff detect_feats –show –fname easy1.png –ai –verbose –scale-max=30 –darken .5

DENSE KEYPOINTS python -m pyhesaff detect_feats –show –fname astro.png –no-affine-invariance –numberOfScales=1 –maxPyramidLevels=1 –use_dense –dense_stride=64 python -m pyhesaff detect_feats –show –fname astro.png –no-affine-invariance –numberOfScales=1 –maxPyramidLevels=1 –use_dense –dense_stride=64 –rotation-invariance python -m pyhesaff detect_feats –show –fname astro.png –affine-invariance –numberOfScales=1 –maxPyramidLevels=1 –use_dense –dense_stride=64 python -m pyhesaff detect_feats –show –fname astro.png –no-affine-invariance –numberOfScales=3 –maxPyramidLevels=2 –use_dense –dense_stride=32 python -m pyhesaff detect_feats –show –only_count=False

Example

```
>>> # ENABLE_DOCTEST
>>> # Test simple detect
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> TAU = 2 * np.pi
```

(continues on next page)

(continued from previous page)

```
>>> fpath = grab_test_imgpath(ub.argval('--fname', default='astro.png'))
>>> theta = float(ub.argval('--theta', 0))  # TAU * 3 / 8)
>>> img_fpath = vt.rotate_image_ondisk(fpath, theta)
>>> kwargs = argparse_hesaff_params()
>>> print('kwargs = %r' % (kwargs,))
>>> (kpts, vecs) = detect_feats(img_fpath, **kwargs)
>>> # Show keypoints
>>> # xdoctest: +REQUIRES(--show)
>>> imgBGR = vt.imread(img_fpath)
>>> # take a random stample
>>> frac = ub.argval('--frac', default=1.0)
>>> print('frac = %r' % (frac,))
>>> idxs = vecs[0:int(len(vecs) * frac]
>>> vecs, kpts = vecs[idxs], kpts[idxs]
>>> default_showkw = dict(ori=False, ell=True, ell_linewidth=2,
>>>                       ell_alpha=.4, ell_color='distinct')
>>> print('default_showkw = %r' % (default_showkw,))
>>> #showkw = ut.argparse_dict(default_showkw)
>>> #from wbia import plottool as pt
>>> #pt.interact_keypoints.ishow_keypoints(imgBGR, kpts, vecs, **showkw)
>>> #pt.show_if_requested()
```

pyhesaff._pyhesaff.**detect_feats2**(*img_or_fpath*, ***kwargs*)

General way of detecting from either an fpath or ndarray

Parameters **img_or_fpath** (*str or ndarray*) – file path string

Returns tuple

pyhesaff._pyhesaff.**detect_feats_in_image**(*img*, ***kwargs*)

Takes a preloaded image and detects keypoints and descriptors

Parameters **img** (*ndarray[uint8_t, ndim=2]*) – image data, should be in BGR or grayscale

Returns (kpts, vecs)

Return type tuple

CommandLine: python -m pyhesaff detect_feats_in_image –show python -m pyhesaff detect_feats_in_image –rebuild-hesaff –show –no-rmbuild

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> img_fpath = grab_test_imgpath('astro.png')
>>> img= vt.imread(img_fpath)
>>> (kpts, vecs) = detect_feats_in_image(img)
>>> # xdoctest: +REQUIRES(--show)
>>> from wbia import plottool as pt
>>> pt.interact_keypoints.ishow_keypoints(img, kpts, vecs, ori=True,
>>>                                       ell_alpha=.4, color='distinct')
>>> pt.set_figtitle('Detect Kpts in Image')
>>> pt.show_if_requested()
```

pyhesaff._pyhesaff.**detect_feats_list**(*image_paths_list*, ***kwargs*)

Parameters **image_paths_list** (*list*) – A list of image paths

Returns (kpts_list, vecs_list) A tuple of lists of keypoints and decsriptors

Return type tuple

Kwargs: numberOfScales (int) : default=3 threshold (float) : default=5.33333333333 edgeEigenValueRatio (float) : default=10.0 border (int) : default=5 maxIterations (int) : default=16 convergenceThreshold (float) : default=0.05 smmWindowSize (int) : default=19 mrSize (float) : default=5.19615242271 spatialBins (int) : default=4 orientationBins (int) : default=8 maxBinValue (float) : default=0.2 initialSigma (float) : default=1.6 patchSize (int) : default=41 scale_min (float) : default=-1.0 scale_max (float) : default=-1.0 rotation_invariance (bool) : default=False

CommandLine: python -m pyhesaff._pyhesaff detect_feats_list –show

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> fpath = grab_test_imgpath('astro.png')
>>> image_paths_list = [grab_test_imgpath('carl.jpg'), grab_test_imgpath('star.png
↪'), fpath]
>>> (kpts_list, vecs_list) = detect_feats_list(image_paths_list)
>>> #print((kpts_list, vecs_list))
>>> # Assert that the normal version agrees
>>> serial_list = [detect_feats(fpath) for fpath in image_paths_list]
>>> kpts_list2 = [c[0] for c in serial_list]
>>> vecs_list2 = [c[1] for c in serial_list]
>>> diff_kpts = [kpts - kpts2 for kpts, kpts2 in zip(kpts_list, kpts_list2)]
>>> diff_vecs = [vecs - vecs2 for vecs, vecs2 in zip(vecs_list, vecs_list2)]
>>> assert all([x.sum() == 0 for x in diff_kpts]), 'inconsistent results'
>>> assert all([x.sum() == 0 for x in diff_vecs]), 'inconsistent results'
```

pyhesaff._pyhesaff.**detect_num_feats_in_image**(*img*, ***kwargs*)

Just quickly returns how many keypoints are in the image. Does not attempt to return or store the values.

It is a good idea to turn off things like ai and ri here.

Parameters **img** (*ndarray[uint8_t, ndim=2]*) – image data

Returns nKpts

Return type int

ISSUE: there seems to be an inconsistency for jpgs between this and detect_feats

CommandLine: python -m pyhesaff detect_num_feats_in_image:0 –show python -m pyhesaff detect_num_feats_in_image:1 –show python -m xdoctest pyhesaff detect_num_feats_in_image:0

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> img_fpath = grab_test_imgpath('zebra.png')
>>> img = vt.imread(img_fpath)
```

(continues on next page)

(continued from previous page)

```
>>> nKpts = detect_num_feats_in_image(img)
>>> kpts, vecs = detect_feats_in_image(img)
>>> #assert nKpts == len(kpts), 'inconsistency'
>>> result = ('nKpts = %s' % (ub.repr2(nKpts),))
>>> print(result)
```

Example

```
>>> # TIMEDOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> setup = ub.codeblock(
    '''
    import vtool as vt
    import pyhesaff
    img_fpath = grab_test_imgpath('carl.jpg')
    img = vt.imread(img_fpath)
    ''')
>>> stmt_list = [
>>>     'pyhesaff.detect_feats_in_image(img)',
>>>     'pyhesaff.detect_num_feats_in_image(img, affine_invariance=False)',
>>>     'pyhesaff.detect_num_feats_in_image(img)',
>>> ]
>>> iterations = 30
>>> verbose = True
>>> #ut.timeit_compare(stmt_list, setup=setup, iterations=iterations,
>>> #                  verbose=verbose, assertsame=False)
```

pyhesaff._pyhesaff.**extract_desc_from_patches**(*patch_list*)

Careful about the way the patches are extracted here.

Parameters **patch_list** (*ndarray[ndims=3]*) –

CommandLine: python -m pyhesaff extract_desc_from_patches –rebuild-hesaff –no-rmbuild python -m pyhesaff extract_desc_from_patches –rebuild-hesaff –no-rmbuild –show python -m pyhesaff extract_desc_from_patches:1 –show

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> img_fpath = grab_test_imgpath(ub.argval('--fname', default='astro.png'))
>>> # First extract keypoints normally
>>> (orig_kpts_list, orig_vecs_list) = detect_feats(img_fpath)
>>> # Take 9 keypoints
>>> img = vt.imread(img_fpath)
>>> kpts_list = orig_kpts_list[1::len(orig_kpts_list) // 9]
>>> vecs_list = orig_vecs_list[1::len(orig_vecs_list) // 9]
>>> # Extract the underlying grayscale patches (using different patch_size)
>>> patch_list_ = np.array(vt.get_warped_patches(img, kpts_list, patch_
↪size=64)[0])
>>> patch_list = np.array(vt.convert_image_list_colorspace(patch_list_, 'gray'))
>>> # Extract descriptors from the patches
>>> vecs_array = extract_desc_from_patches(patch_list)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> img_fpath = grab_test_imgpath(ub.argval('--fname', default='astro.png'))
>>> # First extract keypoints normally
>>> (orig_kpts_list, orig_vecs_list) = detect_feats(img_fpath)
>>> # Take 9 keypoints
>>> img = vt.imread(img_fpath)
>>> kpts_list = orig_kpts_list[1::len(orig_kpts_list) // 9]
>>> vecs_list = orig_vecs_list[1::len(orig_vecs_list) // 9]
>>> # Extract the underlying grayscale patches
>>> #patch_list_ = np.array(vt.get_warped_patches(img, kpts_list)[0])
>>> #patch_list = np.array(vt.convert_image_list_colorspace(patch_list_, 'gray'))
>>> patch_list = extract_patches(img, kpts_list)
>>> patch_list = np.round(patch_list).astype(np.uint8)
>>> # Currently its impossible to get the correct answer
>>> # TODO: allow patches to be passed in as float32
>>> # Extract descriptors from those patches
>>> vecs_array = extract_desc_from_patches(patch_list)
>>> # Comparse to see if they are close to the original descriptors
>>> errors = vt.L2_sift(vecs_list, vecs_array)
>>> print('Errors: %r' % (errors,))
>>> # xdoctest: +REQUIRES(--show)
>>> from wbia import plottool as pt
>>> ax = pt.draw_patches_and_sifts(patch_list, vecs_array, pnum=(1, 2, 1))
>>> ax.set_title('patch extracted')
>>> ax = pt.draw_patches_and_sifts(patch_list, vecs_list, pnum=(1, 2, 2))
>>> ax.set_title('image extracted')
>>> pt.show_if_requested()
```

pyhesaff._pyhesaff.**extract_patches**(*img_or_fpath*, *kpts*, ***kwargs*)

Extract patches used to compute SIFT descriptors.

Parameters

- **img_or_fpath** (*ndarray or str*) –
- **kpts** (*ndarray[float32_t, ndim=2]*) – keypoints

CommandLine: python -m pyhesaff extract_patches:0 –show python -m pyhesaff extract_vecs:1 –fname=astro.png python -m pyhesaff extract_vecs:1 –fname=patsy.jpg –show python -m pyhesaff extract_vecs:1 –fname=carl.jpg python -m pyhesaff extract_vecs:1 –fname=zebra.png

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> kwargs = {}
>>> img_fpath = grab_test_imgpath('carl.jpg')
>>> img = vt.imread(img_fpath)
>>> img_or_fpath = img
>>> kpts, vecs1 = detect_feats(img_fpath)
>>> kpts = kpts[1::len(kpts) // 9]
>>> vecs1 = vecs1[1::len(vecs1) // 9]
```

(continues on next page)

(continued from previous page)

```
>>> cpp_patch_list = extract_patches(img, kpts)
>>> py_patch_list_ = np.array(vt.get_warped_patches(img_or_fpath, kpts, patch_
↪size=41)[0])
>>> py_patch_list = np.array(vt.convert_image_list_colorspace(py_patch_list_,
↪'gray'))
>>> # xdoctest: +REQUIRES(--show)
>>> from wbia import plottool as pt
>>> ax = pt.draw_patches_and_sifts(cpp_patch_list, None, pnum=(1, 2, 1))
>>> ax.set_title('C++ extracted')
>>> ax = pt.draw_patches_and_sifts(py_patch_list, None, pnum=(1, 2, 2))
>>> ax.set_title('Python extracted')
>>> pt.show_if_requested()
```

pyhesaff._pyhesaff.**extract_vecs**(*img_fpath*, *kpts*, ***kwargs*)

Extract SIFT descriptors at keypoint locations

Parameters

- **img_fpath** (*str*) –
- **kpts** (*ndarray[float32_t, ndim=2]*) – keypoints

Returns vecs - descriptor vectors

Return type ndarray[uint8_t, ndim=2]

CommandLine: python -m pyhesaff extract_vecs:0 python -m pyhesaff extract_vecs:1 –fname=astro.png python -m pyhesaff extract_vecs:1 –fname=patsy.jpg –show python -m pyhesaff extract_vecs:1 –fname=carl.jpg python -m pyhesaff extract_vecs:1 –fname=zebra.png

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> img_fpath = grab_test_imgpath('carl.jpg')
>>> kpts = vt.demodata.get_dummy_kpts()
>>> vecs = extract_vecs(img_fpath, kpts)
>>> result = ('vecs = %s' % (str(vecs),))
>>> print(result)
```

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> import vtool as vt
>>> img_fpath = grab_test_imgpath(ub.argval('--fname', default='astro.png'))
>>> # Extract original keypoints
>>> kpts, vecs1 = detect_feats(img_fpath)
>>> # Re-extract keypoints
>>> vecs2 = extract_vecs(img_fpath, kpts)
>>> # Descriptors should be the same
>>> errors = vt.L2_sift(vecs1, vecs2)
>>> errors_index = np.nonzero(errors)[0]
>>> print('errors = %r' % (errors,))
```

(continues on next page)

(continued from previous page)

```
>>> print('errors_index = %r' % (errors_index,))
>>> print('errors.sum() = %r' % (errors.sum(),))
>>> # VISUALIZTION
>>> # xdoctest: +REQUIRES(--show)
>>> from wbia import plottool as pt
>>> # Extract the underlying grayscale patches
>>> img = vt.imread(img_fpath)
>>> #patch_list_ = np.array(vt.get_warped_patches(img, kpts)[0])
>>> #patch_list = np.array(vt.convert_image_list_colorspace(patch_list_, 'gray'))
>>> patch_list = extract_patches(img, kpts)
>>> pt.interact_keypoints.ishow_keypoints(img_fpath, kpts[errors_index],␣
↪vecs1[errors_index], fnum=1)
>>> ax = pt.draw_patches_and_sifts(patch_list[errors_index], vecs1[errors_index],␣
↪pnum=(1, 2, 1), fnum=2)
>>> ax.set_title('patch extracted')
>>> ax = pt.draw_patches_and_sifts(patch_list[errors_index], vecs2[errors_index],␣
↪pnum=(1, 2, 2), fnum=2)
>>> ax.set_title('image extracted')
>>> pt.set_figtitle('Error Keypoints')
>>> pt.show_if_requested()
```

pyhesaff._pyhesaff.**get_cpp_version**()

Returns cpp_version

Return type int

CommandLine: python -m pyhesaff get_cpp_version

Example

```
>>> # ENABLE_DOCTEST
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> cpp_version = get_cpp_version()
>>> isdebug = get_is_debug_mode()
>>> print('cpp_version = %r' % (cpp_version,))
>>> print('isdebug = %r' % (isdebug,))
>>> assert cpp_version == 3, 'cpp version mimatch'
```

pyhesaff._pyhesaff.**get_hesaff_default_params**()

pyhesaff._pyhesaff.**get_is_debug_mode**()

pyhesaff._pyhesaff.**grab_test_imgpath**(*p*)

pyhesaff._pyhesaff.**str_list_t**
alias of pyhesaff._pyhesaff.LP_c_char_p

pyhesaff._pyhesaff.**test_rot_invar**()

CommandLine: python -m pyhesaff test_rot_invar –show –rebuild-hesaff –no-rmbuild python -m pyhesaff test_rot_invar –show –nocpp

python -m vtool.tests.demodata testdata_ratio_matches –show –ratio_thresh=1.0 –rotation_invariance –rebuild-hesaff python -m vtool.tests.demodata testdata_ratio_matches –show –ratio_thresh=1.1 –rotation_invariance –rebuild-hesaff

Example

```
>>> # xdoctest: +REQUIRES(module:wbia)
>>> from pyhesaff._pyhesaff import *  # NOQA
>>> test_rot_invar()
```

pyhesaff._pyhesaff.**vtool_adapt_rotation**(*img_fpath*, *kpts*)
rotation invariance in python

1.4 pyhesaff.ctypes_interface module

pyhesaff.ctypes_interface.**find_lib_fpath**(*libname*, *root_dir*, *recurse_down=True*, *verbose=False*)
Search for the library

pyhesaff.ctypes_interface.**get_lib_dpath_list**(*root_dir*)
input <root_dir>: deepest directory to look for a library (dll, so, dylib) returns <libnames>: list of plausible directories to look.

pyhesaff.ctypes_interface.**get_lib_fname_list**(*libname*)

Parameters **libname** (*str*) – library name (e.g. 'hesaff', not 'libhesaff')

Returns libnames - list of plausible library file names

Return type list

CommandLine: python -m pyhesaff.ctypes_interface get_lib_fname_list

Example

```
>>> from pyhesaff.ctypes_interface import *  # NOQA
>>> libname = 'hesaff'
>>> libnames = get_lib_fname_list(libname)
>>> import ubelt as ub
>>> print('libnames = {}'.format(ub.repr2(libnames)))
```

pyhesaff.ctypes_interface.**get_plat_specifier**()
Standard platform specifier used by distutils

pyhesaff.ctypes_interface.**load_clib**(*libname*, *root_dir*)
Searches for a library matching libname and loads it

Parameters

- **libname** – library name (e.g. 'hesaff', not 'libhesaff')
- **root_dir** – the deepest directory searched for the library file (dll, dylib, or so).

Returns a ctypes object used to interface with the library

Return type clib

1.5 Module contents

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index